**UFRRJ**

**INSTITUTO DE CIÊNCIAS EXATAS**

**MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL - PROFMAT**

**DISSERTAÇÃO**

# GRAFO E O PROBLEMA DO CAMINHO MÍNIMO: ALGORITMO E PROGRAMAÇÃO EM PASCAL

**MARCÍLIO DANIEL DE CASTRO PEREIRA**

**SEROPÉDICA**

**2022**



**UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO**

**INSTITUTO DE CIÊNCIAS EXATAS**

**MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL - PROFMAT**

# GRAFO E O PROBLEMA DO CAMINHO MÍNIMO: ALGORITMO E PROGRAMAÇÃO EM PASCAL

**MARCÍLIO DANIEL DE CASTRO PEREIRA**

**Orientador: Prof. VINÍCIUS LEAL DO FORTE**

Dissertação submetida como requisito parcial para a aprovação no Curso de Pós-Graduação em Mestrado Profissional em Matemática em Rede Nacional – PROFMAT, Área de Concentração em Matemática.

**SEROPÉDICA**

**2022**

Universidade Federal Rural do Rio de Janeiro
Biblioteca Central / Seção de Processamento Técnico

Ficha catalográfica elaborada
com os dados fornecidos pelo(a) autor(a)

P436g

Pereira, Marcílio Daniel de Castro, 1968-
GRAFO E O PROBLEMA DO CAMINHO MÍNIMO: ALGORITMO E PROGRAMAÇÃO EM PASCAL / Marcílio Daniel de Castro Pereira. - Rio de Janeiro, 2022.
152 f.: il.

Orientador: Vinícius Leal do Forte.
Dissertação(Mestrado). -- Universidade Federal Rural do Rio de Janeiro, Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, 2022.

1. Matemática. 2. Ensino Médio. 3. Teoria dos Grafos. 4. Caminhos Mínimos e Algoritmos de Dijkstra e Bellman-Ford. 5. Algoritmo e Programação em Pascal. I. Forte, Vinícius Leal do, 1985-, orient. II Universidade Federal Rural do Rio de Janeiro. Mestrado Profissional em Matemática em Rede Nacional PROFMAT III. Título.

**MINISTÉRIO DA EDUCAÇÃO**
**UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO**
**INSTITUTO DE CIÊNCIAS EXATAS**

**HOMOLOGAÇÃO Nº 2/2022 - ICE (12.28.01.23)**

**Nº do Protocolo: 23083.012984/2022-18**

**Seropédica-RJ, 28 de fevereiro de 2022.**

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO

INSTITUTO DE CIÊNCIAS EXATAS

PROGRAMA DE PÓS-GRADUAÇÃO EM MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL – PROFMAT

MARCÍLIO DANIEL DE CASTRO PEREIRA

Dissertação submetida como requisito parcial para a obtenção de grau de **Mestre**, no Programa de Pós-Graduação em Mestrado Profissional em Matemática em Rede Nacional - PROFMAT, área de Concentração em Matemática.

DISSERTAÇÃO APROVADA EM 16/02/2022

**Conforme deliberação número 001/2020 da PROPPG, de 30/06/2020,** tendo em vista a implementação de trabalho remoto e durante a vigência do período de suspensão das atividades acadêmicas presenciais, em virtude das medidas adotadas para reduzir a propagação da pandemia de Covid-19, nas versões finais das teses e dissertações as assinaturas originais dos membros da banca examinadora poderão ser substituídas por documento(s) com assinaturas eletrônicas. Estas devem ser feitas na própria folha de assinaturas, através do SIPAC, ou do Sistema Eletrônico de Informações (SEI) e neste caso a folha com a assinatura deve constar como anexo ao final da tese / dissertação.

Vinicius Leal do Forte. Dr. UFRRJ (Orientador, Presidente da Banca)

Montauban Moreira de Oliveira Júnior. Dr. UFRRJ (membro interno)

Marilis Bahr Karam Venceslau. Dra. Colégio Pedro II (membro externo)

***(Assinado digitalmente em 28/02/2022 11:31)***
MONTAUBAN MOREIRA DE OLIVEIRA JUNIOR
*PROFESSOR DO MAGISTERIO SUPERIOR*
*DeptM (12.28.01.00.00.00.63)*
*Matrícula: 1633341*

***(Assinado digitalmente em 03/03/2022 15:43)***
VINICIUS LEAL DO FORTE
*PROFESSOR DO MAGISTERIO SUPERIOR*
*DeptM (12.28.01.00.00.00.63)*
*Matrícula: 2620902*

***(Assinado digitalmente em 01/03/2022 16:04)***
MARILIS BAHR KARAM VENCESLAU
*ASSINANTE EXTERNO*
*CPF: 016.572.107-31*

Para verificar a autenticidade deste documento entre em https://sipac.ufrrj.br/public/documentos/index.jsp informando seu número: **2**, ano: **2022**, tipo: **HOMOLOGAÇÃO**, data de emissão: **28/02/2022** e o código de verificação: **2c9c93c979**

Dedico à minha avó Nanzinha (*in memorian*) e à minha mãe Gessy, pilares da construção do meu caráter e saber.

## AGRADECIMENTOS

Gostaria de agradecer a Deus pela perseverança que sempre me deu para seguir meus objetivos. À minha esposa Alpha e à minha filha Letícia, privadas da minha companhia em diversos momentos do curso, que foram o esteio para que eu aqui chegasse. Aos meus colegas de turma do PROFMAT, muitos na idade da minha filha, pela troca de experiência e companheirismo, fundamentais no processo de aprendizagem. Aos professores, pela dedicação, estímulo e amizade e, em especial, ao professor Vinicius, por ter aceitado o convite para ser meu orientador e, através de suas brilhantes sugestões, ter contribuído para qualificar este trabalho. Ao ensino público, que fez parte de toda a minha história, fundamental para a minha formação como pessoa humana.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

*This study was financed in part by the* Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – *Finance Code* 001.

## RESUMO

Pereira, Marcílio Daniel de Castro. **Grafo e o problema do caminho mínimo: algoritmo e programação em Pascal. 2022.** – 152 páginas. Dissertação (Mestrado Profissional em Matemática em Rede Nacional – PROFMAT). Instituto de Ciências Exatas, Departamento de Matemática, Universidade Federal Rural do Rio de Janeiro, Seropédica, RJ, 2022.

Este trabalho tem como objetivo trazer uma proposta de ensino para as turmas de ensino médio abordando um tópico específico da Teoria dos Grafos que trata de caminhos mínimos e dos Algoritmos de Dijkstra e Bellman-Ford utilizados para determiná-los. Como os algoritmos são algo abstrato, tendo grande importância quando transformados em um programa através de uma linguagem de programação, optou-se em trazer para o presente trabalho as noções da linguagem de Programação Pascal, linguagem de fácil compreensão para programadores iniciantes. Neste trabalho, os alunos terão contato com a linguagem de programação Pascal, conhecerão a origem e a importância da Teoria dos Grafos e terão acesso às noções básicas da teoria, pressupostos para o estudo dos caminhos mínimos, de seus algoritmos e dos programas em Pascal a eles relacionados. Foram propostas atividades relacionadas ao cotidiano dos alunos, com o fim de propiciar o domínio dos conceitos básicos tanto da linguagem de programação Pascal como da Teoria dos Grafos, em especial, dos algoritmos voltados para resolução dos problemas envolvendo caminhos mínimos. Espera-se, por fim, uma mudança comportamental do aluno na busca pelo novo, demonstrando a sua curiosidade em conhecer mais sobre a Teoria dos Grafos e fazendo uso do Pascal para criar seus próprios programas no auxílio de suas tarefas diárias.



**Palavras-Chave:** Matemática, Ensino Médio, Teoria dos Grafos, Caminhos Mínimos, Algoritmo, Pascal, Dijkstra, Bellman-Ford.

## ABSTRACT

Pereira, Marcílio Daniel de Castro. **Graph and the shortest path problem: algorithm and programming in Pascal. 2022.** - 152 pages. Dissertation (Professional Master in Mathematics in National Network - PROFMAT). Institute of Exact Sciences, Department of Mathematics, Federal Rural University of Rio de Janeiro, Seropédica, RJ, 2022.

This work aims to bring a teaching proposal to high school classes addressing a specific topic of Graph Theory that deals with shortest paths and the Dijkstra and Bellman-Ford algorithms used to determine them. As algorithms are something abstract, having great importance when transformed into a program through a programming language, it was decided to bring to the present work notions of the Pascal programming language, a language of easy understanding for beginning programmers. In this work, students will have contact with the Pascal programming language, will know the origin and importance of Graph Theory and will have access to the theory's basic notion, presuppositions for the study of shortest paths, their algorithms and related Pascal programs. Activities related to the daily lives of students were proposed, in order to provide mastery of the basic concepts of both the Pascal programming language and Graph theory, in particular, algorithms aimed at solving problems involving shortest paths. Finally, a behavioral change in the student is expected in the search for the new, demonstrating their curiosity to know more about Graph Theory and making use of Pascal to create their own programs to help with their daily tasks.



**Keywords:** Mathematics, High School, Graphs Theory, Shortest Paths, Algorithm, Pascal, Dijkstra, Bellman-Ford.

## LISTA DE FIGURAS ‘

## LISTA DE QUADROS

**SUMÁRIO**

## INTRODUÇÃO

Analisando trabalhos desenvolvidos pela comunidade acadêmica, pode-se afirmar que o estudo dos Grafos possui uma grande relevância para a solução de problemas envolvendo o cotidiano da sociedade brasileira e mundial. A Teoria dos Grafos soluciona problemas envolvendo, entre outros, o controle de semáforos de uma cidade (HERNANDES, 2007), o itinerário dos caminhões para a coleta de lixo em um bairro da cidade (SILVA, 2020), a otimização de fluxo em uma rede de computadores (MOTTA & BRITO, 2017), o planejamento de rotas de voos por uma companhia aérea (FRANCO, 2019), a análise do desenvolvimento humano (ALMEIDA & CUNHA, 2003), o planejamento de interligação de rede elétrica (RESE *et al.*, 2017) e o sequenciamento de DNA (BOAVENTURA NETTO & JURKIEWICZ, 2009).

Optou-se aqui por abordar caminhos mínimos, tópico específico da teoria, bastante utilizado e aplicado na área de Pesquisa Operacional, e os Algoritmos de Dijkstra e Bellman-Ford utilizados para determiná-los. Problemas práticos que ocorrem nas áreas de transportes e logística podem ser solucionados utilizando os algoritmos. Através do seu estudo, solucionam-se problemas envolvendo duas localidades em que se quer saber, por exemplo, o caminho mais curto entre elas ou o caminho em que o percurso será realizado em menor tempo como o apresentado por Oliveira *et al*. (2020), no 11º Congresso de Logística das Faculdades de Tecnologia do Centro Estadual de Educação Tecnológica Paula Souza. Através de um estudo em que se aplicou o Algoritmo de Dijkstra, a rota encontrada pelo algoritmo representou uma redução para uma empresa sediada em Angra dos Reis de 16,38% no tempo praticado na antiga rota de entrega de mercadorias a clientes e de 21,65% na quilometragem percorrida, correspondendo a um ganho financeiro para a empresa, em face da redução dos custos associados a transporte (combustível, mão de obra e serviços).

Pensou-se em inserir a Teoria dos Grafos e, mais especificamente, o estudo de caminhos mínimos e seus algoritmos, no itinerário formativo[1] a ser

[1] Definido no art. 4º da Lei nº 13.415/2017, que alterou o art. 36 da Lei de Diretrizes e Bases da Educação Nacional. Regulamentado pela Resolução nº 3/2018 do Conselho Nacional de Educação.

ofertado aos alunos pelas escolas. A partir do ano de 2022, o curriculum do ensino médio será formado por uma base comum a todos os estudantes, onde estudarão as disciplinas tradicionais, e uma outra parte chamada de itinerário formativo, voltado para quatro grandes áreas do conhecimento, sendo uma delas Matemática e suas Tecnologias, e para a área técnica e profissional. As escolas poderão ofertar disciplinas, cursos, laboratórios, grupos de estudo, projetos etc., com o fim de aprofundar o conhecimento de um conteúdo já estudado pelos alunos ou ampliar esse conhecimento através da abordagem de um conteúdo novo. A Teoria dos Grafos se insere aqui.

Pensou-se, também, em inserir no trabalho a educação digital, tema transversal[2] e fundamental para o desenvolvimento do raciocínio lógico, propiciando mudanças na forma de produzir conhecimento, armazená-lo e transmiti-lo. Como os algoritmos são algo abstrato, passando a ter uma grande importância quando de sua codificação para um programa de computação, optou-se em trazer para o presente trabalho as noções da linguagem de Programação Pascal, linguagem de fácil compreensão para programadores iniciantes (CASTILHO *et al.,* 2020). Prevê-se o seu estudo no itinerário formativo de Matemática e suas tecnologias, uma vez que nele está previsto o ensino de tecnologias e conteúdos digitais como robótica, automação, criação de jogos eletrônicos, inteligência artificial e **programação**.

Assim, tem-se a possibilidade de reunir em um só módulo ou disciplina a ser ofertado como itinerário formativo pelas escolas o estudo de caminhos mínimos, de seus algoritmos e dos programas a eles relacionados.

O trabalho de dissertação é dividido em quatro capítulos. O primeiro é dedicado ao estudo da criação de algoritmos e de um programa em Pascal. Serão apresentados dentre outros a estrutura de um Programa Pascal, as noções de fluxo de execução de um programa, os comandos de manipulação de dados e iteração com o usuário, as expressões da linguagem para a realização de cálculos aritméticos e lógicos, os comandos da linguagem que modificam o fluxo de execução do programa, as funções das sub-rotinas e outros tipos de estruturas

[2] Definido no art. 26 da Lei de Diretrizes e Bases da Educação Nacional. Regulamentado pela Resolução nº 3/2018 do Conselho Nacional de Educação.

como os vetores, as matrizes e os registros. Ao final do capítulo, pretende-se que o aluno não só possa compreender as instruções de um programa em Pascal, mas também possa criar os seus próprios programas. É através dessa compreensão, que serão apresentados os programas relacionados aos algoritmos para a busca do caminho mínimo em um grafo.

O segundo capítulo é dedicado ao estudo dos grafos. Serão apresentadas a história de sua origem, as áreas de aplicação e as noções básicas da teoria.

O terceiro capítulo é dedicado ao estudo do caminho mínimo em um grafo, onde serão apresentados os Algoritmos de Dijkstra e Bellman-Ford utilizados para determiná-lo e os programas escritos em Pascal a eles relacionados.

Por fim, o quarto e último capítulo é dedicado a atividades a serem feitas pelos alunos, de preferência em grupo, as quais visam à compreensão da linguagem Pascal abordada no capítulo 1, a fixação dos conceitos relacionados a grafos abordados no capítulo 2 e a lógica da determinação dos caminhos mínimos em um grafo utilizando-se dos Algoritmos de Dijkstra e Bellman-Ford apresentados no capítulo 3 e dos programas a eles relacionados. Pretende-se que, através dos recursos computacionais disponíveis nas escolas[3] ou, ainda, do seu próprio smartphone, os alunos utilizem o Programa Pascal no auxílio à resolução das atividades.

Pretende-se, ao final, que o aluno possa identificar os casos concretos envolvendo caminhos mínimos e aplicar os algoritmos e respectivos programas para determiná-los.

[3] Através de Programas voltados para a educação digital, os entes da federação vêm criando políticas públicas direcionadas não só para o aperfeiçoamento dos professores e o ensino da Informática nas escolas, mas também para o acesso destes mesmos professores e alunos ao instrumental exigido na informática educativa como computadores, notebooks, tablets e internet. Dentre as políticas atuais do Governo Federal podemos citar o Programa Nacional de Tecnologia Educacional (Proinfo) e o Programa de Inovação Educação Conectada. O primeiro leva às escolas computadores, recursos digitais e conteúdos educacionais. Em contrapartida, estados, Distrito Federal e municípios devem garantir a estrutura adequada para receber os laboratórios e capacitar os educadores para uso das máquinas e tecnologias. O segundo apoia a universalização do acesso à internet de alta velocidade e fomenta o uso pedagógico de tecnologias digitais na Educação Básica, prevendo o direcionamento até o ano de 2024 a 100% dos alunos da educação básica.

Espera-se uma mudança comportamental do aluno na busca pelo novo, demonstrando a sua curiosidade em conhecer mais sobre a Teoria dos Grafos e fazendo uso do Pascal para criar seus próprios programas no auxílio de suas tarefas diárias.

# 1. INTRODUÇÃO À LINGUAGEM PASCAL

Neste capítulo apresentaremos as noções básicas de algoritmo, como construí-lo e como transformá-lo em um programa para ser executado pelo computador. Saber construir algoritmos ou mesmo programas de computador proporciona ao aluno o desenvolvimento do seu raciocínio lógico para a solução de problemas do cotidiano. Para o desenvolvimento dos programas, optou-se pela linguagem de programação PASCAL, criada inicialmente para ser uma linguagem para uso didático. Ela permite ensinar com clareza os principais conceitos envolvidos na programação estruturada de computadores (PEREIRA, 2018). É uma boa linguagem para ser usada por programadores iniciantes (CASTILHO *et al*., 2020).

Inicialmente, vamos definir o conceito de algoritmo. Para Eduardo Moreno e Héctor Ramírez, algoritmo é uma sequência finita e ordenada de passos para realizar uma tarefa de forma precisa (MORENO & RAMÍREZ, 2011, p.21, tradução nossa). Corroborando, Castilho *et al*. (2020, p.15) define algoritmo como “uma sequência extremamente precisa de instruções que, quando lida e executada por uma outra pessoa, produz o resultado esperado, isto é, a solução de um problema”.

Com isso, pode-se afirmar que os algoritmos fazem parte dos nossos hábitos diários. Para ilustrar, Moreno & Ramírez (2011) mostram a rotina de uma pessoa que se prepara todas as manhãs para o trabalho, ao executar uma série de passos como acordar, tomar banho, escovar os dentes, vestir-se, pentear-se, tomar café, pegar uma condução ou caminhar até o trabalho. Esta sequência ordenada forma um algoritmo.

Também formam um algoritmo os passos para o preparo de um bolo de laranja: misturar o açúcar, os ovos e a margarina. Bater bem até obter um creme claro. Adicionar à mistura o suco de laranja, o fermento e a farinha de trigo. Bater todos os ingredientes. Colocar a massa pronta em uma forma e levá-la ao forno pré-aquecido a 180ºC por 35 minutos.

Percebe-se que os algoritmos são formados por instruções, em sequência, cuja ordem deve ser respeitada por quem os executa, a fim de que o

objetivo proposto seja atingido. O êxito fará com que o algoritmo seja utilizado novamente por quem o deseje. Para Mathias (2017), as palavras “sequência”, “ordem”, “passos”, “objetivo” e “repetibilidade” jamais podem ser esquecidas na construção de um algoritmo.

Observe que o resultado esperado no preparo de um bolo de laranja é um bolo delicioso. Aqui nem sempre o resultado é atingido. O êxito no preparo do bolo dependerá do conhecimento de quem o está preparando. Como exemplo de um resultado inadequado ocorre quando o cozinheiro resolve abrir o forno com menos de 15 minutos do cozimento da massa. A probabilidade de o bolo solar é grande, repercutindo na qualidade do bolo. Todavia, quando o algoritmo é utilizado para a solução de problemas em matemática ou em computação, o resultado esperado será sempre o mesmo. Assim, um algoritmo construído para ler dois números digitados pelo usuário, somá-los e imprimir o seu resultado, utilizado por qualquer pessoa que digite os números 2 e 3, terá sempre como valor impresso o número 5.

Saliente-se que o algoritmo passa a ser útil, uma vez que é algo abstrato, quando ele é codificado em um programa por meio de uma linguagem de programação, onde será testado e concretizado em um processo computacional e, a partir de sua utilização, passará a ter as características de repetibilidade (MATHIAS, 2017). Dentre as linguagens de programação, pode-se citar o PASCAL, C, JAVA e PYTHON.

De acordo com Castilho *et al*. (2020), a base das linguagens de programação, em geral, é constituída por:

- noção de fluxo de execução de um programa;
- comandos da linguagem que manipulam os dados da memória e a interação com o usuário (atribuição, entrada e saída de dados);
- expressões da linguagem que permitem a realização de cálculos aritméticos e lógicos;
- comandos da linguagem que modificam o fluxo de execução de um programa.

Os algoritmos também são constituídos de forma semelhante.

O fluxo de execução do programa garante que cada instrução seja executada uma após a outra. Caso haja na construção do algoritmo ou do programa alguma estrutura condicional ou de repetição, o fluxo de execução poderá ser alterado ou desviado. A interação com o usuário ocorre com a inserção de dados pelo teclado (leitura de dados) ou pela saída de dados (escrita ou impressão na tela do monitor). A manipulação dos dados ocorre com os comandos de atribuição de igual ( = ), na construção do algoritmo, de dois-pontos seguidos de igual ( := ), na construção do programa em Pascal, destes ou de outro comando de atribuição quando se tratar de outra linguagem de programação. A realização de cálculos aritméticos e lógicos (booleanos) é a própria essência dos algoritmos e dos programas de computador.

Na Figura 1, à esquerda, encontra-se o algoritmo que calcula a média aritmética entre dois números inteiros digitados pelo usuário e a imprime na tela do monitor e, ao lado, à direita, o programa com as instruções em Pascal que codifica o algoritmo.

**Figura 1 –** Construção de um Algoritmo e de um Programa em Pascal

```
1  Algoritmo media_aritmetica;
2  variáveis
3  numero1, numero2: inteiro;
4     media: real;
5
6  inicio
7     escreva (‘Digite o 1º número: ‘);
8     leia (numero1);
9     escreva (Digite o 2º número: ‘);
10    leia (numero2);
11    media = (numero1 + numero2) / 2;
12    escreva (‘Média: ‘, media:8:2);
13 fim.
```

```
1  Program media_aritmetica;
2  var
3     numero1, numero2: integer;
4     media: real;
5
6  begin
7     write (‘Digite o 1.  numero: ‘);
8     readln (numero1);
9     write (Digite o 2. numero: ‘);
10    readln (numero2);
11    media := (numero1 + numero2) / 2;
12    writeln (‘Media: ‘, media:8:2);
13 end.
```

Fonte: Autor

Apresentar-se-á a seguir uma síntese da linguagem de programação Pascal, que dará ao aluno boas noções de como programar nesta linguagem. Caberá ao aluno o seu aprofundamento.

## 1.1. ESTRUTURA DE UM PROGRAMA EM PASCAL

Um programa em Pascal é dividido em três partes distintas: um cabeçalho, uma área de declarações, nem sempre obrigatória, e o corpo do programa, que é a área reservada ao programa principal (MANZANO & YAMATUMI, 2002).

### 1.1.1. CABEÇALHO

O cabeçalho é a área destinada a identificação do programa. Possui a seguinte sintaxe: *program <nome_do_programa>;*

No programa ilustrado na Figura 1, o cabeçalho é identificado como:

*Program media_aritmetica;*

### 1.1.2. ÁREA DE DECLARAÇÕES

A área de declarações é destinada a validar o uso de identificadores que não sejam predefinidos pelo Pascal como as palavras reservadas usadas no cabeçalho (ex: *program*), na área de declarações (ex: *var, integer, const, function*) e no corpo do programa (ex: *write, read, mod*), bem como o nome do programa. Esta área é subdividida em subáreas, a saber: rótulos (*label*), constantes (*const*), variáveis (*var*), procedimentos (*procedure*), funções (*function*) e outras estruturas de dados. Cada uma das subáreas possui um modo de ser referenciada pelo programa.

Inicialmente, abordar-se-á as subáreas variáveis e constantes. Com o maior conhecimento da linguagem, outras subáreas serão abordadas mais adiante.

#### 1.1.2.1. DECLARAÇÃO DE VARIÁVEIS

O programador, ao declarar uma variável, faz com que o computador reconheça aquele nome e passe a reservar um endereço de memória para armazenar o conteúdo daquela variável (MANZANO & YAMATUMI, 2002). As variáveis são declaradas através da palavra reservada ***var***, tendo a seguinte sintaxe:

*var*

*<identificador>: tipo;*

onde <*identificador*> representa um ou mais identificadores separados por vírgula ( , )*; e tipo,* o tipo das variáveis pré-definidas pelo compilador ou pelo próprio programador. Para cada linha do programa, deverá ser declarado um tipo diferente de dados. Os tipos pré-definidos pelo compilador são: inteiro (*integer*), real (*real*), caractere (*char*), literal (*string*) e lógico (*boolean*), além de outros tipos como vetor e matriz (*array*) e registro (*record*). No programa ilustrado na Figura 1, na área de declarações foram declaradas três variáveis, duas do tipo inteiro e uma do tipo real:

```
var
   numero1, numero2: integer;
   media: real;
```

### 1.1.1.2. DECLARAÇÃO DE CONSTANTES

Uma constante é um valor fixo que não se altera durante a execução do programa, contrariamente do que pode ocorrer com a variável. As constantes assumem o tipo do seu conteúdo, podendo, por exemplo, ser do tipo inteira, real ou lógica. São declaradas através da palavra reservada ***const***, tendo a seguinte sintaxe:

```
const
   <identificador> = valor;
```

onde <identificador> identifica a constante; e valor, o seu conteúdo. Para cada linha do programa, deverá ser declarado um tipo diferente de dados.

Exemplo:

```
const
   pi = 3.1416;
   nao = false;
```

### 1.1.3. CORPO DO PROGRAMA

Segundo Manzano & Yamatumi (2002), o corpo do programa é a área destinada a escrita do programa propriamente dito. Inicia-se com a palavra reservada ***begin*** e se encerra com a palavra reservada ***end*** seguida de ponto ***( . )***.

Entre as palavras reservadas *end* e *begin*, haverá uma sequência de comandos ou instruções, também chamada de bloco, que será processada pelo computador.

No exemplo ilustrado na Figura 1, o corpo do programa está representado pelas instruções:

```
begin
  write (‘Digite o 1. numero: ‘);
  readln (numero1);
  write (Digite o 2. numero: ‘);
  readln (numero2);
  media := (numero1 + numero2) / 2;
  writeln (‘Media: ‘, media:8:2);
end.
```

## 1.2. LEGIBILIDADE DO CÓDIGO

Para uma melhor visualização do programa, muitas vezes faz-se necessário inserir linhas em branco em seu código, as quais são ignoradas pelo compilador. Além disso, é importante inserirmos no programa comentários úteis a quem lê o programa, também ignorados pelo compilador. Eles devem vir precedidos de “duas barras (//)”.

O compilador não distingue comandos e identificadores em maiúsculo ou minúsculo.

Para que não haja erro na compilação dos dados, cada instrução deverá se limitar a uma única linha. Também para que não haja erro de compilação, as palavras que compõem as instruções em Pascal não podem estar acompanhadas de acentuação gráfica, exceto quando se referem a comentários.

Para melhor apresentação de um programa, o código deverá estar alinhado em colunas, identificando quais linhas do código estão vinculadas a um determinado comando.

Os programas ***Program media_aritmetica1*** e ***Program media_aritmetica2***, ilustrados na Figura 2, gerarão o mesmo código executável,

todavia para quem lê os programas, a disposição do código de ***Program media_aritmetica1*** torna mais compreensível o programa.

**Figura 2 –** Legibilidade do Código de um Programa

```
1  Program media_aritmetica1;
2  var
3     numero1, numero2: integer;
4     media: real;
5
6  begin
7     //leitura de dados a serem digitados
8     write (‘Digite o 1.  numero: ‘);
9     readln (numero1);
10
11    write (Digite o 2. numero: ‘);
12    readln (numero2);
13
14   //cálculo da média
15   media := (numero1 + numero2) / 2;
16
17   //impressão do resultado
18  writeln (‘Media: ‘, media);
18 end.
```

```
1  Program media_aritmetica2;
2  VAR
3  NUMERO1, numero2: integer;
4      media: real;
5  begin
6  write (‘Digite o 1.  numero: ‘);
7    readln (numero1);
8        write (Digite o 2. numero: ‘);
9    readln (numero2);
10 media := (numero1 + numero2) / 2;
11          writeln (‘Media: ‘, MEDIA);
12 end.
```

Fonte: Autor

## 1.3. BASE DA LINGUAGEM DE PROGRAMAÇÃO

A base das linguagens de programação, em geral, é constituída por comandos ou instruções de atribuição que manipulam os dados armazenados em memória, por comandos ou instruções de iteração com o usuário (entrada e saída de dados), pela noção de fluxo de execução de um programa e de sua modificação e pelas expressões da linguagem que permitem a realização de cálculos aritméticos e lógicos (CASTILHO *et al*., 2020).

### 1.3.1. COMANDOS DE ENTRADA E SAÍDA

Toda linguagem de programação precisa possuir mecanismos de iteração com o usuário. Através deles, o usuário poderá fornecer dados ao computador, para que este os processe e gere como saída aquilo que o programa se propõe a fazer, que, no caso do programa ilustrado na Figura 1, é a impressão na tela do monitor da média entre dois números inteiros. Esses mecanismos são chamados de

dispositivos de entrada (teclado, modem, leitores óticos e disco) e saída (vídeo, impressora e disco).

O dispositivo de entrada padrão usado pelo Pascal é o teclado. Através dele, os dados serão fornecidos ao computador, através do comando de entrada ***read*** ou ***readln****,* que irá processá-los, retornando uma saída através do monitor de vídeo, que é o dispositivo de saída padrão, através do comando de saída ***write*** ou ***writeln****.* Os comandos *readln* e *writeln* são variantes de *read* e *write* e fazem com que o cursor do monitor mude de linha após a exibição dos dados. No caso de *read* e *write* o cursor continuará na mesma linha.

Os comandos de entrada e saída possuem a seguinte sintaxe (CASTILHO *et al.*, 2009):

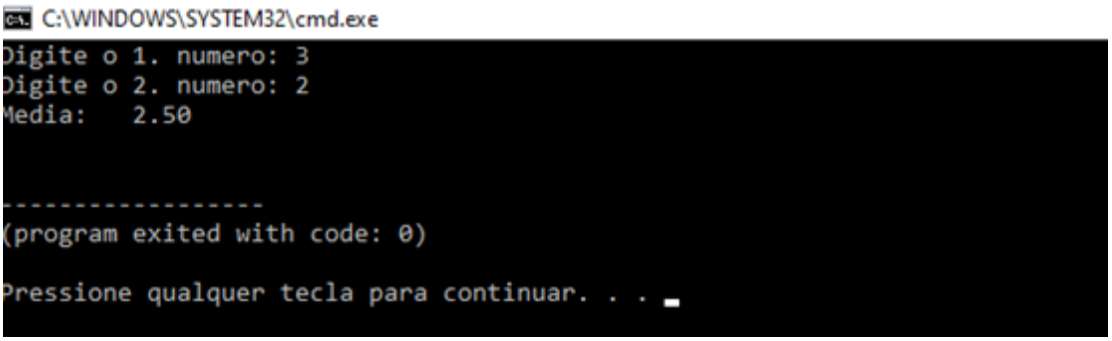
*read (<lista de variaveis>);*
*readln (<lista de variaveis>);*
*write (<lista>);*
*writeln (<lista>);*

em que <lista de variáveis> representa um identificador ou uma lista de identificadores de variáveis separados por vírgula ( , ), e <lista> é uma lista de elementos separados por vírgula, em que cada elemento pode ser um identificador de variável, uma expressão aritmética ou uma cadeia de caracteres apresentada entre aspas simples ( ‘ ), representando uma mensagem a ser mostrada na tela do monitor.

A Figura 3 ilustra a execução das instruções de entrada e saída do programa ***Program media_aritmetica***. Percebe-se que após o programa solicitar a entrada de dados, houve a digitação pelo usuário dos números inteiros 3 e 2. Em seguida há a realização do cálculo aritmético, para a impressão do seu resultado na tela do monitor. O resultado impresso foi formatado com 2 casas decimais, dentro de um espaço de 8 caracteres conforme código definido no programa principal:

*writeln (‘Media: ‘,* ***media:8:2****);*

**Figura 3 –** Execução do Programa *Program media_aritmetica*




Fonte: Autor

### 1.3.2. FLUXO DE EXECUÇÃO DO PROGRAMA

No exemplo ilustrado na Figura 1, o fluxo de execução do programa obedece a seguinte sequência ordenada: Inicialmente se executa a instrução ***write ('Digite o 1. numero: ')***, que escreverá na tela do monitor a expressão entre aspas simples ( ' ). O cursor do monitor continuará na mesma linha e piscará. Depois de concluída a 1ª instrução, executa-se a instrução ***readln (numero1)***. Esta instrução faz com que o programa guarde um dado, de mesmo tipo da variável declarada, a ser digitado pelo usuário. Após a digitação, o endereço de memória associado a variável *numero1* armazena o número inteiro digitado. O cursor do monitor mudará de linha. Depois de concluída a 2ª instrução, o programa executa a instrução ***write (Digite o 2. numero: ')***. O cursor do monitor continuará na mesma linha e piscará. Depois será executada a instrução ***readln (numero2)***. O endereço de memória associado à variável numero2 armazenará o segundo número inteiro digitado pelo usuário. Supondo que esses números sejam 3 e 2, o programa executará o cálculo da média entre os números conforme a expressão aritmética ***(numero1 + numero2) / 2***, e o resultado, através do comando de atribuição dois-pontos seguidos de igual ***( := )***, será armazenado no endereço de memória associado à variável ***media***. A última instrução do programa escreverá o resultado na tela do monitor reservando um espaço de 8 caracteres para a impressão do número real no formato de duas casas decimais.

### 1.3.3. REPETIÇÃO DE COMANDOS

Imagine que o programador queira ler 10 números reais e calcular a média destes números. Com o que vimos até o momento, o programador deveria inserir no corpo do programa principal, dez comandos ***read*** ou ***readln***. Note que o programa começaria a ficar extenso. Imagine agora a inserção de 100 números. Para resolver essa questão, o Pascal, como outras linguagens de programação, utiliza em sua estrutura comandos de repetição, também conhecida como *loop* ou laço. De acordo com Castilho *et al.* (2009, p.50), "os comandos de repetição servem para desviar o fluxo de execução de maneira que um determinado trecho do código possa ser repetido por um número determinado de vezes*"*. Essa repetição também pode ocorrer até que uma condição de seu encerramento seja satisfeita. O Pascal disponibiliza os comandos de repetição ***while ... do, for ... do*** e ***repeat ... until****.* Abordar-se-ão os dois primeiros.

a) *while ... do*

O comando possui as seguintes sintaxes:

```
while <condição> do
    <instrução>;

while <condição> do
begin
    <instruções>;
end;
```

```
enquanto <condição> faça
   <instrução ou instruções>;
fim enquanto;
```

No comando de repetição ***while ... do***, a instrução ou o bloco de instruções só será executado caso o teste lógico feito no início do *loop* seja verdadeiro (*TRUE*). Concluída a execução da única ou última instrução, novo teste lógico é feito. Enquanto a condição for satisfeita, a instrução ou o bloco de instruções será executado. No momento em que a condição deixa de ser satisfeita, o fluxo de execução do programa é desviado para a primeira instrução após o *loop*.

b) *for ... do*

O comando possui as seguintes sintaxes:

| | |
|---|---|
| *for <var>:= <Vi> to/downto <VF> do*<br>*<instrução>;*<br><br>*for <var>:= <Vi> to/downto <VF> do*<br>*begin*<br>*<instruções>;*<br>*end;* | para <var>:= <$V_i$> a <$V_F$> faça<br><instrução ou instruções>;<br>fim para; |

Usa-se o comando ***for ... do*** para executar uma instrução ou um bloco de instruções um número fixo de vezes, o qual é controlado pela variável de controle *var*, variando do valor inicial ($V_I$) ao final ($V_F$). O comando de repetição pode ser realizado de duas formas, com a palavra reservada ***to***, neste caso $V_I < V_F$, ou com a palavra reservada ***downto****,* em que $V_I > V_F$. Sempre o valor da variável de controle é comparado com $V_F$. No caso do uso do *to*, se *var* ≤ $V_F$, a instrução ou o bloco de instruções é executado e o valor de *var* é incrementado de uma unidade. No caso do uso de *downto*, se *var* ≥ $V_F$, a instrução ou o bloco de instruções é executado e o valor de *var* é decrementado de uma unidade. O *loop* é executado até que a variável de controle ultrapasse $V_F$, no caso do uso do *to,* ou se torne inferior a $V_F$, no caso do uso do *downto* (CASTILHO *et al.*, 2009).

### 1.3.4. DESVIO DE FLUXO CONDICIONAL

Imagine que se deseje criar um programa que calcule a quantidade de pessoas acima de 70 anos vacinadas contra a Covid-19 em um determinado município do Brasil. Observe que, na elaboração do programa, precisaremos comparar a idade de cada pessoa vacinada com o número inteiro 70, para tomar a decisão no sentido de alterar ou não a variável de controle (contador), criada para calcular essa quantidade.

Para Manzano & Yamatumi (2002, p.65), a solução para problemas semelhantes é “trabalhar uma nova instrução **if ... then** que tem por finalidade tomar uma decisão e efetuar um desvio de processamento, dependendo, é claro, da condição atribuída ser Verdadeira ou Falsa”.

Sendo a condição verdadeira, será executada a instrução ou o bloco de instruções relacionado àquela condição, não havendo alteração do fluxo de execução do programa. Sendo a condição falsa, o fluxo de execução do programa será desviado para a primeira instrução após a instrução ou o bloco de instruções considerado verdadeiro.

A instrução ***if ... then*** possui as seguintes sintaxes:

```
If <condição> then
    <instrução>;

If <condição> then
begin
    <instruções>;
end;
```

```
se <condição> então
    <instrução ou instruções>;
fim se;
```

Imagine, agora, que se deseje criar um programa que calcule a quantidade de pessoas acima de 70 anos ou abaixo desta idade vacinadas contra a Covid-19 em um determinado município do Brasil. Na elaboração do programa, continuamos precisando comparar a idade de cada pessoa vacinada com o número inteiro 70, para tomar a decisão no sentido de alterar duas variáveis de controle (contadores), criadas para calcular essas quantidades. Se a idade for maior ou igual a 70, uma das variáveis será incrementada de uma unidade. Caso contrário, a outra variável será incrementada de uma unidade.

Para a solução de tais problemas, utiliza-se a instrução ***if ... then ... else***, que possui as seguintes sintaxes dentre outras:

```
If <condição> then
    <instrução1>
else
    <instrução2>;

If <condição> then
begin
    <instruções1>;
```

```
se <condição> então
    <instrução1 ou instruções 1>
senão
    <instrução2 ou instruções 2>;
fim se;
```

```
end
else
begin
    <instruções2>;
end;
```

No uso da instrução ***if ... then ... else***, sendo a condição verdadeira, a instrução1 ou o bloco de instruções1 será executado. Em seguida, o fluxo de execução do programa será desviado para a primeira instrução após a estrutura *if ... then ... else*. Sendo a condição falsa, o fluxo de execução do programa será desviado para a execução da instrução2 ou do bloco de instruções2 e seguirá o fluxo normalmente com a execução da primeira instrução após a estrutura *if ... then ... else*. Observe, nas sintaxes da estrutura, de que a instrução única ou a palavra reservada *end,* que encerra o bloco de instruções correspondente ao *if,* não é seguida de ponto-e-vírgula ( ; ).

Vimos até aqui os conceitos elementares da linguagem Pascal capazes de auxiliar o aluno a elaborar seu programa de computador. A partir de agora, serão introduzidos novos conceitos e estruturas da linguagem que farão o aluno elaborar programas mais complexos e sofisticados de forma mais inteligente.

## 1.4. PROCEDIMENTOS E FUNÇÕES

De acordo com Castilho *et al.* (2020, p.145):

> (...) à medida em que os problemas exigem códigos com muitas linhas, os programas gerados vão se tornando cada vez mais complexos tanto para serem desenvolvidos quanto para serem mantidos em funcionamento.

Para a solução do problema, Manzano & Yamatumi (2002) propõem que o problema deva ser subdividido em problemas menores. Por consequência, cada parte menor terá um algoritmo mais simples chamado de sub-rotina. A própria sub-rotina pode ser dividida em outras sub-rotinas, quando necessário, visando sempre uma solução mais simples de um problema maior. Este processo é conhecido como Método de Refinamento Sucessivo.

As sub-rotinas são também conhecidas como subprogramas porque atuam dentro dos programas. São módulos utilizados para facilitar o trabalho dos programadores e analistas de sistemas e devem ser construídos com códigos bem elaborados coerentes com o problema que se deseja modelar. A motivação para o uso da estrutura, além da modularidade, são a facilidade da leitura do código e o aproveitamento deste código no programa principal (CASTILHO *et al.*, 2020).

Os subprogramas são divididos conforme a sua estrutura em procedimentos (***procedure***) e funções (***function***). A diferença básica entre eles é que a função retorna um valor e o procedimento não retorna valor, realizando apenas a execução de instruções que serão aproveitadas no programa principal ou em outro subprograma.

Castilho *et al.* (2009, p.16) mostra a similaridade entre a forma de estruturação dos programas e dos subprogramas. Assim como os programas, os procedimentos e funções são divididos em blocos com cabeçalho, uma área de declarações, nem sempre necessária, e uma área reservada ao bloco de comandos ou instruções. Assim se referem ao cabeçalho:

> A linha que contém o cabeçalho do procedimento ou função define a assinatura ou protótipo do subprograma, isto é, estabelece o seu nome, os parâmetros com respectivos tipos e, no caso das funções, o tipo de retorno.
> Os parâmetros podem ser passados de duas maneiras, tanto para procedimentos quanto para funções: por valor ou por referência. Na assinatura do subprograma, a diferença é que os parâmetros passados por referência levam a palavra **var** antes da lista de parâmetros.
> Os **parâmetros passados por valor** recebem uma **cópia** dos valores das variáveis usadas como argumento na chamada do subprograma. Os **parâmetros passados por referência** remetem, na verdade, à própria variável; que **sofrerá todas as alterações realizadas** (grifo nosso).

Normalmente nesses subprogramas são declaradas *variáveis locais* que têm escopo apenas dentro daquele subprograma, não interferindo no programa principal. Difere da variável global, que são as variáveis declaradas no cabeçalho do programa visíveis no programa principal, assim como nos subprogramas (CASTILHO *et al.*, 2020).

Para que os subprogramas possam ser executados, eles devem ser chamados pelo programa principal ou por outro subprograma. No caso dos procedimentos, a chamada é feita pelo seu nome e pela lista de argumentos

(também chamados de parâmetros reais), que deve ter o mesmo número de parâmetros (os chamados parâmetros formais) e tipos definidos no cabeçalho do subprograma. No caso das funções, a chamada é semelhante, todavia como há o retorno de um valor numérico ou booleano, a chamada poderá vir de qualquer lugar que se admita este valor.

a) *Function*:

O subprograma ilustrado na Figura 4 mostra a estrutura de uma função.

**Figura 4 –** Estrutura de uma Função

```
1  function eh_primo (n: integer): boolean;
2  var
3     i, j: integer;
4
5  begin
6     eh_primo := true;
7     j := trunc (sqrt (n));
8
9     if n = 1 then
10       eh_primo := false
11    else
12       for i:=2 to j do
13          if n mod i = 0 then
14             eh_primo := false;
15 end;
```

```
1  função eh_primo (n: inteiro): lógico;
2  variáveis
3     i, j: inteiro;
4
5  início
6     eh_primo = verdadeiro;
7     j := piso (raiz (n));
8
9     se n = 1 então
10       eh_primo := falso;
11    senão
12       para i = 2 a j faça
13          se resto (n, i) = 0 então
14             eh_primo := falso;
15       fim se;
16       fim para;
17    fim se;
18 fim;
```

Fonte: Autor

Em regra geral, o cabeçalho contém a palavra reservada ***function***, o nome da função, a lista de parâmetros e seu tipo (cada parâmetro separado por ponto-e-vírgula), nem sempre necessária, e o tipo de retorno da função (inteiro ou lógico). De acordo com o subprograma ilustrado na Figura 4, o cabeçalho da função está assim definido:

*function eh_primo (n: integer): boolean;*

A função tem um único parâmetro formal (*n*), do tipo inteiro, que corresponde a cópia do valor a ser passado como argumento pelo programa principal ou um outro subprograma. A função retornará um valor lógico *TRUE* ou *FALSE* que será utilizado pela estrutura de chamada da função.

Na área reservada a declarações, foram declaradas as variáveis locais ***i*** e ***j*** do tipo inteiro.

O corpo do subprograma contém as instruções que auxiliarão o programa principal a verificar se um número inteiro digitado pelo usuário é ou não primo. A primeira instrução parte do pressuposto que todo número recebido ***n*** é primo (***eh_primo := true***). A segunda instrução envolve as funções ***sqrt*** e ***trunc*** definidas pelo compilador. A função *sqrt (x)* retorna a raiz quadrada de um número racional *x* não negativo, e a função *trunc (x)* retorna o número inteiro menor ou igual ao número racional *x*. Assim, *j* receberá o número inteiro menor ou igual a raiz quadrada do número lido *n* (***j := trunc (srqrt (n))***. As demais instruções do subprograma dentro da estrutura ***if ... then ... else*** definirão se o número *n* de fato é ou não primo. Se *n = 1*, o número lido não é primo. Caso contrário, o subprograma testará se *n* é múltiplo de números que variam de *2 a j* (***for i := 2 to j do***). Caso o resto da divisão de *n* por *i* seja zero (***if n mod i = 0 then***), a função retornará falsa (***eh_primo := false***), indicando que *n* não é primo. Caso contrário, ***eh_primo*** continuará com o seu valor original ***TRUE***.

b) *Procedure*:

Os subprogramas ilustrados na Figura 5 mostram a estrutura de um procedimento.

**Figura 5 –** Estrutura de um Procedimento

```
1  procedure soma_aritmetica (num: real;
1                             var soma_arit: real);
2  begin
3    soma_arit := soma_arit + num;
4  end;
5
6  procedure prod_geometrica (num: real;
6                             var prod_geom: real);
7  begin
8    prod_geom := prod_geomt * num;
9  end;
10
11 procedure soma_harmonica (num: real;
11                            var soma_harm: real);
12 begin
13   soma_harm := soma_harm + (1/num);
14 end;
15
16 procedure soma_quadratica (num: real;
```

```
1  procedimento soma_aritmetica (num: real;
1                                variável soma_arit: real);
2  início
3    soma_arit:= soma_arit + num;
4  fim;
5
6  procedimento prod_geometrica (num: real;
6                                variável prod_geom: real);
7  inícion
8    prod_geom = prod_geomt * num;
9  fim;
10
11 procedimento soma_harmonica (num: real;
11                               variável soma_harm: real);
12 início
13   soma_harm = soma_harm + (1/num);
14 fim;
15
16 procedimento soma_quadratica (num: real;
```

```
16                                   var soma_quad: real);
17  begin
18    soma_quad := soma_quad + sqr (num);
19  end;
```

```
16                                   variável soma_quad: real);
17  início
18    soma_quad = soma_quad + sqr (num);
19  fim;
```

Fonte: Autor

Em regra geral, o cabeçalho contém a palavra reservada ***procedure***, o nome do procedimento e a lista de parâmetros e seu tipo (cada parâmetro separado por ponto-e-vírgula), nem sempre necessária. O cabeçalho do procedimento que auxilia o programa principal no cálculo da média aritmética está assim definido:

*procedure soma_aritmetica (num: real; var soma_arit: real);*

tendo dois parâmetros *num e soma_arit* do tipo real, separados por ponto-e-vírgula ( ; ). O parâmetro formal *num* é um parâmetro passado por valor, portanto uma cópia do argumento ou parâmetro real passado pelo programa principal. O parâmetro formal *soma_arit* é um parâmetro passado por referência (declarado com a instrução *var),* recebe o valor do argumento ou parâmetro real passado pelo programa principal. Assim, qualquer alteração do valor de *soma_arit* promoverá a alteração do valor da variável passada como argumento pelo programa principal e vice-versa. Os demais subprogramas possuem estruturas semelhantes.

Em nenhum dos procedimentos houve a necessidade de se reservar uma área para declarações.

Os quatro procedimentos ilustrados na Figura 5 auxiliam o programa principal a calcular, respectivamente, as médias aritmética, geométrica, harmônica e quadrática dos números reais não negativos digitados pelo usuário.

Podemos identificar, no corpo da ***Procedure soma_quadratica***, a instrução que envolve a função pré-definida pelo Pascal ***sqr (x),*** que retorna o quadrado de um número racional *x*.

## 1.5. VETORES

Vamos escrever um programa que leia 500 números inteiros digitados pelo usuário e os imprima em ordem inversa de sua entrada. Nos programas desenvolvidos até o momento, uma única variável foi criada para receber os

números digitados. Essa forma de escrever o programa torna inviável a solução do problema proposto. Castilho *et al.* (2020) explicam essa inviabilidade, pois ao se ler o segundo dado, o primeiro armazenado se perderia. O programa, então, precisaria utilizar tantas variáveis quantos fossem os dados de entrada, aqui a declaração de 500 variáveis. Todavia, o uso de grandes quantidades de variáveis reflete diretamente no tempo de compilação do programa, o que torna inexequível a implementação do programa.

Como solução, deve-se buscar uma nova forma do uso de variáveis. O Pascal criou estruturas conhecidas como vetores (*array*), que através de uma única variável (indexada) é possível alocar uma grande quantidade de dados de mesmo tipo. No momento da declaração da variável, o programador passa a ter acesso a uma quantidade definida de posições de memória. A declaração da variável é feita da seguinte forma:

*array [<início> .. <fim>] of <tipo>;*

onde <tipo> representa os tipos definidos na linguagem Pascal ou pelo próprio programador; <início> e <fim> são do tipo inteiro. Normalmente, para que sejam associadas 500 posições a uma variável, <início> será igual a 1 e <fim> será igual a 500, todavia nada impede que <início> seja igual a 0 ou a 5, e <fim> seja igual, respectivamente, a 499 ou a 504.

A variável do problema proposto seria declarada da seguinte forma:

*var v: array [1..500] of integer;*

onde v é o nome da variável com 500 posições de memória para serem guardados até 500 números inteiros.

Para armazenar um determinado valor, digamos 80, na posição 15 do vetor v, usaríamos umas das seguintes instruções no programa principal:

*v[15] := 80;*
*read (v[15]);  [e digitaríamos 80 seguido de ENTER].*

O vetor, quando usado em subprogramas sofre uma limitação na forma de ser declarado. Castilho *et al.* (2020, p.178-179) alertam que o compilador não

aceita que se passe o tipo array como parâmetro em funções e procedimentos, por isto normalmente o programador deve declarar um novo tipo para o array.

Isto se consegue com o uso da declaração *type*, conforme ilustrado a seguir:

*Type vetor = array [1..500] of integer;*

O programa ilustrado na Figura 6 utiliza a estrutura de um vetor, que recebe a entrada de 10 números inteiros, os processa e os imprime na tela do monitor na ordem inversa de suas entradas.

**Figura 6 –** Uso do Vetor

```
Program imprime_ordem_inversa;
type
   vetor = array [1..10] of integer;
var
   vet: vetor;

procedure leia_numero;
var
   I, num: integer;
begin
   for i := 1 to 10 do
   begin
      write ('Digite o ', i, '. numero: ');
      readln (num);
      vet[i] := num;
   end;
end;

procedure imprima_ordem_inversa (vet1:
                                        vetor);
var
   k: integer;
begin
   writeln ('Segue(m) o(s) numero(s) na
            ordem inversa de suas entradas: ');
   for k := 10 downto 1 do
      write (vet1[k]; ‘    ‘);
   end;

begin
   writeln ('Este programa recebe a entrada
         de 10 numeros inteiros e os imprime
         na ordem inversa de suas entradas');
   leia_numero;
   imprima_ordem_inversa (vet);
end.
```

```
Algoritmo imprime_ordem_inversa;
Tipo
   vetor = vetor [1..10] de inteiros;
variáveis
   vet: vetor;

procedimento leia_numero;
variáveis
   I, num: inteiro;
início
   para i := 1 a 10 faça
      escreva ('Digite o ', i, '. numero: ');
      laia (num);
      vet[i] := num;
   fim para;
fim;

procedimento imprima_ordem_inversa
                                    (vet1: vetor);
variáveis
   k: iinteiro;
início
   escreva ('Segue(m) o(s) numero(s) na
         ordem  inversa de suas entradas: ');
   para k := 1 a 10 faça
      escreva (vet1[11 - k]; ‘    ‘);
   fim para;
fim;

início
   escreva ('Este programa recebe a
            entrada de 10 numeros inteiros e
            os imprime na ordem inversa de
               suas entradas');
   leia_numero;
   imprima_ordem_inversa (vet);
fim.
```

Fonte: Autor

Inicialmente, o programa principal chamará a ***procedure leia_numero***, para a leitura e armazenamento dos 10 números inteiros digitados pelo usuário. O primeiro número lido será atribuído a ***vet[1]***. Em seguida, o segundo número lido será atribuído a ***vet[2]***, e assim por diante. Após a conclusão da leitura dos 10 números, o programa principal chama o procedimento ***ordem_inversa,*** passando para ele como argumento uma cópia dos valores de ***vet***. O procedimento permite que os valores armazenados no vetor sejam impressos em sua ordem inversa.

## 1.6. MATRIZES

As matrizes, assim como os vetores, são *arrays*. Enquanto os vetores têm uma estrutura unidimensional, onde o acesso às posições da memória é feito com base no nome da variável e em apenas um deslocamento, as matrizes têm estrutura bidimensional, onde o acesso às posições da memória é feito com base no nome da variável e em dois deslocamentos, o horizontal e o vertical (CASTILHO *et al.*, 2020).

A declaração da variável é feita da seguinte forma:

*m: array [<início1> .. <fim1>, <início2> .. <fim2>] of <tipo>;*

onde <tipo> representa os tipos definidos na linguagem Pascal ou pelo próprio programador; <início1>, <fim1>, <início2> e <fim2> são do tipo inteiro, sendo que o intervalo <início1> .. <fim1> representa o deslocamento horizontal (número de linhas da matriz); e o intervalo <início2> .. <fim2>, o deslocamento vertical (número de colunas da matriz).

Para declarar uma matriz de 800 posições reais, sendo 20 na horizontal e 40 na vertical, usa-se a seguinte sintaxe:

*var m: array [1..20, 1..40] of real;*

Para armazenar um determinado valor, digamos 175, na linha 16 e coluna 10 da matriz *m*, usaríamos umas das seguintes instruções no programa principal:

*m[16, 10] := 175;*
*read (m[16, 10]);  [e digitaríamos 175 seguido de ENTER].*

A matriz assim como o vetor também é utilizada nos subprogramas. Como o compilador não aceita que se passe o tipo array como parâmetro em funções e

procedimentos, o programador deve declarar um novo tipo para o array, conforme declaração a seguir:

*Type matriz = array [1..20, 1..40] of real;*

E usaríamos a seguinte construção:

*Procedure ler_numero (m: matriz);*

Para ilustrar o uso da matriz, vamos escrever um programa para construir uma matriz simétrica (Figura 7). Como a matriz simétrica é uma matriz quadrada A de ordem n, que satisfaz $A^t = A$, ou seja, os elementos $a_{ij}$ são iguais aos elementos $a_{ji}$, para a leitura de dados precisa-se apenas dos elementos $a_{ij}$, tais que $i \leq j$. Assim, o programa receberá os dados digitados pelo usuário dos valores dos elementos $a_{ij}$ da matriz $A_n$, quando $i \leq j$. Após o último dado digitado, será impressa a matriz simétrica.

**Figura 7** – Uso da Matriz

```
1  program matriz_simetrica;
2  type
3     matriz = array [1..50, 1.. 50] of real;
4  var
5     mat: matriz;
6     i, j, n: integer;
7
8  begin
9     write ('Ordem da matriz: ');
10    readln (n);
11    writeln ('Informe os valores dos
11                elementos da matriz');
12    for i := 1 to n do
13       for j:= i to n do
14       begin
15          write ('a', i, j, ' = ');
16          readln (mat[i,j]);
17          mat[j,i] := mat[i,j];
18       end;
19
20    writeln ('Matriz A:');
21    for i := 1 to n do
22    begin
23       for j:= 1 to n do
24          write (mat[i,j]:8:2);
25       writeln;
26    end;
27 end.
```

```
1  Algoritmo matriz_simetrica;
2  Tipo
3     matriz = matriz [1..50, 1.. 50] of real;
4  variáveis
5     mat: matriz;
6     i, j, n: integer;
7
8  início
9     escreva ('Ordem da matriz: ');
10    leia (n);
11    escreva ('Informe os valores dos
11                elementos da matriz');
12    para i := 1 a n faça
13       para j:= i a n faça
14          escreva ('a', i, j, ' = ');
15          leia (mat[i,j]);
16          mat[j,i] := mat[i,j];
17       fim para;
18    fim para;
19
20    escreva ('Matriz A:');
21    para i := 1 a n faça
22       para j:= 1 a n faça
23          escreva (mat[i,j]:8:2);
24       fim para;
25       escreva;
26    fim para;
27 fim..
```

Fonte: Autor

Até agora vimos estruturas ditas homogêneas, no sentido de que as diversas posições de memória alocadas são sempre do mesmo tipo. Passaremos a abordar uma estrutura heterogênea de dados, onde uma única variável é capaz de armazenar dados de diversos tipos.

## 1.7. REGISTROS

Usa-se o registro (*register*) para juntar em uma única variável dados de diferentes tipos. Segundo Castilho *et al.* (2009):

> *O* Tipo *record* (registro) *é* usado para aglomerar sob um mesmo nome de variável uma coleção de outras variáveis de tipos potencialmente diferentes. Por isto é uma estrutura heterogênea, contrariamente ao tipo *array* que é homogêneo.

Deve ser usado, por exemplo, para cadastrar o nome, CPF, data de nascimento e endereço dos pacientes de uma clínica médica, ou ainda, para cadastrar os dados de notas fiscais emitidas por uma empresa como emitente e destinatário, número da nota fiscal, descrição da mercadoria, quantidade e valor.

Vamos, então, declarar uma variável que possa receber as informações dos pacientes de uma clínica médica:

```
var paciente: record
        nome: string[60];
        cpf: string[11];
        idade: integer;
        sexo: char;
        endereço: string[100];
    end;
```

Para o registro do paciente, que pode ser feito através do comando de atribuição ou por meio de leitura dos dados digitados, usa-se o nome da variável seguido de um ponto ( . ) e do nome do campo. O paciente Marcos Pereira, CPF 111.111.111-11, com 56 anos, residente na rua tal, nº tal, seria assim registrado:

*paciente.nome := 'Marcos Pereira';* ou *read (paciente.nome);*

*paciente.cpf := '11111111111';* ou *read (paciente.cpf);*

*paciente.idade := 56;* ou *read (paciente.idade);*

*paciente.sexo := 'M';* ou *read (paciente.sexo);*

*paciente.endereço := 'rua tal, nº tal';* ou *read (paciente.endereço);*

Como ocorre ao se declarar vetores e matrizes, o compilador não aceita que se passe o tipo *record* como parâmetro em funções e procedimentos, necessitando que o programador declare um novo tipo para o *record*, conforme declaração a seguir:

```
Type dados_paciente = record;
(...)
end;
var pac: dados_paciente;
```

E usaríamos a seguinte construção:

```
Procedure ler_registro (var pac1: dados_paciente);
```

Castilho *et al.* (2020) ressaltam que os registros normalmente estão integrados a outras estruturas de dados, como vetores e matrizes, o que pode ser observado ao analisar o código do programa ilustrado na Figura 8.

Neste programa trabalhar-se-á com procedimentos e funções para a criação de um conjunto e manipulação de seus elementos (inserção, ordenação e remoção).

**Figura 8 –** Uso do Registro

```
Program Conjunto;
const max = 101;
type
  conjunto = record
    tam: integer ;
    v: array [ 0..MAX+1] of integer ;
  end;
var
  c: conjunto;
  j, n, elemento: integer;

procedure inicializar_conjunto (var c:
                                  conjunto);
begin
  c.tam:= 0;
end;

function conjunto_vazio (c: conjunto):
                                  boolean;
begin
procedure inserir_no_conjunto (x: integer;
                                  var c: conjunto);
var
  I, integer: integer ;
begin
  indice:= busca_binaria(x, c) ; //Localiza a
                        posicao do elemento
  if indice = 0 then //nao localizou o elemento
  begin
    c.v[0]:= x;
    i:= c . tam;

     while x < c .v[ i ] do //(* procura ponto
        para inserir o elemento e abe espaco
    begin
      c .v[ i+1]:= c .v[ i ] ;
      i := i - 1;
    end;

    c .v[ i+1]:= x;
```

```
19    if c. tam = 0 then
20       conjunto_vazio:= true
21    else
22       conjunto_vazio:= false;
23  end;
24
25  function busca_binaria (x: integer;
25                          c: conjunto): integer;
26  var
27     ini , fim: integer;
28  begin
29     ini:= 1;
30     fim:= c . tam;
31     busca_binaria:= 0;
32
33     while (ini <= fim) do
34     begin
35        if x < c .v[ini] then
36           ini:= fim + 1
37        else if x = c .v[ini] then
38        begin
39           busca_binaria:= ini;
40           Ini:= fim + 1;
41        end
42        else
43           ini:= ini + 1;
44     end;
45  end;
```

```
62       c . tam := c . tam + 1;
63    end;
64  end;
65
66  procedure remover_do_conjunto (x: integer;
66                          var c: conjunto);
67  var
68     i , indice : integer ;
69  begin
70     indice:= busca_binaria(x, c) ;     //localiza
70                          a posicao do elemento
71     if indice <> 0 then    //localizou  o elemento
72     begin       //empurra os elementos para tras
73        for i := indice to c.tam - 1 do
74           c .v[ i ]:= c .v[ i +1];
75        c.tam := c.tam - 1;
76     end;
77  end;
78
79  begin
80  // (*programa principal *)
```

Fonte: Castilho *et al.,* 2020

Na área de declarações do programa ***program conjunto***, declarou-se ***conjunto*** como um tipo de registro (***register***) formado pelos campos ***v***, um vetor que armazena os elementos do conjunto de números inteiros, e ***tam*** que guardará o número de elementos do conjunto ou a sua cardinalidade. Definiu-se a variável ***c*** do tipo conjunto, representando o conjunto a ser criado e manipulado.

Inicialmente o conjunto deve ser criado com o chamamento do procedimento ***inicializar_conjunto (c)***, onde a cardinalidade do conjunto será nula (*c.tam := 0*).

Para saber se o conjunto é vazio, chama-se a função ***conjunto_vazio (c)*** Caso a cardinalidade do conjunto seja nula, indicando que o conjunto é vazio, a função retorna *TRUE*; caso contrário, a função retorna *FALSE*.

A inserção de elementos no conjunto é feita através da chamada do procedimento ***inserir_no_conjunto (x, c)***. Caso o elemento ***x*** não pertença ao conjunto, há a sua inserção de tal forma que ele fique ordenado no conjunto. A

cardinalidade do conjunto será incrementada de uma unidade. A verificação de pertencimento ou não do elemento no conjunto ocorre através da chamada da função ***busca_binaria(x, c)***, para localizar a posição do elemento no conjunto. Se o valor retornado de ***busca_binaria(x, c)*** for igual a zero, o elemento não foi encontrado no conjunto, ou seja, ele não pertence ao conjunto.

A exclusão de um elemento do conjunto é feita através da chamada do procedimento ***remover_do_conjunto (x, c)***. Caso o elemento ***x*** pertença ao conjunto, há a sua remoção e a cardinalidade do conjunto será decrementada de uma unidade. A verificação de pertencimento ou não do elemento no conjunto ocorre através da chamada da função ***busca_binaria(x, c)***, para localizar a posição do elemento no conjunto. Se o valor retornado de ***busca_binaria(x, c)*** for diferente de zero, o elemento foi encontrado no conjunto, ou seja, ele pertence ao conjunto.

## 2. INTRODUÇÃO À TEORIA DOS GRAFOS

### 2.1. NOÇÕES PRELIMINARES

Neste capítulo introduziremos o tema Grafos.

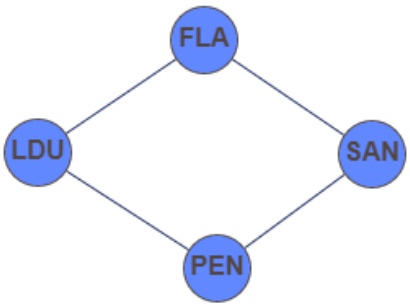
Como o Brasil é ou era o país do futebol, vamos começar a abordar a teoria trazendo como exemplo a maior competição da América do Sul, a Copa Libertadores das Américas.

Se pedirmos para que os alunos da educação básica relacionem os jogos do Grupo D da Copa Libertadores das Américas de 2019[4], formado por Flamengo (FLA), San José (SAN), LDU e Peñarol (PEN), que se enfrentaram até a 2ª rodada do campeonato, após uma rápida consulta na Internet, os alunos apresentariam a seguinte resposta:

- FLA jogou com SAN e LDU;
- SAN jogou com FLA e PEN;
- LDU jogou com PEN e FLA;
- PEN jogou com LDU e SAN.

Agora, se pedirmos para que os mesmos alunos elaborem um diagrama em que cada time é representado por um círculo e os jogos são representados por uma linha, os diagramas apresentados por eles seriam semelhantes ao da Figura 9.

**Figura 9 –** Diagrama




Fonte: Autor

Isso mostra que situações do nosso cotidiano podem ser representadas por meio de diagramas formados por um conjunto de círculos e linhas, as quais

[4] No ano 2019, o Flamengo sagrou-se bicampeão da Copa Libertadores das Américas ao derrotar na final a equipe argentina do River Plate por 2 a 1.

ligam pares desses círculos no caso de haver alguma relação predeterminada entre eles. No diagrama da Figura 9, a relação seria os jogos efetivamente disputados.

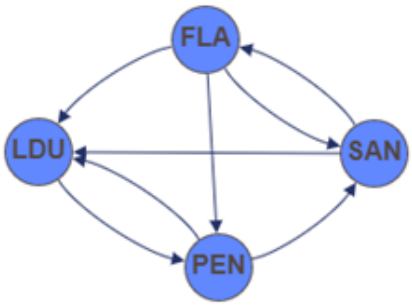
O que acabamos de representar no diagrama é conhecido como grafo. Moreno & Ramírez (2011) define grafo como um par ordenado G = (V, E) composto por um conjunto finito V = V(G) de vértices ou nós, representado por V = {v1, v2, ..., vn}, tipicamente desenhados por círculos, e um conjunto de arestas ou elos E = E(G) ⊆ V2, representado por E = {{v1, v2}, ..., {vi, vj}, ....}, donde definimos V2 = (V × V).

O grafo da Figura 9 é representado pelo conjunto de vértices V = {FLA, SAN, LDU, PEN} e pelo conjunto de arestas E = {{FLA, SAN}, {FLA, LDU}, {SAN, PEN}, {LDU, PEN}}.

Observe que não precisamos colocar o elemento {SAN, FLA} no conjunto de arestas, pois o conjunto já é formado pelo elemento {FLA, SAN}. Contrariamente, caso a relação predeterminada fosse, além do confronto entre os times, o mando de campo, a ordem importaria. Para a representação dessa ordem, não utilizaríamos as chaves e sim parêntesis. Por exemplo, (FLA, SAN) poderia representar o confronto entre Flamengo e San José, em que o mandante da partida fosse o Flamengo, enquanto (SAN, FLA) representaria o confronto entre os mesmos times tendo como mandante a equipe do San José.

O grafo da Figura 10 representa os jogos realizados pelas equipes do Grupo D da Copa Libertadores das Américas de 2019 até a 4ª rodada do campeonato, com a inclusão do critério do mando de campo.

**Figura 10 –** Grafo




Fonte: Autor

Observe que o grafo está representado por círculos e arcos. Este tipo de grafo é classificado como grafo dirigido ou orientado, diferentemente do grafo da Figura 9, classificado como grafo não dirigido ou não orientado. A diferença básica entre eles diz respeito à importância da ordem dos elementos que formam as arestas, no caso de grafos não dirigidos, ou os arcos, no caso de grafos dirigidos. No caso de grafos não dirigidos, não existe essa importância. Uma das características para este tipo de grafo é a relação de simetria entre seus elementos, onde {a, b} e {b, a} representam a mesma aresta. Devido à relação de simetria, foi desnecessário inserir no conjunto das arestas do grafo da Figura 9 o elemento {SAN, FLA}, uma vez que o elemento {FLA, SAN} já pertencia ao conjunto.

Um grafo dirigido ou orientado $D = (V, A)$ é um grafo cujos arcos $a = (v_1, v_2) \in A$ representam uma conexão ou ligação entre dois dos vértices na direção de $v_1$ até $v_2$, usualmente denotada por um arco que vai desde $v_1$ até $v_2$ (MORENO & RAMÍREZ, 2011).

O conjunto dos arcos do grafo da Figura 10 é formado pelos elementos a seguir:

A = {(SAN, FLA), (LDU, PEN), (FLA, LDU), (PEN, SAN), (SAN, LDU), (FLA, PEN), (PEN, LDU), (FLA, SAN)}

Feita essa breve introdução, passa-se a conhecer a história do surgimento da Teoria dos Grafos e a sua importância na modelagem e solução dos problemas envolvendo o nosso cotidiano.

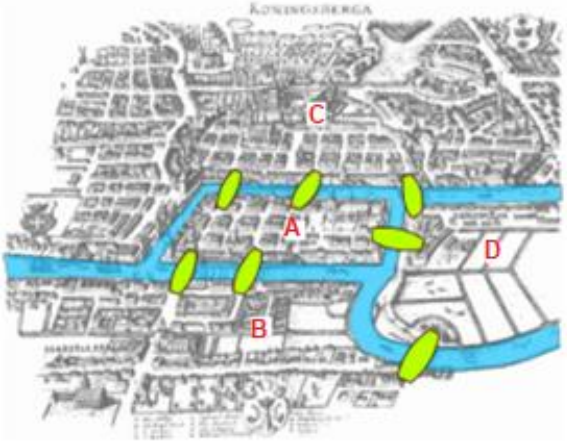
## 2.2. ORIGEM DA TEORIA DOS GRAFOS

Segundo Boaventura Netto e Jurkiewicz (2009), o primeiro registro de um problema envolvendo a Teoria dos Grafos remonta o ano de 1736. Conhecido como o Problema das sete Pontes de Konigsberg, a solução envolveu a formulação de alguns dos conceitos básicos utilizados atualmente pela teoria.

A antiga cidade de Konigsberg, na então Prússia Oriental, atual Caliningrado (território russo localizado entre Polônia e Lituânia) é banhada pelo rio

Pregel, que em seu curso, forma a ilha de Keniphof, a qual se liga às demais partes da cidade por sete pontes. A Figura 11 ilustra o mapa da região à época.

**Figura 11 –** As sete pontes de Konigsberg



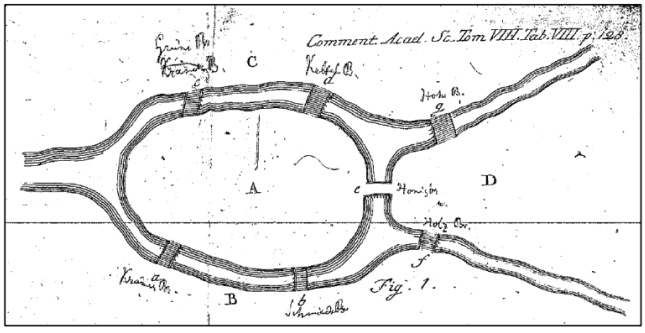

Fonte: Souza, 2013

Relata-se que os habitantes da cidade almejavam encontrar um percurso que passasse pelas sete pontes uma única vez retornando ao local de partida.

Coube ao notável matemático Leonardo Euler (1707-1783) encontrar a solução do problema. Há três cartas, que integram a coleção do Arquivo da Academia de Ciências de São Petersburgo que relatam a troca de mensagens sobre o assunto (SANTOS & MOTA, 2010).

Para resolver o problema das Sete Pontes de Konigsberg, Euler elaborou o esquema apresentado na Figura 12 e representou a ilha pela letra A e as porções de terra pelas letras B, C e D. Representou as sete pontes pelas letras minúsculas a, b, c, d, e, f e g.

**Figura 12 –** Esquema do Problema das Sete Pontes apresentado por Euler




Fonte: Santos & Mota, 2010

Em seguida, apresentou o seu entendimento para a solução do problema, não se limitando a encontrar uma solução para o caso em particular. Ao final, apresentou as condições para que houvesse um percurso passando uma única vez por cada uma das pontes que ligasse as porções de terra:

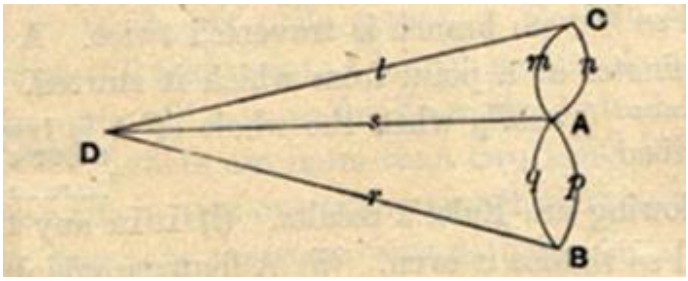
a) Só se pode realizar um caminho passando em todas as pontes uma única vez se, e somente se, cada porção de terra possuir uma quantidade par de pontes. Neste caso, pode-se retornar à porção de terra de origem. Em homenagem a Euler, o caminho passou-se a chamar de Euleriano;

b) Caso apenas duas porções de terra possuam uma quantidade ímpar de pontes, é possível realizar um caminho passando em todas as pontes uma única vez, desde que a origem e o fim do caminho sejam essas porções de terra. Aqui não existe a possibilidade de se retornar à origem. Também em homenagem a Euler, esse caminho passou-se a chamar de Semi-Euleriano.

Mesmo Euler não considerando em sua carta o problema das Pontes de Konigsberg relacionado à matemática, fez questão de publicar a sua solução no artigo "*Solutio problematis ad geometriam situs pertinentis*", enviado ao *Commentarii Academiae Scientiarum Imperialis Petropolitana,* marcando o início da Teoria dos Grafos (SANTOS & MOTA, 2010).

Todavia, como bem ressaltou Santos & Mota (2010), o modelo de grafo com seus vértices e arestas, correspondendo ao problema das Pontes de Konigsberg, representado pelo grafo da Figura 13, surgiu cerca de 150 anos depois, em 1892, quando W. W. Rouse Ball apresentou o problema num livro de recreações matemáticas.

**Figura 13 –** Representação do grafo do problema das pontes de Konigsberg




Fonte: Santos & Mota, 2010

Fazendo analogia entre a atual linguagem aplicada à Teoria dos Grafos e a solução proposta por Euler, as porções de terra representariam os vértices do grafo; e as pontes, as suas arestas. O número de pontes ligadas a cada porção de terra representaria o grau do vértice, que, como veremos adiante, é o número de arestas a ele ligadas.

A contribuição de Euler com a matemática vai além de resolver o problema das sete pontes. Euler foi o autor de uma quantidade imensa de artigos, que serviram de inspiração para a publicação de mais de 500 livros e artigos durante o período de sua vida. Graças aos trabalhos desenvolvidos pelo matemático, os conhecimentos disponíveis em quase todos os ramos da matemática pura e aplicada expandiram-se. A grandeza de Euler se mostra inclusive na linguagem e notação matemáticas usadas por ele em seus trabalhos, as mesmas usadas hoje, cerca de três séculos depois de sua brilhante passagem pelo mundo da matemática e da filosofia (BOYER, 1974).

Talvez pela pouca importância dada ao tema por Euler, a Teoria dos Grafos só se tornou relevante ao longo do século XX, dando um salto gigantesco a partir da década de 1950, com as aplicações no campo da pesquisa operacional, viabilizadas pelo uso intenso do computador (BOAVENTURA NETTO, 2011).

Antes disso, há alguns trabalhos que merecem destaque pela contribuição à teoria. Boaventura Netto (2011) cita o trabalho desenvolvido pelo físico alemão Gustav Robert Kirchhoff, que ao utilizar grafos em seus estudos sobre circuitos elétricos em 1847, criou a teoria das árvores, baseada em grafos conexos e sem ciclos.

Dez anos depois, em 1857, surgiu um outro personagem que fez importantes contribuições à Teoria dos Grafos, o matemático britânico Arthur Cayley (1821-1895). Ele introduziu a palavra “árvore” na teoria. Estudou árvores de modo sistemático e, através desses estudos, deixou como contribuição importante trabalho na área da química orgânica, mostrando a sua utilidade na enumeração de compostos químicos, conduzindo a descoberta de compostos desconhecidos e

desenvolvendo uma técnica para determinar o número de diferentes isômeros dos hidrocarbonetos.[5]

Atualmente, as aplicações da Teoria dos Grafos ocorrem em diversas áreas do conhecimento como na engenharia no estudo de traçado de rotas (SILVA, 2020), do controle de tráfego viário (HERNANDES, 2007), na interligação de rede elétrica (RESE *et al.*, 2017), ou no desenvolvimento de rede de computadores (MOTTA & BRITO, 2017); na química com os projetos de novos compostos químicos (BOAVENTURA NETTO & JURKIEWICZ, 2009); na biologia através dos estudos de sequenciamento de DNA (BOAVENTURA NETTO & JURKIEWICZ, 2009), e mesmo na psicologia, na análise do desenvolvimento humano (ALMEIDA & CUNHA, 2003).

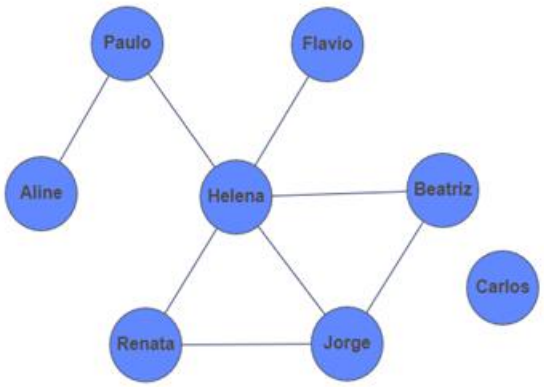
## 2.3. CONCEITOS BÁSICOS

### 2.3.1. ROTULAÇÃO

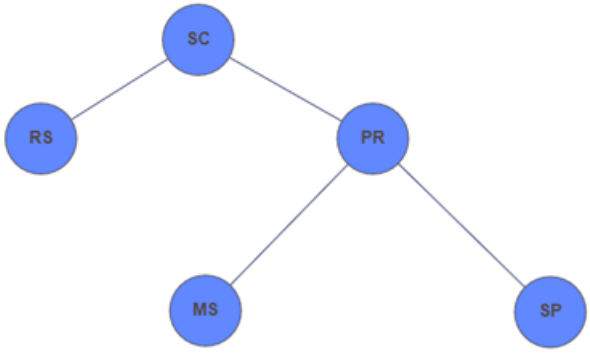
Segundo Boaventura Netto e Jurkiewicz (2009), para que o conjunto dos vértices de um grafo G = (V, E) fique bem definido, os seus elementos devem ser rotulados apropriadamente, permitindo a sua correta identificação. Os rótulos podem ser nome de times de futebol, nomes de pessoas, nomes de estados ou cadeias numéricas. Esta última é importante quando se manipula dados através de um programa de computador.

**Figura 14 –** Rotulação

a)

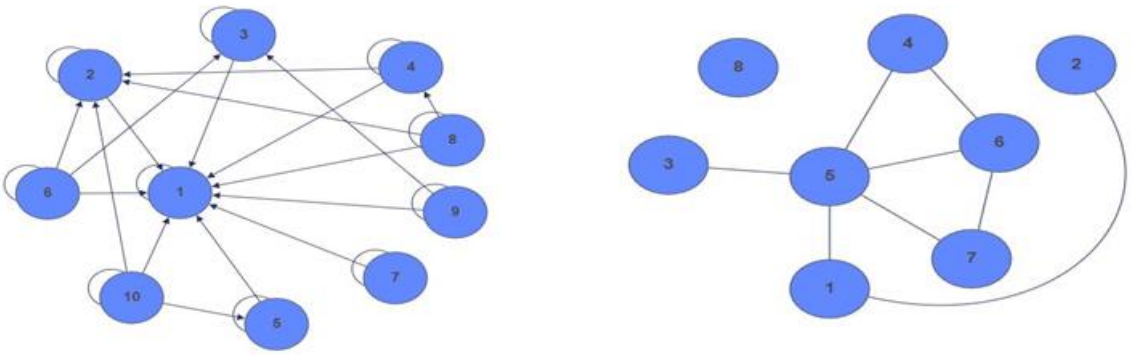



b)




[5] Disponível em: http://www.mat.uc.pt/~picado/ediscretas/2012/apontamentos/cap2.pdf. Acesso em 01.set. 2021.

c) d)




Fonte: Autor

No tópico 2.1 deste capítulo, definimos aresta e arco como uma relação entre pares de vértices. No caso dos grafos ilustrados nas Figuras 9 e 10, a relação era, respectivamente, os jogos do grupo D já realizados até a 4ª rodada da Copa Libertadores das Américas de 2019 e o mando de campo relacionados a estes jogos. No caso dos grafos da Figura 14, a relação entre os vértices se refere à relação de amizade entre os alunos de um curso de línguas (Figura 14a), às fronteiras entre os estados da região Sul do Brasil com outros estados da federação (Figura 14b) e aos divisores dos números naturais menores ou iguais a 10 (Figura 14c). Observe que neste grafo cada vértice se conecta a ele mesmo, uma vez que todo número inteiro diferente de zero é divisor dele próprio. Estes arcos são chamados de laços. Quanto à relação entre os vértices do grafo da Figura 14d, abordar-se-á mais adiante no tópico sobre isomorfismo entre grafos.

### 2.3.2. ORDEM E TAMANHO DE UM GRAFO

Ordem de um grafo é o número de vértices que ele possui, enquanto o tamanho é o seu número de arestas ou arcos.

O grafo da Figura 14b tem ordem 5 e tamanho 4.

### 2.3.3. GRAFO COMPLEMENTAR ($\overline{G}$)

Dado um grafo G, o grafo complementar $\overline{G}$ é um grafo que contém as ligações que não estão no grafo G em relação a um universo dado.

Esse universo, em grafos não orientados, é o conjunto de todas as

arestas possíveis em um grafo $G_1$ de mesma ordem que G, chamado de grafo completo. Sendo *n* a ordem de G, então o conjunto de arestas do grafo completo $G_1$ será:

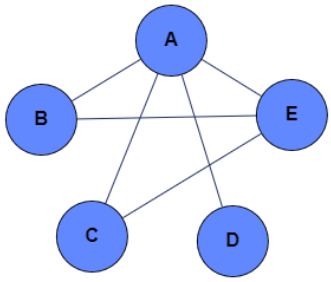
$$C_2^n = \frac{n \,.\, (n-1)}{2}$$

Se o tamanho de G é *m*, então $\overline{G}$ terá o tamanho $m_1 = C_2^n - m$. Logo:

$$m_1 = \frac{n \,.\, (n-1)}{2} - m$$

A Figura 15 ilustra um grafo G não orientado de ordem n = 5 e tamanho m = 6. Os grafos das Figuras 16a e 16b representam, respectivamente, o grafo completo de mesma ordem que G e tamanho m = 10 e o grafo complementar $\overline{G}$ de tamanho m = 4.

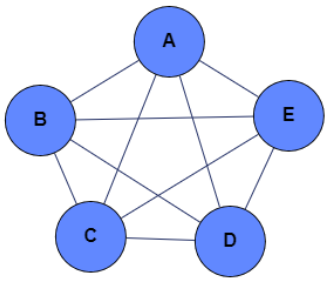
**Figura 15 –** Grafo não Orientado



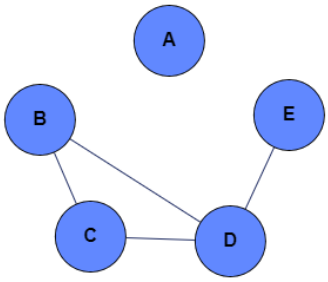

Fonte: Autor

**Figura 16 –** Grafo Completo e Grafo Complementar

a)




b)




Fonte: Autor

Em grafos orientados, o conjunto universo conterá todos os arcos, em ambos os sentidos, tratando-se de um grafo completo simétrico. Assim, dado um

grafo orientado D, de ordem *n*, o conjunto de arcos do grafo completo simétrico de mesma ordem de G será:

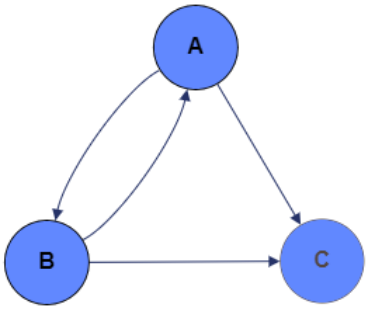
$$2 \, . C_2^n = n \, . (n-1)$$

Se o tamanho de D é *m*, então $\bar{D}$ terá o tamanho $m_1 = 2 \, . C_2^n - m$. Logo:

$$m_1 = \; n \, . (n-1) - \; m$$

A Figura 17 ilustra um grafo D orientado de ordem n = 3 e tamanho m = 4. Os grafos das Figuras 18a e 18b representam, respectivamente, o grafo completo de mesma ordem que D e tamanho m = 6 e o grafo complementar $\bar{D}$ de tamanho m = 2.

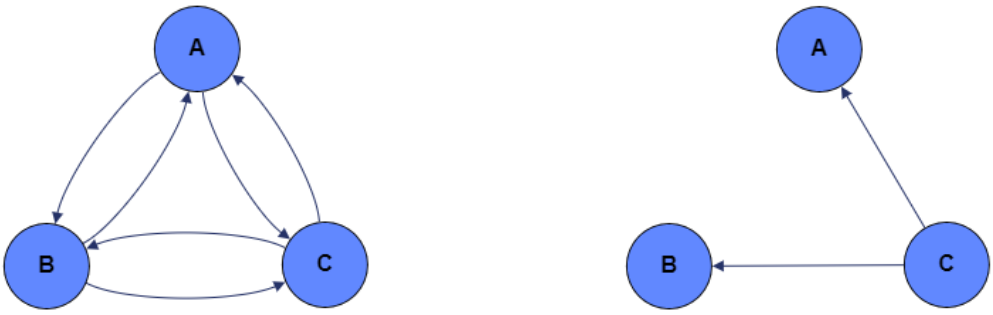
**Figura 17 –** Grafo Orientado




Fonte: Autor

**Figura 18 –** Grafo Completo e Grafo Complementar

a) b)




Fonte: Autor

### 2.3.4. SUBGRAFO

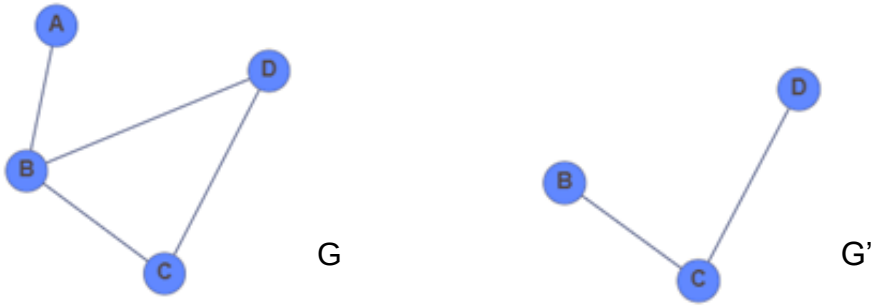
Em uma linguagem informal, podemos dizer que um subgrafo está contido em um grafo. Na verdade, são os conjuntos de vértices e arestas (arcos) de um

subgrafo que estão contidos, respectivamente, nos conjuntos de vértices e de arestas (arcos) de um grafo.

Assim, se $G' = (V', E')$ é subgrafo do grafo $G = (V, E)$ então $V'(G') \subseteq V(G)$ e $E'(G') \subseteq E(G)$.

De acordo com a Figura 19, o grafo G' é um subgrafo do grafo G.

**Figura 19 –** Subgrafo




Fonte: Autor

### 2.3.5. VIZINHANÇA OU ADJACÊNCIA

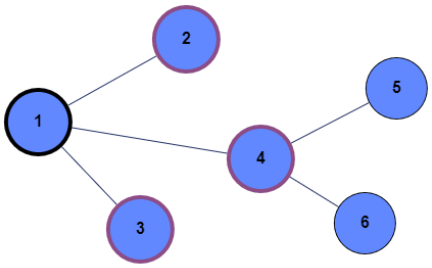
Vértices vizinhos ou adjacentes são aqueles ligados por uma aresta ou arco.

Seja $v_i$ um vértice do grafo não orientado $G = (V, E)$. A vizinhança de $v_i$, denotada por $N(v_i)$, é o conjunto de vértices de G a ele ligados, ou seja, o conjunto dos vizinhos de $v_i$. Assim:

$$N(v_i) = \{v_j \in V \mid \exists \{v_i, v_j\} \in E\}$$

De acordo com o grafo $G = (V, E)$ ilustrado na Figura 20, a vizinhança do vértice 1 é $N(1) = \{2, 3, 4\}$.

**Figura 20 –** Vizinhança do Vértice 1




Fonte: Autor

No caso de grafos orientados, os vizinhos de um vértice são chamados de sucessores (vértices de entrada dos arcos) ou antecessores (vértices de saída dos arcos).

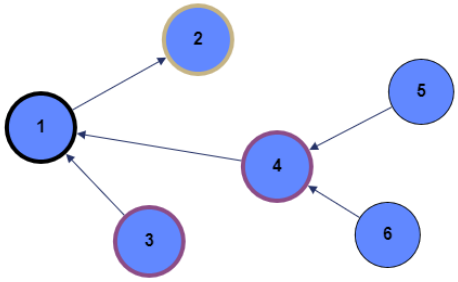
Seja $v_i$ um vértice do grafo orientado D = (V, A). Os conjuntos de sucessores e antecessores de $v_i$, denotados respectivamente por $N^+(v_i)$ e $N^-(v_i)$, são assim definidos:

$$N^+(v_i) = \{v_j \in V \mid \exists\ (v_i,v_j) \in A\}$$

$$N^-(v_i) = \{v_j \in V \mid \exists\ (v_j, v_i) \in A\}$$

De acordo com o grafo D = (V, A) ilustrado na Figura 21, os conjuntos dos sucessores e dos antecessores do vértice 1 são, respectivamente, $N^+(1) = \{2\}$ e $N^-(1) = \{3, 4\}$.

**Figura 21 –** Sucessores e Antecessores do Vértice 1




Fonte: Autor

### 2.3.6. GRAU E SEMIGRAUS

Em um grafo não orientado G = (V, E), o grau de um vértice $v_i \in V$, denotado por $d(v_i)$, é o número de vizinhos que ele possui. No caso de vértices isolados, ou seja, vértices que não possuem vizinhos, o seu grau é igual a zero.

No grafo da ilustrado na Figura 20, tem-se:

- d(1) = d(4) = 3;
- d(2) = d(3) = d(5) = d(6) = 1.

Para um grafo não orientado G, o grau máximo é denotado por Δ(G) e o grau mínimo por δ(G), correspondendo, respectivamente, ao maior e ao menor grau

entre os vértices do grafo. No grafo ilustrado na Figura 20, tem-se: Δ(G) = 3 e δ(G) = 1.

Caso todos os vértices de um grafo não orientado G, de ordem n, tenham o mesmo grau k, diz-se que G é um grafo (K)-regular ou simplesmente regular. Sendo G um grafo completo de ordem n, tem-se:

$$\Delta(G) = \delta(G) = d(v_i) = k = n - 1$$

É o caso do grafo de ordem 5 ilustrado na Figura 16a, em que todos os vértices têm grau igual a 4.

Em um grafo orientado D = (V, A), o grau de um vértice $v_i \in V$ é formado por dois semigraus: o semigrau exterior $d^+(v_i)$, que é o número de sucessores de $v_i$, e o semigrau interior $d^-(v_i)$, que é o número de antecessores de $v_i$. A soma dos semigraus de $v_i$ representa o seu grau.

$$d(v_i) = d^+(v_i) + d^-(v_i)$$

No grafo ilustrado na Figura 21, tem-se:

- $d^+(2) = d^-(3) = d^-(5) = d^-(6) = 0$;
- $d^+(1) = d^+(3) = d^+(4) = d^+(5) = d^+(6) = d^-(2) = 1$;
- $d^-(1) = d^-(4) = 2$.

Logo:

- $d(1) = d(4) = 3$
- $d(2) = d(3) = d(5) = d(6) = 1$

**Teorema 1:** A soma dos graus de todos os vértices de um grafo (orientado ou não orientado) é igual a duas vezes o número de arestas.

$$\sum d(Vi) = 2\ m$$

**Prova:** Cada aresta (arco) contribui com duas unidades para a soma dos graus.

### 2.3.7. PERCURSO OU CADEIA E CICLO

O percurso ou cadeia em um grafo é um conjunto de vértices e arestas (arcos) sequencialmente conectados ($v_0A_1v_1A_2v_2A_{3...}v_n$ ou, simplesmente, $v_0v_1v_{2...}v_n$).

Diz-se que o percurso é fechado se o primeiro elemento foi igual ao último ($v_0 = v_n$). Caso contrário, o percurso será aberto. O percurso que não repete ligações é dito simples, e o que não repete vértices é dito elementar (BOAVENTURA NETTO & JURKIEWICZ, 2009).

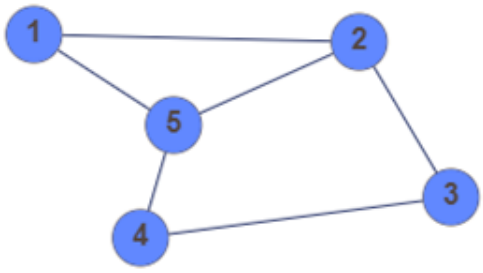
O comprimento de um percurso é o número de arestas (arcos) da sequência. Se um percurso tem n vértices, então seu comprimento é pelo menos n – 1, em face da possibilidade de na sequência haver vértices repetidos, ou igual a n – 1 quando a cadeia é dita elementar.

Um ciclo em um grafo é um percurso elementar fechado de comprimento maior que 1.

Essas definições servem tanto para grafos orientados (bastando que se ignore a orientação dos arcos) como para grafos não orientados, todavia, em relação ao primeiro, há definições específicas como passeio, caminho e circuito.

Considere o grafo da Figura 22:

**Figura 22 –** Percurso ou Cadeia e Ciclo




Fonte: Autor

Pode-se afirmar que:

- As sequências 1, 1-5-4 e 1-2-3-4-5 representam percursos simples, elementares e abertos com comprimentos, respectivamente, iguais a 0, 2 e 4;
- A sequência 5-2-5-4 representa um percurso, não se caracterizando como percurso simples ou elementar (repetição do vértice 5 e da aresta {2,5});
- A sequência 2-3-4-5-2 representa um ciclo e, a sequência 2-3-4-5-2-1-5-2 (vértice 5 se repete) não representam um ciclo.

### 2.3.8. PASSEIO, CAMINHO E CIRCUITO

Em se tratando de grafos orientados, Feofiloff (2020) atribui nomes específicos para percursos e ciclos como passeio, caminho e circuito.

Um passeio em um grafo é um conjunto de vértices e arcos sequencialmente conectados ($v_0A_1v_1A_2v_2A_3.....v_n$ ou, simplesmente, $v_0v_1v_2.....v_n$), em que tais arcos estão sob a mesma orientação (sentido do vértice para o seu sucessor). Um passeio é dito fechado quando seu primeiro vértice coincide com o último ($v_0 = v_n$).

Um caminho em um grafo é um passeio sem arcos repetidos. Um caminho é dito simples se não possui vértices repetidos.

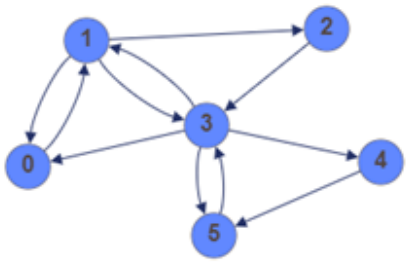
Se um caminho começa em $v_0$, passa ou não pelos vértices $v_i \in V \setminus \{v_0, v_n\}$ e termina em $v_n$, dizemos que ele vai de $v_0$ a $v_n$.

O comprimento de um caminho é o número de arcos da sequência. Se um caminho tem n vértices, então seu comprimento é pelo menos n – 1, em face da possibilidade de na sequência haver vértices repetidos, ou igual a n – 1 quando o caminho é dito simples.

Um circuito em um grafo é um caminho fechado, portanto um passeio sem arcos repetidos, possuindo comprimento maior que 1, em que o primeiro vértice da sequência coincide com o último ($v_0 = v_n$). Um circuito é dito simples se não possui vértices em comum, exceto o último que coincide com o primeiro.

Considere o grafo da Figura 23:

**Figura 23 –** Passeio, Caminho e Circuito




Fonte: Autor

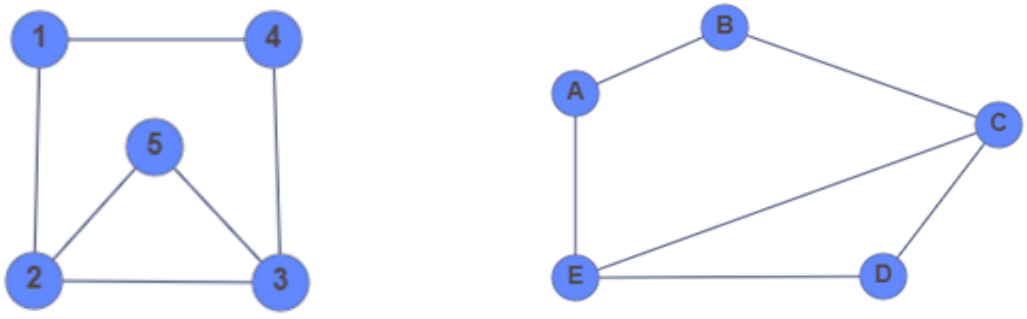
Pode-se afirmar que:

- As sequências 1, 1-2-3, 1-2-3-4-5, 0-1-3-5 representam caminhos simples, com comprimentos, respectivamente, iguais a 0, 2, 4 e 3;
- A sequência 2-3-4-5-3-0 representa um caminho, que não se caracteriza como simples (repetição do vértice 3);
- A sequência 1-3-0-1-3-4-5 representa um passeio, mas não um caminho (repetição do arco (1, 3));
- A sequência 1-0-3-4-5 não representa um passeio (inexistência do arco (0-3));
- Alguns dos caminhos que vão do vértice 1 ao vértice 0 são 1-0, 1-3-0, 1-2-3-0 e 1-2-3-4-5-3-0;
- As sequências 1-2-3-1, 1-2-3-0-1, 2-3-1-2, 3-4-5-3 e 1-2-3-4-5-3-0-1 representam circuitos, sendo as três primeiras caracterizadas como simples;
- As sequências 1 (comprimento igual a zero) e 0-1-2-3-1-3-4-5-3-1-0 (repetição do arco (3, 1)) representam passeios, mas não circuitos.

### 2.3.9. ISOMORFISMO

Observe os grafos $G_1$ e $G_2$ da Figura 24:

**Figura 24 –** Grafos Isomorfos




$G_1$ $G_2$

Fonte: Autor

Grafos como estes são ditos isomorfos por preservarem a mesma estrutura (grau de seus vértices, número de arestas ou arcos, direção dos arcos – no

caso de grafos direcionados –, etc). Dito numa linguagem informal é como se os dois grafos fossem um só, usando uma roupagem diferente (forma e/ou rotulação).

Por definição, dois grafos são isomorfos se, e somente se, existir uma função bijetiva entre seus vértices que preserve suas relações de adjacência. A função garante que os mesmos pares de vértices estejam envolvidos na definição das mesmas arestas, todavia podendo haver rótulos novos e/ou uma nova disposição espacial dos vértices no grafo.

As características de cada grafo constam nos Quadros 1a e 1b:

**Quadro 1 –** Características dos Grafos

a)

| Grafo $G_1$ | | |
|---|---|---|
| Vértices | Grau dos vértices | Vizinhos |
| 1 | 2 | 2, 4 |
| 2 | 3 | 1, 3, 5 |
| 3 | 3 | 2, 4, 5 |
| 4 | 2 | 1, 3 |
| 5 | 2 | 2, 3 |

b)

| Grafo $G_2$ | | |
|---|---|---|
| Vértices | Grau dos vértices | Vizinhos |
| A | 2 | B, E |
| B | 2 | A, C |
| C | 3 | B, D, E |
| D | 2 | C, E |
| E | 3 | A, C, D |

Fonte: Autor

Note que:

- Ambos os grafos têm 5 vértices;
- Em cada grafo, dois dos vértices têm grau 3; e os demais, grau 2.

Sejam f e $f^{-1}$ funções que associam, respectivamente, um vértice de $G_1$ a um vértice de $G_2$ e vice-versa. Podemos estabelecer a seguinte correspondência entre os vértices dos grafos $G_1$ e $G_2$.

Tomemos:

f(1) = A, f(2) = E, f(3) = C, f(4) = B e f(5) = D e

$f^{-1}$(A) = 1, $f^{-1}$(E) = 2, $f^{-1}$(C) = 3, $f^{-1}$(B) = 4 e $f^{-1}$ (D) = 5.

Ao se estabelecer uma correspondência entre os vértices de um grafo $G_1$ com os vértices de um grafo $G_2$, conclui-se que $G_1$ e $G_2$ são isomorfos.

No início deste capítulo, no tópico sobre rotulação, abordou-se o que representava cada aresta ou arco nos grafos ilustrados pelas Figuras 14a a 14d. Em relação ao grafo da Figura 14d, ressaltou-se que a relação entre os vértices do grafo seria abordada neste tópico. Vê-se que aquele grafo é isomorfo ao grafo da Figura 14a. Portanto, as arestas representam a relação de amizade entre as pessoas de um curso de línguas. A relação de correspondência entre os vértices dos grafos é mostrada no Quadro 2:

**Quadro 2 –** Relação entre os Vértices dos Grafos

| **Grafos** | **Vértices** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Fig. 14a** | Aline | Paulo | Flavio | Helena | Beatriz | Renata | Jorge | Carlos |
| **Fig. 14d** | 2 | 1 | 3 | 5 | 4 | 7 | 6 | 8 |

Fonte: Autor

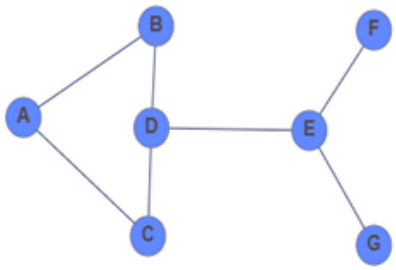
### 2.3.10. CONEXIDADE

a) Grafos não orientados:

Diz-se que um grafo não orientado G = (V, E) é conexo se existe pelo menos um caminho entre cada par de vértices do grafo. No caso de não existir um caminho que ligue pelo menos dois vértices do grafo, diz-se que o grafo é não conexo (BOAVENTURA NETTO & JURKIEWICZ, 2009).

A Figura 25 ilustra o caso de um grafo conexo.

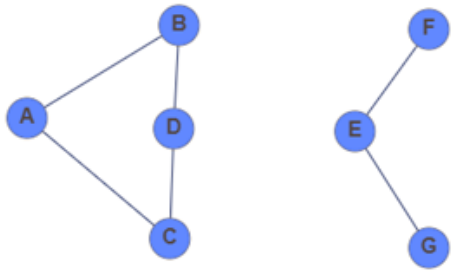
**Figura 25 –** Grafo não Orientado Conexo




Fonte: Autor

O grafo de ordem 7 da Figura 26 ilustra o caso de um grafo não conexo. Vê-se, por exemplo, que não existe um caminho que ligue o vértice A ao vértice F.

**Figura 26 –** Grafo não Orientado não Conexo




Fonte: Autor

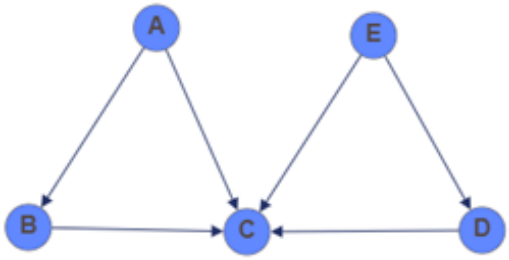
Observe que, ao se retirar a aresta {D, E} do grafo da Figura 25, ele se torna não conexo, ou ainda, ao introduzir a aresta {D, E} no grafo da Figura 26, ele se torna conexo. Assim, a aresta {D, E} é essencial para conectar ambos os grafos. Arestas como esta, indispensáveis para a conexão de um grafo, são chamadas de ponte. Para ilustrar, o vértice D poderia representar a cidade do Rio de Janeiro; e o vértice E, a cidade de Niterói. Sabemos que a Ponte Rio-Niterói é fundamental para ligar as duas cidades. Aqui ela estaria representada pela ponte {D, E}.

b) Grafos orientados

De acordo com Boaventura Netto e Jurkiewicz (2009), os grafos orientados D = (V, A) podem ser simplesmente conexo (s-conexo), semi-fortemente conexo (sf-conexo), fortemente conexo (f-conexo) ou não conexo. Vamos definir cada um deles, ressaltando apenas que nos três primeiros tipos de grafos, ao se deixar de lado a questão da orientação, eles estariam definidos como grafos conexos.

- **Grafo simplesmente conexo (s-conexo):** Há pelo menos um par de vértices u, v sem caminho que os ligue. No grafo da Figura 27, não há caminho que liga o vértice A ao vértice E em ambos os sentidos.

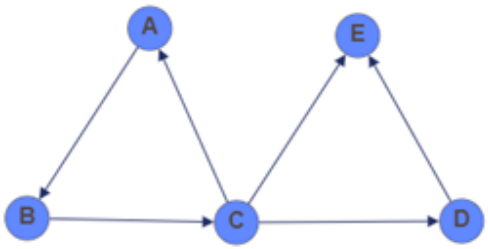
**Figura 27 –** Grafo Orientado s-Conexo




Fonte: Autor

- **Grafo semi-fortemente conexo (sf-conexo):** Para todo par de vértices u, v existirá um caminho de u até v e/ou de v até u, todavia, em pelo menos um deles, será em apenas um sentido. No grafo da Figura 28, há caminhos que ligam cada par de vértices, todavia não existe caminho que saia de E e chegue em A.

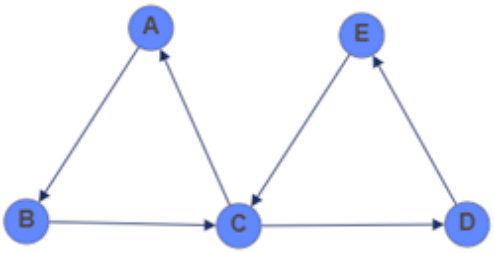
**Figura 28 –** Grafo Orientado sf-Conexo




Fonte: Autor

- **Grafo fortemente conexo (f-conexo):** Para todo par de vértices u, v existe um caminho de u até v e existe um caminho de v até u. No grafo da Figura 29, há caminhos que ligam cada par de vértices em ambos os sentidos.

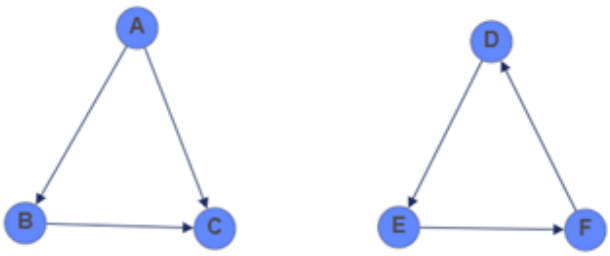
**Figura 29 –** Grafo Orientado f-Conexo




Fonte: Autor

- **Grafo não conexo:** Ao desconsiderar a orientação do grafo, haverá vértices que não se conectam, como ocorre com os vértices C e E do grafo da Figura 30.

**Figura 30 –** Grafo Orientado não Conexo




Fonte: Autor

### 2.3.11. GRAFOS ESPECIAIS

Além dos grafos completo, complementar e regular citados anteriormente, Boaventura Netto e Jurkiewicz (2009) relacionam como grafos especiais o grafo nulo ou vazio, o grafo bipartido e as árvores.

#### 2.3.11.1. GRAFO NULO OU VAZIO

É o grafo cujo conjunto de arestas é vazio.

Como exemplo, os times sorteados para compor o Grupo D da Copa Libertadores das Américas de 2019: FLA, SAN, LDU e PEN. Até o início da 1ª rodada, em que não ocorrera nenhum jogo, o grafo ficaria como o da figura.

**Figura 31 –** Grafo Nulo ou Vazio


Fonte: Autor

Observe que o complementar do grafo nulo ou vazio é o grafo completo. Em relação ao grafo da Figura 31, o grafo completo representaria todas as partidas envolvendo FLA, SAN, LDU e PEN.

#### 2.3.11.2. GRAFO BIPARTIDO

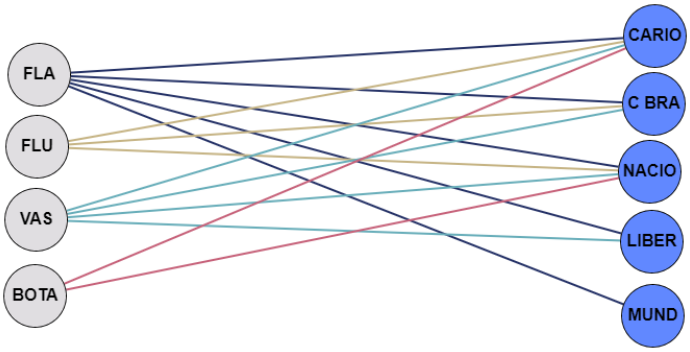
É um grafo cujo conjunto V de vértices pode ser particionado em dois subconjuntos disjuntos e independentes $V_1$ e $V_2$. As ligações (arestas ou arcos) se dão entre os vértices de $V_1$ e $V_2$ e nunca entre os vértices de um mesmo subconjunto. Para ilustrar, $V_1$ pode ser o subconjunto de jogadores do Flamengo e $V_2$ o subconjunto de times de futebol. Assim, uma aresta que liga um vértice de $V_1$ a um vértice de $V_2$ pode corresponder ao time de futebol que determinado jogador do Flamengo já atuou no passado.

Um caso especial de grafo bipartido é o grafo bipartido completo, em que todos os vértices de $V_1$ estão ligados a todos os vértices de $V_2$. Grafos desse tipo

são habitualmente designados pela notação $K_{p,q}$, onde p e q representam, respectivamente, o número de vértices dos subconjuntos $V_1$ e $V_2$. Logo, $K_{p,q}$ terá p x q arestas ou 2 x p x q arcos.

A Figura 32 ilustra um grafo bipartido formado pelos subconjuntos TIME e CAMPEONATO, onde as arestas representam os títulos de campeão obtidos por um determinado time em determinado campeonato.

**Figura 32 –** Grafo Bipartido




Fonte: Autor

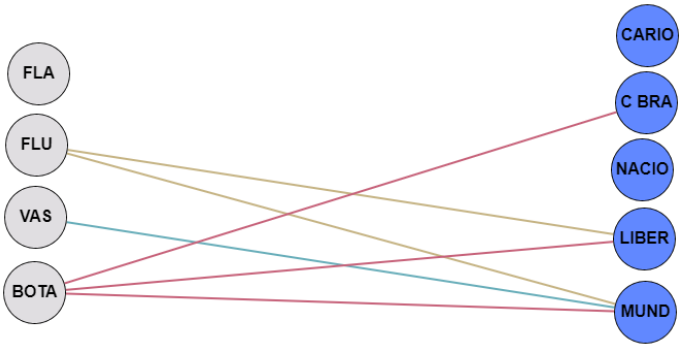
Os subconjuntos são formados pelos seguintes elementos:

- TIME = {FLA, FLU, VAS, BOTA};
- CAMPEONATO = {CARIO, C BRAS, NACIO, LIBER, MUND}

De acordo com o grafo, o Flamengo (FLA) foi campão de todos os campeonatos dos quais participou: Carioca (CARI), Copa do Brasil (C BRA), Campeonato Brasileiro (NACIO), Copa Libertadores das Américas (LIBER) e Campeonato Mundial de Clubes (MUND). O Vasco (VAS) ainda não possui o título do Campeonato Mundial de Clubes. O Fluminense (FLU) não ganhou nem a Copa Libertadores das Américas, nem o Campeonato Mundial de Clubes. E o Botafogo (BOTA), além de não ter ganho estes dois últimos campeonatos, também não tem o título da Copa do Brasil.

O grafo bipartido complementar ilustrado na Figura 33 representa a situação dos quatro clubes cariocas de expressão em relação aos títulos ainda não ganhos nos citados campeonatos.

**Figura 33 –** Grafo Bipartido Complementar




Fonte: Autor

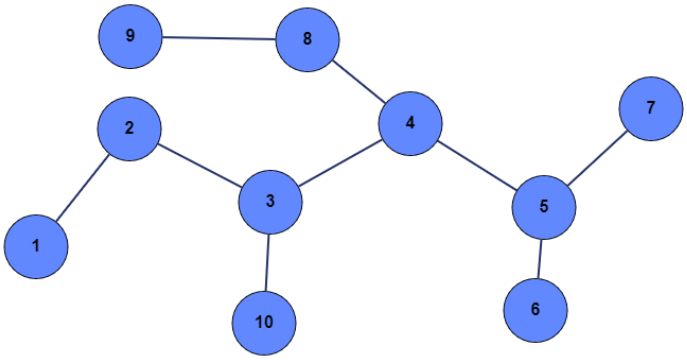
O melhor para os clubes cariocas seria que todos fossem campeões dos 5 campeonatos em referência. Tal feito seria representado pelo grafo bipartido completo.

### 2.3.11.3. ÁRVORES

As árvores são grafos conexos sem ciclos. Em toda árvore, sempre haverá vértices de grau 1, que corresponde ao grau mínimo da árvore. Esses vértices são chamados de folhas da árvore. Uma árvore com *n* vértices tem *n – 1* arestas (BOAVENTURA NETTO & JURKIEWICZ, 2009).

A Figura 34 ilustra uma árvore com 10 vértices e 9 arestas.

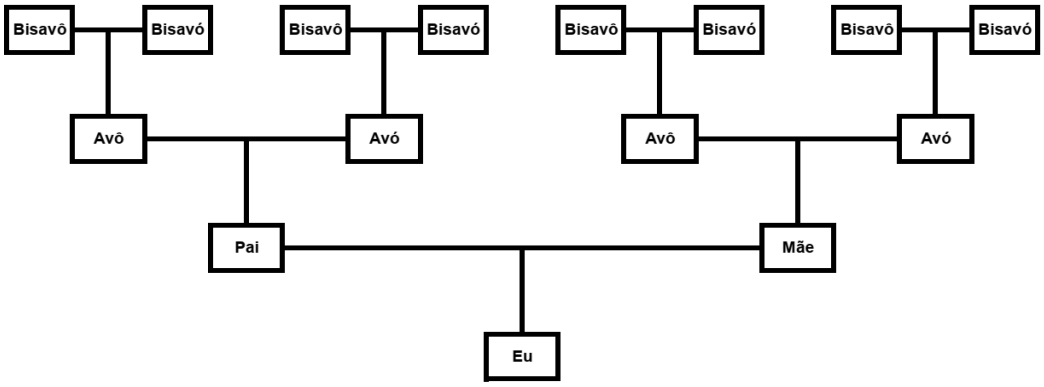
**Figura 34 –** Árvore




Fonte: Autor

Diversas situações do nosso cotidiano podem ser modeladas por meio de árvores. Uma delas é a árvore genealógica, ilustrada na Figura 35.

**Figura 35 –** Árvore Genealógica




Fonte: Autor

## 2.4. REPRESENTAÇÃO

São as formas de organizar os grafos, de modo que seus dados possam ser interpretados pelo computador, uma vez que problemas relacionados a grafos envolvem na maioria das vezes recursos computacionais para a sua solução (BOAVENTURA NETTO & JURKIEWICZ, 2009).

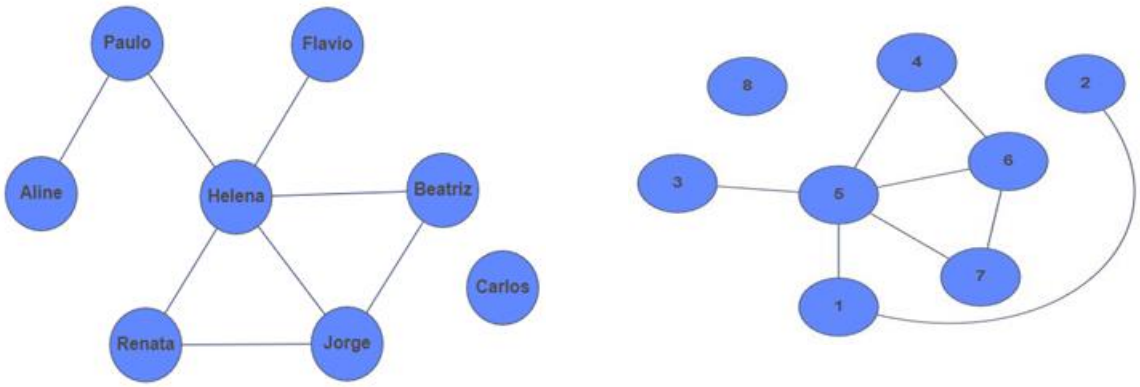
### 2.4.1. LISTA DE ADJACÊNCIA

Quando existe uma aresta (arco) ligando dois vértices dizemos que os vértices são adjacentes e que a aresta (arco) é incidente aos vértices.

Considere os seguintes grafos:

**Figura 36 –** Grafo para representação em Lista de Adjacência

a) b)




Fonte: Autor

No grafo da Figura 36b, a aresta (5,6) é incidente aos vértices 5 e 6. Logo, 5 e 6 são vértices adjacentes.

Já o vértice 8 não é adjacente a nenhum outro vértice, todavia, ele faz parte do grafo e representa algo. O mesmo ocorre no grafo da Figura 36a, grafo isomorfo ao grafo da Figura 36b, onde o vértice rotulado como “Carlos” não é adjacente a nenhum outro, indicando que Carlos não fez amizade com qualquer outro colega do curso, o que pode não significar absolutamente nada ou significar um possível problema comportamental do aluno. Este caso é um exemplo do uso do grafo na área da Sociologia[6] e Psicologia Social[7] (DA SILVEIRA JUNIOR *et al*., 2021).

A lista de adjacência indica quais vértices do grafo estão ligados ou são adjacentes a outro qualquer vértice, inclusive a ele mesmo.

O Quadro 3 mostra a lista de adjacência do grafo ilustrado na Figura 36b.

**Quadro 3 –** Lista de Adjacência

**LISTA DE ADJACÊNCIA**

| Vértice | Vértices Adjacentes | Vértice | Vértices Adjacentes |
|---|---|---|---|
| 1 | 2, 5 | 5 | 1, 3, 4, 6, 7 |
| 2 | 1 | 6 | 4, 5, 7 |
| 3 | 5 | 7 | 5, 6 |
| 4 | 5, 6 | 8 | - |

Fonte: Autor

### 2.4.2. MATRIZ DE ADJACÊNCIA

A matriz de adjacência é uma matriz cujos elementos representam a relação de adjacência entre os *n* vértices do grafo. Trata-se, portanto, de uma matriz quadrada de ordem n. A construção da matriz de adjacência difere para grafos orientados e não orientados.

[6] Estudo objetivo das relações sociais, isto é, das relações que só se estabelecem com fundamento na coexistência social, as quais se concretizam em normas, leis, valores e instituições consciente ou inconscientemente incorporadas pelos indivíduos que constituem a sociedade (Dicionário Aurélio).

[7] Ciência que estuda os comportamentos dos indivíduos considerados como tais, dentro do campo social, por ele influenciados, mas igualmente reagindo a ele e transformando-o (Dicionário Aurélio).

a) Grafo não orientado:

Seja A = [$a_{ij}$] a matriz de adjacência que representa o grafo não orientado G = (V, E). O valor de $a_{ij}$ dependerá da existência ou não de uma ligação entre os vértices i e j. Por convenção, o elemento $a_{ij}$ da matriz A receberá:

- 1, se os vértices i e j forem adjacentes
- 0, se os vértices i e j não forem adjacentes

Como em grafos não orientados a relação de adjacência é simétrica (os elementos $a_{ij}$ são iguais aos elementos $a_{ji}$), a matriz de adjacência será uma matriz simétrica, onde A = $A^t$. No caso de o grafo não conter laços, a matriz de adjacência terá os elementos da diagonal principal iguais a zero. A Figura 37 mostra a matriz de adjacência do grafo ilustrado na Figura 36b.

**Figura 37 –** Matriz de Adjacência de um Grafo não Orientado

A =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fonte: Autor

Observe que tanto os elementos da linha 8 como os elementos da coluna 8 da matriz de adjacência são nulos, o que equivale a dizer que não existe ligação do vértice 8 com qualquer outro vértice, caracterizando o grafo como não conexo.

b) Grafo orientado:

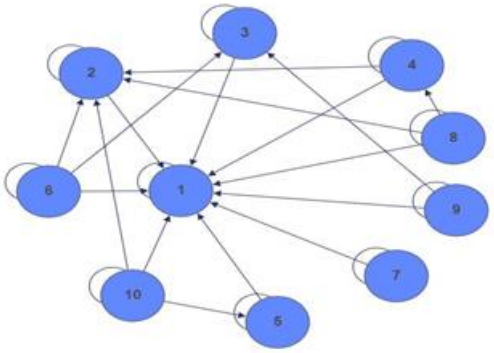
Seja A = [$a_{ij}$] a matriz de adjacência que representa o grafo orientado D = (V, A). O valor de $a_{ij}$ dependerá da existência ou não de uma ligação entre os vértices i e j e da orientação em que tais vértices estão ligados. Por convenção, o elemento $a_{ij}$ da matriz A receberá:

- 1, se os vértices i e j forem adjacentes, com orientação de i para j
- 0, nos demais casos

Portanto, a diferença entre as matrizes de adjacência de um grafo não orientado e de um grafo orientado está nos valores de seus elementos.

Considere o seguinte grafo da figura:

**Figura 38 –** Grafo orientado para representação em Matriz de Adjacência




Fonte: Autor

A Figura 39 mostra a matriz de adjacência do grafo orientado ilustrado na Figura 38.

**Figura 39 –** Matriz de Adjacência de um Grafo Orientado

$$A = \begin{array}{c|cccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 7 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 8 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 9 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 10 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

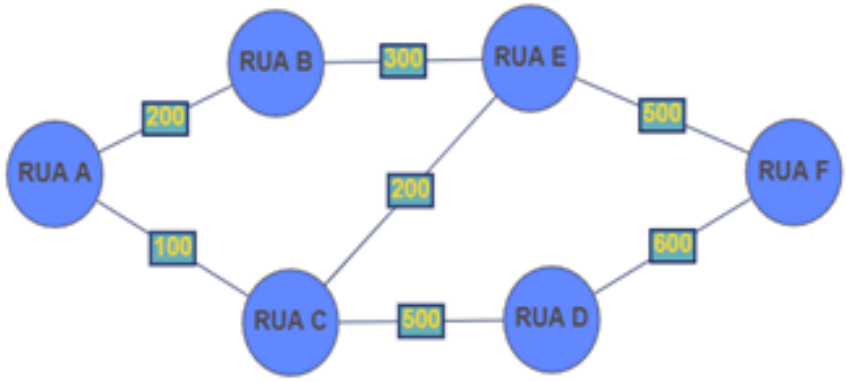
Fonte: Autor

### 2.4.3. MATRIZ DE VALORES

A matriz de valores é uma matriz análoga à matriz de adjacência, em que os elementos $a_{ij}$ assumirão o custo ou peso das arestas (distância, custo, tempo, vazão etc.) que ligam os vértices i ao j de um grafo ponderado[8]. Não havendo ligação entre os vértices i e j, $a_{ij}$ assumirá o valor igual a zero.

O grafo da Figura 40 representa as ruas de um conjunto habitacional recém-criado em Seropédica/RJ. Observe que as ruas são de mão e contramão. Um veículo ao ingressar no conjunto pela Rua A pode ir diretamente às Ruas B e C percorrendo, respectivamente, 200 metros e 100 metros. Essas distâncias representam os custos ou pesos das arestas. O mesmo veículo não pode passar pela Rua A e ir direto às Ruas D ou E, uma vez que não há ligação direta entre as ruas. Assim os custos das arestas {RUA A, RUA D} e {RUA A, RUA E} são iguais a zero.

**Figura 40 –** Grafo Ponderado para representação em Matriz de Valores




Fonte: Autor

A Figura 41 mostra a matriz de valores do grafo ponderado. As linhas e colunas em sua sequência representam, respectivamente, os vértices RUA A, RUA B, ..., RUA F do grafo.

[8] Grafo Ponderado é um grafo cujas arestas ou arcos são valorados, e estes valores representam distância, tempo, vazão, velocidade ou qualquer outra grandeza entre vértices adjacentes.

**Figura 41 –** Matriz de Valores de um Grafo Ponderado

$$A = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 200 & 100 & 0 & 0 & 0 \\ 2 & 200 & 0 & 0 & 0 & 300 & 0 \\ 3 & 100 & 0 & 0 & 500 & 200 & 0 \\ 4 & 0 & 0 & 500 & 0 & 0 & 600 \\ 5 & 0 & 300 & 200 & 0 & 0 & 500 \\ 6 & 0 & 0 & 0 & 600 & 500 & 0 \end{array}$$

Fonte: Autor

### 2.4.4. MATRIZ DE INCIDÊNCIA

A matriz de incidência é uma matriz n x m formada por n linhas que correspondem a cada um dos vértices do grafo, e por m colunas que correspondem a cada uma das ligações entre os vértices.

Por convenção, o elemento $a_{ij}$ da matriz de incidência $A = [a_{ij}]_{nxm}$ será:

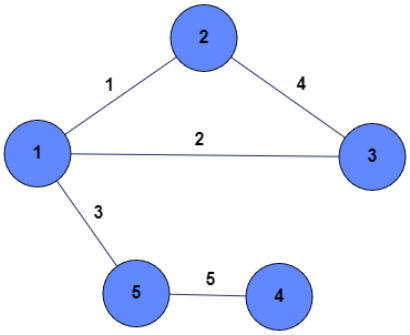
a) Grafo não orientado:
   - 1, quando o vértice i for incidente à aresta j
   - 0, caso contrário

b) Grafo orientado:
   - +1, quando o vértice i for de saída em relação ao arco j
   - -1, quando o vértice i for de entrada em relação ao arco j
   - 0, caso contrário aos anteriores

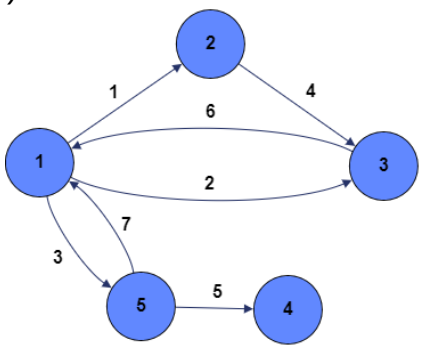
As matrizes de incidência ilustradas nas Figuras 43a e 43b representam, respectivamente, os grafos não orientado e orientado das Figuras 42a e 42b.

**Figura 42 –** Grafos para representação em Matriz de Incidência

a)




b)




Fonte: Autor

**Figura 43 –** Matrizes de Incidência

a)

$A_1 =$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 1 | 0 | 1 |

b)

$A_2 =$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | -1 | -1 |
| 2 | -1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | -1 | 0 | -1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 0 | -1 | 0 | 0 |
| 5 | 0 | 0 | -1 | 0 | 1 | 0 | 1 |

Fonte: Autor

O programa da Figura 44 imprime a lista de adjacência e as matrizes de adjacência, valores e incidência de um grafo ponderado, seja ele orientado ou não.

**Figura 44 –** Programa em Pascal para elaboração da Lista de Incidência e das Matrizes de Incidência, Adjacência e Valores

```
1   Program Representacao_de_Grafo;
2   Const
3     MAX = 100;
4   Type
5     matriz = array [0..MAX, 0..MAX] of real;
6     matriz1 = array [0..MAX, 0..MAX] of
7   integer;
8     vetor = array [0..MAX] of integer;
9   var
10    m_val: matriz;
11    m_adj, m_inc, l_adj: matriz1;
12    index: vetor;
13    lin, col, i, j, k, ordem, tam: integer;
14    custo: real;
15    tipo: char;
16
17  procedure inicio;
66  Procedure imprimir_matrizes_e_lista;
67  //Procedimento para imprimir as matrizes
67    de valores, adjacência e incidência e a
67    lista de adjacência criadas.
68  begin
69    writeln ('Matriz de Valores: ');
70    for i:=1 to ordem do
71    begin
72      for j:=1 to ordem do
73        write (m_val [i, j]:8:2);
74      writeln;
75    end;
76    writeln;
77
78    writeln ('Matriz de Adjacencia: ');
79    for i:=1 to ordem do
80    begin
```

```
18  //Torna nulo os valores dos vetores e das
18  matrizes
19  begin
20    for i:=1 to ordem do
21    begin
22      index [i] := 0;
23      for j:=1 to ordem do
24      begin
25        m_val [i, j] := 0;
26        m_adj [i, j] := 0;
27      end;
28
29      for k:=1 to tam do
30        m_inc [i, k] := 0;
31    end;
32  end;
33
34  Procedure Criar_matrizes_e_lista;
35  //Procedimento para criar as matrizes de
35   valores, adjacência e incidência e a lista de
35    adjacência.*
36  begin
37    m_val [lin, col] := custo;
38    m_adj [lin, col] := 1;
39    m_inc [lin, i] := 1;
40    index [lin] := index [lin] + 1;
41    l_adj [lin, index [lin]] := col;
42    if tipo = 'g' then
43    begin
44      m_val [col, lin] := custo;
45      m_adj [col, lin] := 1;
46      m_inc [col, i] := 1;
47      index [col] := index [col] + 1;
48      l_adj [col, index [col]] := lin;
49    end
50    else
51      m_inc [col, i] := -1;
52  end;
53
54  procedure ler_grafo;
55  //*procedimento para a leitura dos dados
55    envolvendo as arestas (arcos)
55    ponderadas do grafo (vértices de origem
55    e destino e custo). *
56  begin
57    write ('Vi: ');
58    readln (lin);
59    write ('Vj: ');
60    readln (col);
61    write ('w (',lin,', ',col, '): ');
62    readln (custo);
63    writeln;
64    Criar_matrizes_e_lista;
65  end;
```

```
81      for j:=1 to ordem do
82        write (m_adj [i, j]:8);
83      writeln;
84    end;
85    writeln;
86
87    writeln ('Matriz de Incidencia: ');
88    for i:=1 to ordem do
89    begin
90      for j:=1 to tam do
91        write (m_inc [i, j]:8);
92      writeln;
93    end;
94    writeln;
95
96    writeln ('Lista de Adjacencia: ');
97    for i:=1 to ordem do
98      If index [i] <> 0 then
99      begin
100       writeln ('Vertice ', i, ' :');
101       for j:=1 to index [i] do
102         write (l_adj [i, j]:8);
103       writeln;
104     end;
105   end;
106
107 //*Programa principal *
108 begin
109   Write ('Digite "D" para grafo orientado e
109           "G" para grafo nao orientado: ');
110   readln (tipo);
111   Write ('Digite a ordem n do grafo: ');
112   readln (ordem);
113   Write ('Digite o tamanho m do grafo: ');
114   readln (tam);
115   inicio;
116   writeln ('Dados da aresta (arco): Digite
116            o vertice de origem Vi, o vertice
116            de destino Vj e o custo da
116            aresta (arco) w (Vi, Vj)');
117   for i:=1 to tam do
118     ler_grafo;
119   imprimir_matrizes_e_lista;
120 end.
```

Fonte: Autor

O programa, ao iniciar, solicitará ao usuário o tipo de grafo: “D” para grafo dirigido e “G” para grafo não dirigido. Em seguida, solicitará a ordem *n* (número de vértices) e o tamanho *m* (número de arestas ou arcos) do grafo. O próximo passo será o programa principal chamar a ***procedure inicio***. Este procedimento criará as matrizes de adjacência de valores de ordem n e a matriz de incidência de ordem n x m, todas matrizes nulas. Criará também um indexador do tipo vetor, inicialmente nulo, que armazenará a quantidade de vizinhos de um vértice. Em seguida será executado m vezes a ***procedure ler_grafo***. Este procedimento permite ao usuário digitar os dados dos vértices $v_i$ e $v_j$ que formam as arestas $\{v_i, v_j\}$ ou os arcos $(v_i, v_j)$ do grafo e o custo das respectivas arestas (arcos) do grafo. A cada iteração, a ***procedure Criar_matrizes_e_lista*** será chamada, a fim de alterar os valores dos elementos das matrizes, de acordo com o que foi convencionado para a construção das matrizes visto nos itens “a” a “d” deste tópico. Concluídas as iterações, o programa principal chama a ***procedure imprimir_matrizes_e_lista*** *para* imprimir a lista de adjacência e as matrizes de adjacência, valores e incidência do grafo ponderado.

## 3. CAMINHO MÍNIMO EM GRAFOS

No capítulo 2 estudamos a definição de caminho em grafos, que consiste num conjunto de vértices conectados por uma sequência de arcos, todos diferentes entre si, em que tais arcos estão sob a mesma orientação.

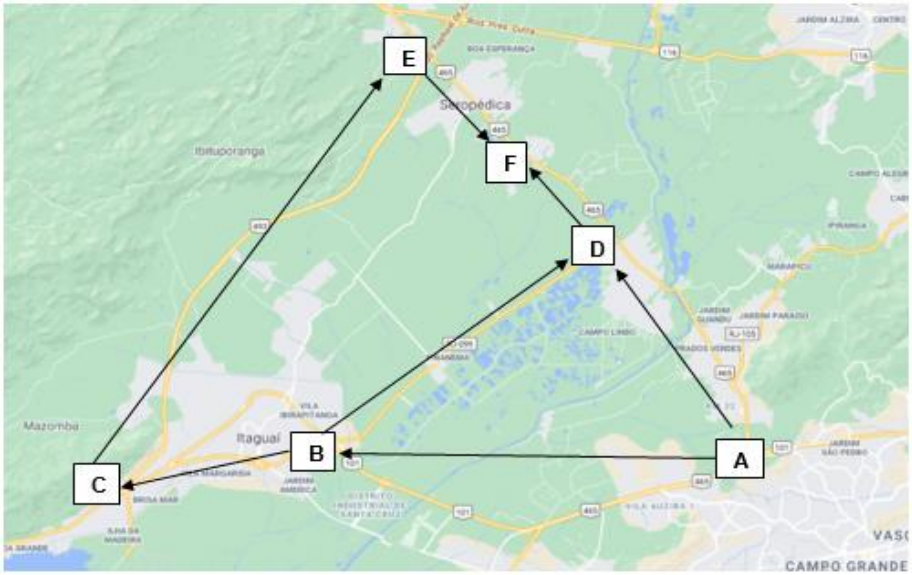
Estudamos, também, o significado de grafo ponderado, um grafo cujas arestas ou arcos são valorados, e estes valores representam distância, tempo, vazão ou qualquer outra grandeza entre vértices adjacentes.

Vamos, então, definir o que seja caminho mínimo.

Suponha que um motorista de UBER esteja em seu veículo, na Av. Brasil, na altura do bairro de Campo Grande (A), indo em direção à Universidade Federal Rural do Rio de Janeiro (F), e ele queira percorrer a menor distância até o seu destino.

O mapa da Figura 45 mostra os possíveis caminhos a serem tomados pelo motorista. Os pontos A, B, C, D, E e F representam localizações[9] no mapa.

**Figura 45 –** Mapa




Fonte: Autor

[9] A: localização atual do motorista; B: início da Reta de Piranema, em Itaguaí/RJ; C: saída da Rodovia Rio-Santos para o Arco Metropolitano, em Itaguaí/RJ; D: final da Reta de Piranema, em Seropédica/RJ; E: saída do Arco Metropolitano para a antiga Rodovia Rio-São Paulo, em Seropédica/RJ; F: Universidade Federal Rural do Rio de Janeiro.

Observe que o mapa pode ser representado por um grafo orientado e ponderado D = (V, A), cujos vértices representam coordenadas do mapa e os custos dos arcos $w(v_i, v_j)$, com $v_i, v_j \in V$, representam as distâncias em quilômetros (Km) entre dois vértices adjacentes. De acordo com o aplicativo *google maps*, os custos dos arcos são (Quadro 4):

**Quadro 4 –** Custos dos Arcos

| w(A, B) | w(A, D) | w(B, C) | w(B, D) | w(C, E) | w(D, F) | w(E, F) |
|---|---|---|---|---|---|---|
| 18,4 | 10,4 | 9,8 | 14,4 | 21,4 | 4,2 | 3,9. |

Fonte: Autor

O grafo mostra que há três caminhos que saem de A e chegam a F: (A, B, C, E, F), (A, B, D, F) e (A, D, F). Para conhecermos o custo total do vértice A até o vértice F, devemos somar os custos dos arcos conexos que saem de A e chegam a F. Assim:

- w(A, B, C, E, F) = w(A,B) + w(B,C) + w(C,E) + w(E,F) = 53,5
- w(A,B,D,F) = w(A,B) + w(B,D) + w(D,F) = 37
- w(A,D,F) = w(A,D) + w(D,F) = 14,6

Logo, o menor caminho ou caminho mínimo do vértice de origem A ao vértice de destino F é (A, D, F), caminho procurado pelo motorista do UBER.

Caso o custo do arco fosse uma medida de tempo, por exemplo, minutos, dependendo do dia e horário, o caminho mínimo poderia ser (A, B, C, E, F), representado o menor tempo entre o vértice de origem A e o vértice de destino F, uma vez que o trecho da antiga Rodovia Rio-São Paulo, representado pelo arco (A, D), é altamente congestionado em horários de pico.

Assim, conforme esclarece Boaventura Netto e Jurkiewicz (2009, p.38), o problema do caminho mínimo “consiste em encontrar o caminho de menor custo – por um critério dado – entre dois vértices quaisquer de um grafo G = (V, E), orientado ou não”.

O problema do caminho mínimo tem seu uso em áreas como engenharia de transporte, engenharia genética e ciências da computação.

Neste capítulo abordaremos problemas envolvendo caminhos mínimos, em um grafo D = (V, A) orientado e ponderado, entre um vértice de origem dado e os demais vértices de D, e os Algoritmos de Dijkstra e de Bellman-Ford utilizados para determiná-los.

## 3.1. ALGORITMO DE DIJKSTRA

O Algoritmo de Dijkstra, ilustrado na Figura 46, foi desenvolvido pelo cientista da computação Edsger Dijkstra (1930 – 2002) em 1956 e é utilizado para determinar o menor caminho entre um vértice s e os demais vértices de um grafo D = (V, A) ponderado com arcos de custo não negativo.

**Figura 46 –** Algoritmo de Dijkstra

```
1  Algoritmo Dijkstra (G(V,E), s);
2  // Entrada: (1) Lista de arestas do Grafo; (2) Vértice s de origem e t de destino; (3) Matriz w de
2     pesos das arestas
3  // Saída: Caminho mais curto de s até t
4
5  Iniício
6     para cada v ∈ V(G) faça
7        d[v] = ∞;
8        π[v] = null;
9     fim para;
10
11    d[s] = 0;
12    S = { };
13    Q = V(G);
14
15    enquanto Q <> Ø faça
16       u = Extrai-Min(Q);
17       S = S U {u};
18
19       para cada v ∈ Adj[u] faça
20          se d[v] > d[u] + w(u,v) então
21               d[v] = d[u] + w(u,v);
22               π[v] = u;
23          fim se;
24       fim para;
25    fim enquanto;
26 fim.
```

Fonte: Adaptado de Castro Junior, 2003

O algoritmo utiliza duas variáveis do tipo vetor d[v] e π[v]. A variável π[v] é o vértice pai ou predecessor do vértice v. Para se chegar a v, deve-se passar antes por π[v]. A variável d[v] é o custo do caminho que parte do vértice de origem s e chega em π[v], igual a d[π[v]], somado ao custo do arco (π[v], v).

Como o algoritmo é utilizado para encontrar caminhos de custos totais mínimos entre o vértice de origem s e o vértice v do grafo, inicialmente é atribuído a d[v] um valor muito alto (∞), tal que o algoritmo possa no decorrer da sua execução encontrar valores menores de d[v]. Assim, para todo v ∈ V, d [v] = ∞ e d [s] = 0, uma vez que a distância do vértice de origem a ele mesmo é nula.

O algoritmo utiliza duas variáveis do tipo registro, representando os conjuntos S e Q, onde S U Q = V e S ∩ Q = Ø. O conjunto S contém os vértices para os quais já se definiu, ao longo da execução do algoritmo, um caminho mínimo, e o conjunto Q contém os vértices para os quais ainda não se definiu um caminho mínimo e estará, ao longo da execução do algoritmo, ordenado de forma não-decrescente do custo d[v] e, em caso de igualdade dos custos, por sua ordem lexicográfica[10]. Inicialmente, S = Ø e Q = V, onde o primeiro elemento de Q é o vértice s, pois (d[s] = 0) < (∞ = d [v]).

O algoritmo possui comandos de repetição que serão executados enquanto houver vértice para o qual não se definiu um caminho mínimo, ou seja enquanto Q <> Ø (Q é diferente de vazio).

A cada iteração, extrai-se o 1º elemento do conjunto Q, o qual é atribuído a u e inserido no conjunto S. Assim, o conjunto S, de menores caminhos, é iniciado pelo vértice de origem s.

O próximo passo irá buscar nas adjacências dos vértices pertencentes a S aquele com menor distância relativa ao vértice de origem s. Para tal, o algoritmo implementará iterações envolvendo os vértices v adjacentes a u, em que se verificará se d[v] é maior que a distância w[u,v] + d[u]. Em caso afirmativo, o caminho do vértice inicial ao vértice v, passando por u, é menor que o anteriormente existente. Neste caso, d[v] passará a ter o valor de d[u] + w[u,v], e o vértice v terá um novo pai ou predecessor, o vértice u, ou seja, π[v] = u.

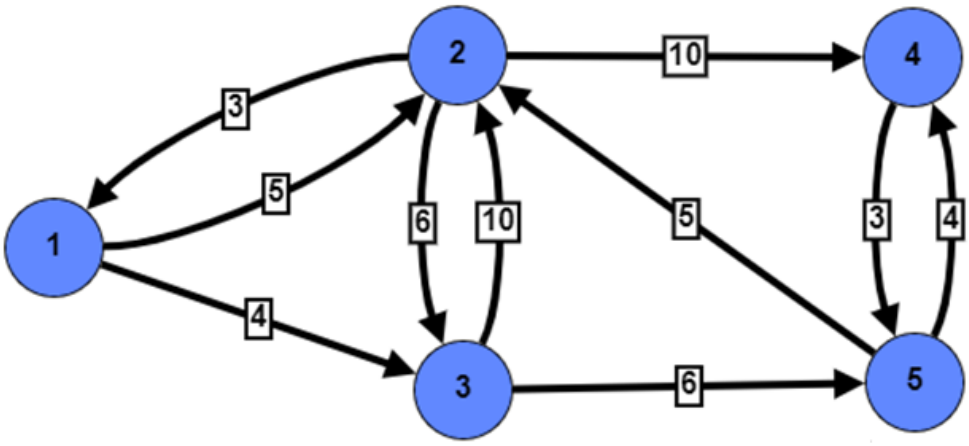
Após encerradas essas iterações, o conjunto Q é ordenado, sendo o

[10] Ordem Lexicográfica em matemática: Dados dois conjuntos parcialmente ordenados A e B, a ordem lexicográfica sobre o produto cartesiano A x B é definida como(a,b) ≤ (a′,b′) se e somente se a < a′ ou a = a′, e b ≤ b

primeiro elemento do conjunto o vértice mais próximo de s. O algoritmo retornará ao início do *loop*, para verificar a condição de Q. Quando Q se tornar vazio, o algoritmo se encerrará.

Para ilustrar o funcionamento do Algoritmo de Dijkstra, vamos determinar o caminho mínimo entre o vértice 1 e os demais vértices do grafo D = (V, A) da Figura 47.

**Figura 47 –** Problema Envolvendo Caminho Mínimo (Dijkstra)




Fonte: Autor

Para executar o algoritmo, inicialmente teremos que proceder à leitura dos custos de cada arco, conforme dados do Quadro 5:

**Quadro 5 –** Custos dos Arcos

| *w(1, 2)* | *w(1, 3)* | *w(2, 1)* | *w(2, 3)* | *w(2, 4)* | *w(3, 2)* | *w(3, 5)* | *w(4, 5)* | *w(5, 2)* | *w(5, 4)* |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 4 | 3 | 6 | 10 | 10 | 6 | 3 | 5 | 4 |

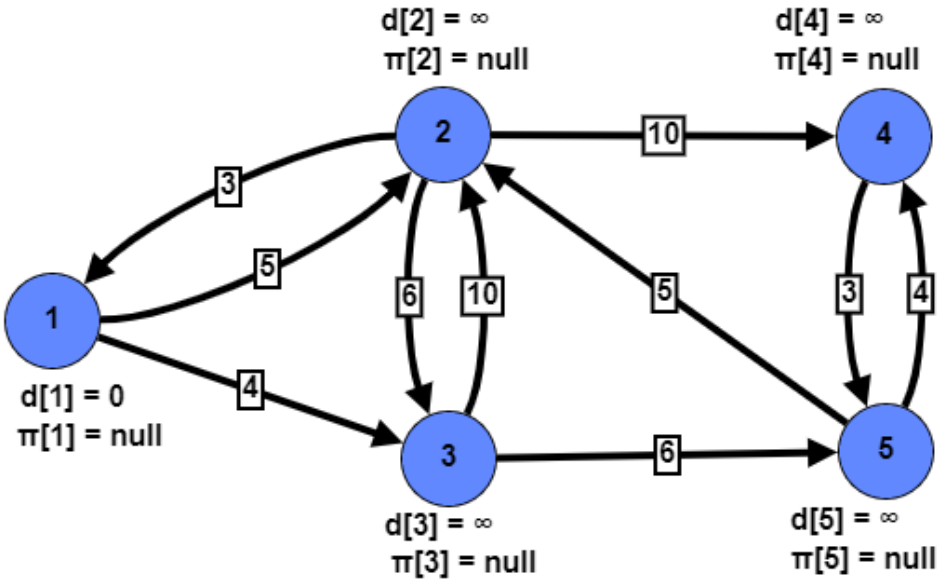
Fonte: Autor

Conforme enunciado do problema, tem-se s = 1.

Após a inicialização do algoritmo e antes da verificação de que o conjunto Q é ou não vazio, tem-se: S = { } e Q = {1, 2, 3, 4, 5}.

O grafo estaria assim representado (Figura 48):

**Figura 48 –** Representação do Grafo antes da Primeira Iteração




Fonte: Autor

Em seguida, o algoritmo verifica a condição de *Q* para a execução ou não do *loop*. Como Q não é vazio, os comandos de repetição serão executados.

**1ª Iteração:**

O 1º elemento de Q é extraído do conjunto, atribuído a u e inserido no conjunto S. Assim u = 1 e S = {1}.

O próximo passo será a implementação de iterações envolvendo os vértices v adjacentes a u, em que se verificará se o custo d[v] é maior que o custo do arco w[u,v] somado ao custo d[u]. Em caso afirmativo, d[v] passará a ter o valor de d[u] + w[u,v], e o vértice v terá um novo pai ou predecessor, o vértice u (π[v] = u).

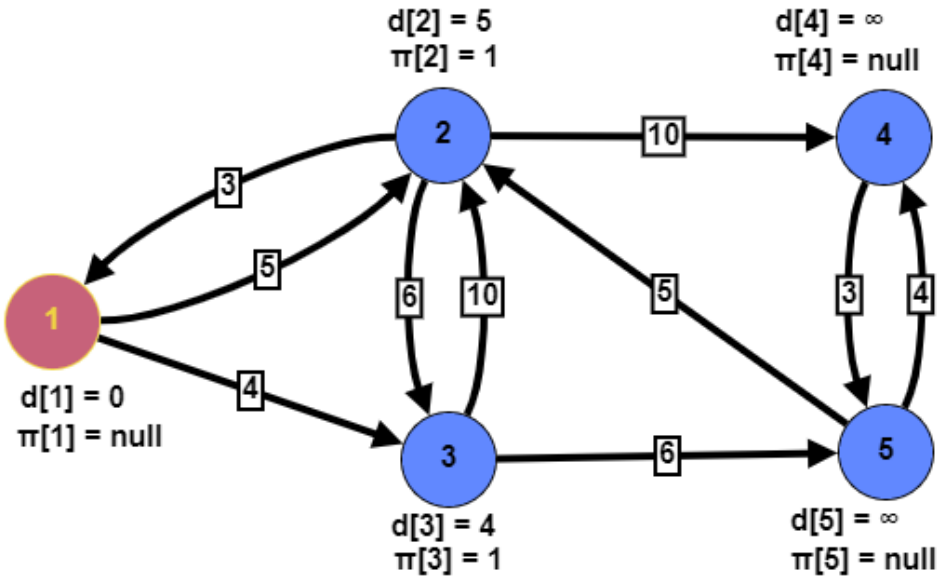
O vértice 1 tem como vértices adjacentes os vértices 2 e 3. Assim, teremos:

Para v = 2 ∴ (d[2] = ∞) > (d[1] + w[1,2] = 0 + 5 = 5), então d[2] = 5 e π[2] = 1;
Para v = 3 ∴ (d[3] = ∞) > (d[1] + w[1,3] = 0 + 4 = 4), então d[3] = 4 e π[3] = 1.

Após esses passos, tem-se: S = {1} e Q = {3, 2, 4, 5}.

O grafo estaria assim representado (Figura 49):

**Figura 49 –** Representação do Grafo após a Primeira Iteração




Fonte: Autor

Observe que não há como tornar menor a distância do vértice 1 ao vértice 3. Então, o arco (1,3) é o menor caminho entre os vértices. Após a ordenação de Q, o vértice 3 passará a ser o 1º vértice do conjunto e será extraído na próxima iteração, para ser inserido em S, uma vez que já temos o menor caminho da origem até ele.

Executados os comandos de repetição, o fluxo de comando é desviado para a verificação da condição de Q. Como Q não é vazio, o *loop* é novamente realizado.

**2ª Iteração:**

O 1º elemento de Q é extraído do conjunto, atribuído a u e inserido no conjunto S. Assim u = 3 e S = {1, 3}.

O próximo passo será a implementação de iterações envolvendo os vértices 2 e 5 adjacentes ao vértice 3. Assim, teremos:

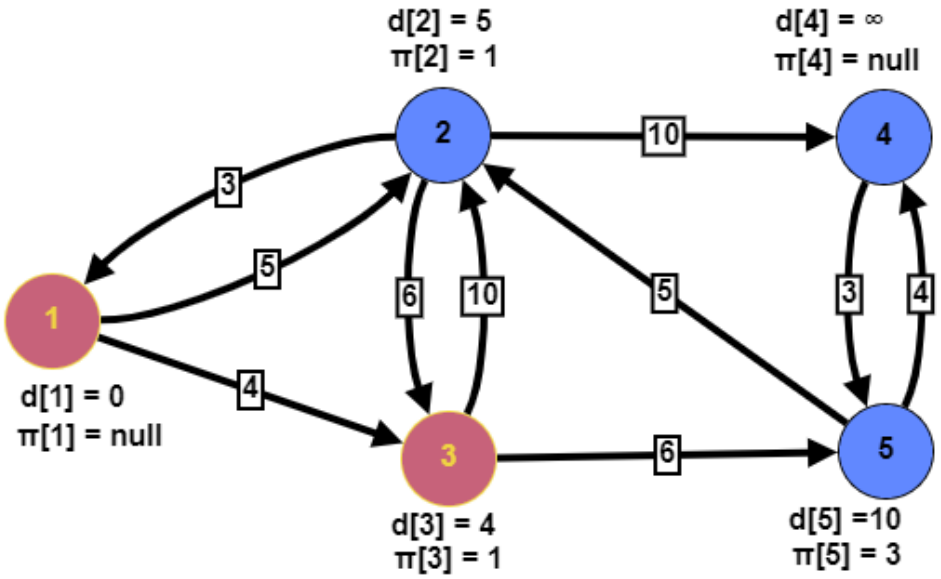
Para v = 2 ∴ (d[2] = 5) ⊁ (d[3] + w[3,2] = 4 + 10 = 14),
Para v = 5 ∴ (d[5] = ∞) > (d[3] + w[3,5] = 4 + 6 = 10), então d[5] = 10 e π[5] = 3.

Após a ordenação de Q, o vértice 2 passará a ser o 1º vértice do conjunto e será extraído na próxima iteração, para ser inserido em S, uma vez que já temos o menor caminho da origem até ele.

Após esses passos, tem-se: S = {1, 3} e Q = {2, 5, 4}.

O grafo estaria assim representado (Figura 50):

**Figura 50 –** Representação do Grafo após a Segunda Iteração




Fonte: Autor

Perceba que não há como tornar menor a distância do vértice 1 até o vértice 2. Então, o menor caminho até 2 é o arco (1, 2).

Após a execução dos comandos de repetição, o fluxo é desviado para a verificação da condição de Q. Como Q não é vazio, o *loop* é executado.

**3ª Iteração:**

O 1º elemento de Q é extraído do conjunto, atribuído a u e inserido no conjunto S. Assim u = 2 e S = {1, 3, 2}.

O próximo passo será a implementação de iterações envolvendo os vértices 1, 3 e 4 adjacentes ao vértice 2. Os vértices 1 e 3 já fazem parte do caminho mínimo (elemento de S), não tendo como melhorar a distância da origem até eles. Quanto ao vértice 4, tem-se:

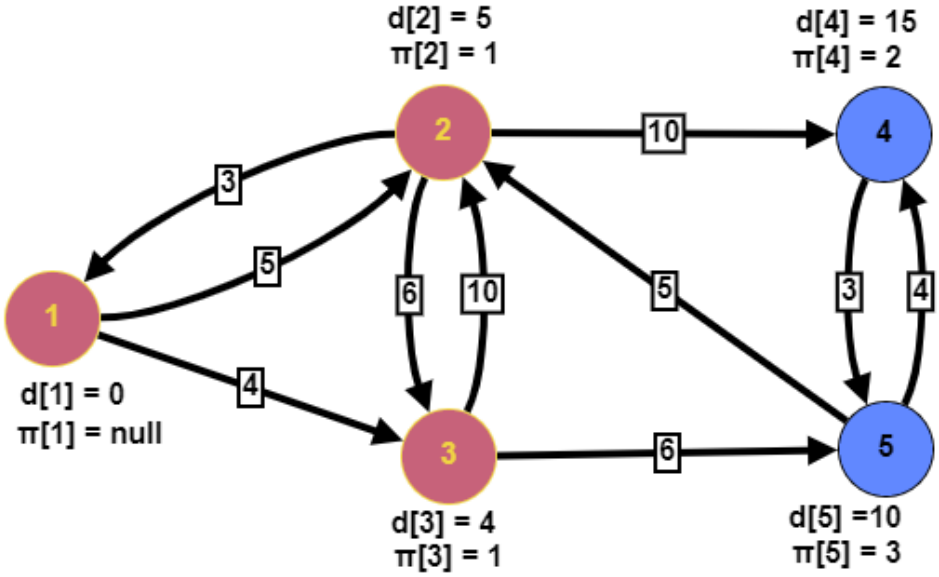
Para v = 4 ∴ (d[4] = ∞) > (d[2] + w[2,4] = 5 + 10 = 15), então d[4] = 15 e π[4] = 2.

Após a ordenação de Q, o vértice 5 passará a ser o 1º vértice do conjunto e será extraído na próxima iteração, para ser inserido em S, uma vez que já temos o menor caminho da origem até ele.

Após esses passos, tem-se: S = {1, 3, 2} e Q = {5, 4}.

O grafo estaria assim representado (Figura 51):

**Figura 51 –** Representação do Grafo após a Terceira Iteração




Fonte: Autor

Observe que não há como tornar menor a distância do vértice 1 até o vértice 5. Então, o menor caminho até 5 é o formado pelos arcos (1, 3) e (3, 5).

Concluída a execução dos comandos de repetição, o fluxo é desviado para a verificação da condição de Q. Como Q não é vazio, o *loop* é executado uma vez mais.

**4ª Iteração:**

O 1º elemento de Q é extraído do conjunto, atribuído a u e inserido no conjunto S. Assim u = 5 e S = {1, 3, 2, 5}.

O próximo passo será a implementação de iterações envolvendo os vértices 2 e 4 adjacentes ao vértice 5. O vértice 2 já faz parte do caminho mínimo (elemento de S), não tendo como melhorar a distância da origem até ele. Quanto ao vértice 4, tem-se:

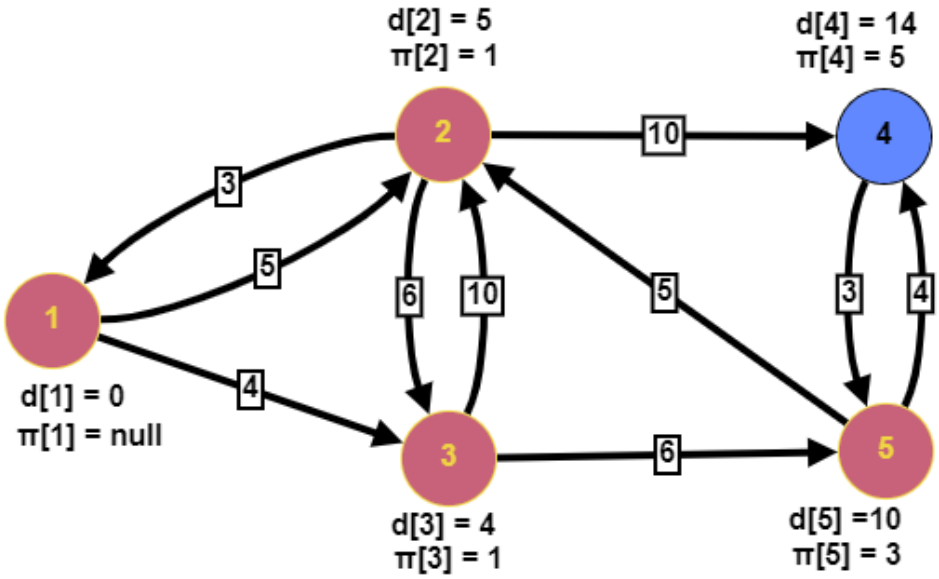
Para v = 4 ∴ (d[4] = 15) > (d[5] + w[5,4] = 10 + 4 = 14), então d[4] = 14 e π[4] = 5.

O conjunto Q passará a ter um único elemento (Q = {4}), que será extraído na próxima iteração, para ser inserido em S, uma vez que já temos o menor caminho da origem até ele.

Após esses passos, tem-se: S = {1, 3, 2, 5} e Q = {4}.

O grafo estaria assim representado (Figura 52):

**Figura 52 –** Representação do Grafo após a Quarta Iteração




Fonte: Autor

Perceba que não há como tornar menor a distância do vértice 1 até o vértice 4. Então, o menor caminho até 4 é o formado pelos arcos (1, 3), (3, 5) e (5, 4).

Concluída a execução dos comandos de repetição, o fluxo é desviado para a verificação da condição de Q. Como Q não é vazio, o *loop* é executado.

**5ª Iteração:**

O 1º elemento de Q é extraído do conjunto, atribuído a u e inserido no conjunto S. Assim u = 4 e S = {1, 3, 2, 5, 4}.

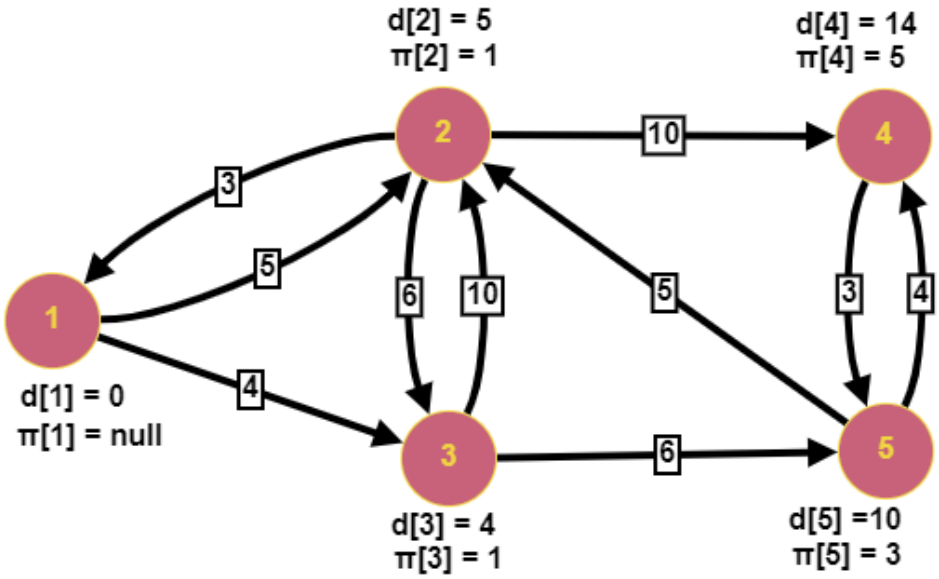
O próximo passo será a implementação de iterações envolvendo o vértice 5 adjacente ao vértice 4. O vértice 4 já faz parte do caminho mínimo (elemento de S), não tendo como melhorar a distância da origem até ele.

Q passará a ser um conjunto vazio.

Após esses passos, tem-se: S = {1, 3, 2, 5, 4} e Q = { }.

O grafo estaria assim representado (Figura 53):

**Figura 53 –** Representação do Grafo após a Quinta Iteração




Fonte: Autor

Após a execução dos comandos de repetição, o fluxo é desviado para a verificação da condição de Q. Como Q é vazio, o *loop* é encerrado e a execução do algoritmo é concluída.

Para saber, por exemplo, a menor distância entre os vértices 1 e 4, extrai-se o valor de d[4], que é igual a 14. Todavia, para o trajeto que corresponde ao caminho mínimo entre estes vértices, o Algoritmo de Dijkstra não nos imprime a resposta, o que é resolvido através do procedimento ilustrado na Figura 54 incluído

no algoritmo. Através dele, o caminho pode ser extraído facilmente através do vetor π, permitindo imprimir na tela do monitor o caminho do vértice de origem s a qualquer outro vértice do grafo.

**Figura 54 –** Algoritmo para Imprimir Caminho Mínimo

```
1  Procedimento Escreva_caminho (G, s, v);
2  início
3    se v = s então
4      escreva (s);
5    senão se π[v] = null
6      escreva (“não existe trajeto”);
7    senão
8      Escreva_caminho (G, s, π[v]);
9      escreva (v);
10   fim se;
11 fim;
```

Fonte: Adaptado de Loureiro & Goussevskaia, 2015

Na execução do procedimento, tem-se:

- 1º passo: Escreva_caminho (G, 1, 4),
- 2º passo: (1 <> 4) e (5 <> *null*) então Escreva_caminho (G, 1, 5),
- 3º passo: (1 <> 5) e (3 <> *null*) então Escreva_caminho (G, 1, 3),
- 4º passo: (1 <> 3) e (1 <> *null*) então Escreva_caminho (G, 1, 1),
- 5º passo: 1 = 1 então “1” [impressão do 1],
- 6º passo: retorna a Escreva_caminho (G, 1, 3),
- 7º passo: “3” [impressão do 3],
- 8º passo: retorna a Escreva_caminho (G, 1, 5),
- 9º passo: “5” [impressão do 5],
- 10º passo: retorna a Escreva_caminho (G, 1, 4),
- 11º passo: “4” [impressão do 4],
- 12º passo: o algoritmo é encerrado.

Assim, ao término da execução do procedimento, estará impresso na tela do monitor a sequência 1 3 5 4, representando o caminho mínimo do vértice 1 ao vértice 4.

Os códigos em Pascal do Algoritmo de Dijkstra ajustado pelo procedimento ***Escreva_caminho*** constam no programa ***Program Dijkstra*** (Figura 55), o qual tem sua versão completa no Apêndice A.

**Figura 55 –** Programa em Pascal baseado no Algoritmo de Dijkstra

```
1   Program Dijkstra;
2   uses
3     Crt; // A linha Uses Crt informa ao compilador que ele deve incluir a biblioteca Crt no seu programa
4           // Vai habilitar funções como gotoxy e clrscr
5   Const
6     MAX = 100;
7
8   Type
9     matriz = array [0..MAX, 0..MAX] of real;
10    matriz1 = array [0..MAX, 0..MAX] of integer;
11    vetor = array [0..MAX] of real;
12    vetor1 = array [0..MAX] of integer;
13
14  conjunto = record
15          tam: integer ;
16          vet: array [ 0..MAX+1] of integer ;
17  end;
18
19  var
20    S,Q: conjunto;
21    w: matriz;
22    mat_adj: matriz1;
23    d: vetor;
24    pai, vet_adj: vetor1;
25    lin, col, i, j, p, t, u, v, y, resp, ordem, tam, vert: integer;
26    custo: real;
27
28  procedure tela_inicial;
29  / / escreve na tela do monitor informações sobre o programa e recomendações ao usuário
...   (...)
40
41  procedure ler_grafo (var j: integer);
42  // procedimento para a leitura dos arcos ponderados do grafo (vértices de origem e destino e custo)
43  // gotoxy (x, y): representa a coordenada (x, y) do cursor na tela do monitor
...   (...)
55
56  Procedure Criar_grafo_ponderado;
57  // Procedimento para criar o grafo ponderado. O usuário informará a ordem e o tamanho do grafo.
57     Em seguida, o procedimento chamará
58  // o procedimento ler_grafo. Após a leitura dos dados, testes serão feitos para verificar se o vértice
58     de origem e/ou destino fazem parte
59  // do grafo, no caso as variáveis lin e col terão que ser menores ou iguais a ordem do grafo.  Os dados
59     serão armazenados em uma matriz
60  // de valores, a qual representará o grafo ponderado
...   (...)
131
132 Procedure Imprimir_Iteracao;
133 // Permite a impressão dos valores atualizados de d[i] e pai[i] em cada iteração
...   (...)
142
143 Procedure Escrever_caminho (vert: integer);
144 // O procedimento determina o caminho do vertice de origem s ao vertice de destino v
...   (...)
156
157 Procedure Determinar_caminho;
158 //Solicita dado do vértice de destino para imprimir o caminho até ele
...   (...)
169
170 Procedure definir_predecessor_inicial (ordem: integer; t:integer);
171 // Inicialização do Algoritmo de Dijkstra
...   (...)
182
183 procedure inicializar_conjunto (var c : conjunto);
```

```
184 // Cria um conjunto vazio.
... (...)
188
189 function conjunto_vazio (c : conjunto): boolean;
190 // Retorna true se o conjunto c eh vazio e false caso contrario
... (...)
194
195 function cardinalidade (c : conjunto) : integer;
196 // Retorna a cardinalidade do conjunto c
... (...)
200
201 Procedure ordenar_Q (var c: conjunto);
202 // Ordena os elementos do conjunto c de acordo com a o custo de d[v]
... (...)
217
218 procedure inserir_no_conjunto (x: integer ; var c : conjunto);
219 // Insere o elemento x (vértice) no conjunto c.
... (...)
227
228 procedure remover_do_conjunto (x: integer ; var c : conjunto);
229 // Remove o elemento x (vértice) do conjunto c
... (...)
237
238 // Programa Principal
239 begin
240   tela_inicial;
241   Criar_grafo_ponderado;
242   definir_predecessor_inicial(ordem, t);
243   inicializar_conjunto (S);
244   inicializar_conjunto (Q);
245
246   for i:=1 to ordem do       // insere os vértices no conjunto Q
247     inserir_no_conjunto (i, Q);
248
249   Ordenar_Q (Q);      // ordena Q de acordo com o custo de d[v]
250
251   y:=1;
252   while not conjunto_vazio (Q) do    // Executa as iterações enquanto Q não for vazio
253   begin
254     writeln (' Iteracao: ', y);
255     u := Q.vet[1]; //u recebe sempre o 1º elemento de Q
256     remover_do_conjunto (Q.vet[1], Q); //remove o 1º elemento de Q
257     inserir_no_conjunto (u, S); // u é inserido em S
258
259     for p:=1 to vet_adj[u] do //seleciona os vizinhos de u de acordo com a lista de adjacência
260     begin
261       v := mat_adj [u, p];
262       if d[v] > d[u] + w[u,v] then
263       begin
264         d[v] := d[u] + w[u,v];
265         pai[v] := u;
266       end;
267     end;
268     Ordenar_Q (Q); // ordena os elementos de Q em vista da alteração dos custos de d[v]
269     y:= y+1;
270     Imprimir_Iteracao; // imprime os valores atualizados de d[v] e pai[v] em cada iteração
271   end;
272   Determinar_caminho; // imprime o caminho do vértice s de origem ao vértice de destino v
273 end.
```

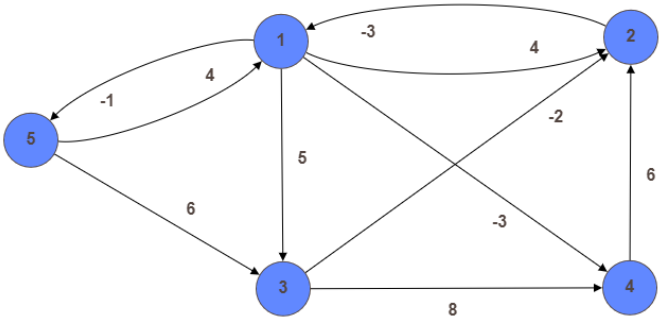
Fonte: Autor

## 3.2. ALGORITMO DE BELLMAN-FORD

No tópico anterior, apresentamos o Algoritmo de Dijkstra e mostramos o modo de determinamos o caminho mínimo em um grafo formado por arcos de custo não negativo. Vamos, agora, determinar o caminho mínimo em um grafo que possui arcos de custo negativo.

Seja o grafo D = (V, A) da Figura 56. Determinemos o caminho mínimo entre os vértices 5 e 4 do grafo utilizando o Algoritmo de Dijkstra.

**Figura 56 –** Problema Envolvendo Arcos de Custo Negativo (Dijkstra)




Fonte: Autor

Na execução do algoritmo, após cada iteração, as variáveis d[v] e π[v] assumirão os seguintes valores:

**1ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *4* | *∞* | *6* | *∞* | *0* |
| ***π*** | *5* | *NULL* | *5* | *NULL* | *NULL* |
| *S = {5} e Q = {1, 3, 2, 4}* | | | | | |

**2ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *4* | *8* | *6* | *1* | *0* |
| ***π*** | *5* | *1* | *5* | *1* | *NULL* |
| *S = {5, 1} e Q = {4, 3, 2}* | | | | | |

**3ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *4* | *7* | *6* | *1* | *0* |
| ***π*** | *5* | *4* | *5* | *1* | *NULL* |
| *S = {5, 1, 4} e Q = {3, 2}* | | | | | |

**4ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *4* | *4* | *6* | *1* | *0* |
| ***π*** | *5* | *3* | *5* | *1* | *NULL* |
| *S = {5, 1, 4, 3} e Q = {2}* | | | | | |

**5ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *1* | *4* | *6* | *1* | *0* |
| ***π*** | *2* | *3* | *5* | *1* | *NULL* |
| *S = {5, 1, 4, 3, 2} e Q = Ø* | | | | | |

Assim, a sequência 5 3 2 1 4 representa o caminho mínimo do vértice 5 ao vértice 4, de custo total igual a 1. Todavia, ao somarmos os custos de cada arco que compõe o caminho mínimo, constataremos que o custo total será diferente de 1. Pois vejamos:

Custo total = w(5,3) + w(3,2) + w(2,1) + w(1,4), então:

Custo total = 6 + (-2) + (-3) + (-3) = -2

Logo, o caminho mínimo do vértice 5 ao vértice 4 terá custo total igual a -2.

Esta diferença ocorre porque o Algoritmo de Dijkstra não tenta encontrar um caminho mais curto para vértices que já foram extraídos de Q. Caso o algoritmo previsse tal possibilidade, ao final da sua execução, teríamos:

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *1* | *4* | *6* | *-2* | *0* |
| ***π*** | *2* | *3* | *5* | *1* | *NULL* |

A limitação do Algoritmo de Dijkstra na determinação do caminho mínimo em grafos com arcos negativos é suprida pelo Algoritmo de Bellman-Ford (MORENO & RAMÍREZ, 2011).

O algoritmo da Figura 57 foi desenvolvido pelos matemáticos estadunidenses Richard Ernest Bellman (1920-1984) e Lester Randolph Ford (1886-1967) na década de 50 e apresenta o seguinte código:

**Figura 57 –** Algoritmo de Bellman-Ford

```
1  Algoritmo Bellman-Ford (G(V,E), s)
2  //Entrada: (1) Grafo G com pesos nas arestas; (2) Vértice s de origem; (3) Matriz w de pesos
2   das arestas
3  //Saída: Caminho mais curto de s até todos os demais vértices de G
4
5  inicio
6     para cada v ∈ V(G) faça
7        d[v] = ∞;
8        π[v] = null;
9     fim para;
10    d[s] = 0;
11
12    para i = 1 a n – 1 faça
13       para todo (u,v) ∈ E faça
14          se d[v] > d[u] + w(u,v) então
```

```
15            d[v] = d[u] + w(u,v);
16            π[v] = u;
17          fim se;
18        fim para;
19      fim para;
20
21      //Verificar ciclo negativo
22      Ciclo_negativo = falso;
23      para todo (u,v) ∈ E faça
24        se d[v] > d[u] + w(u,v) então
25          Ciclo_negativo = verdadeiro;
26        fim se;
27      fim para;
28  fim.
```

Fonte: Adaptado de Castro Junior, 2003

De modo semelhante ao Algoritmo de Dijkstra, o Algoritmo de Bellman-Ford trabalha com duas variáveis do tipo vetor d[v] e π[v]. A variável π[v] é o vértice pai ou predecessor de v. Para se chegar a v, deve-se passar antes por π[v]. A variável d[v] é o custo do caminho que parte do vértice de origem s e chega em π[v], igual a d[π[v]], somado ao custo do arco (π[v], v).

Como o algoritmo é utilizado para encontrar caminhos de custos totais mínimos entre o vértice de origem e o vértice v do grafo, inicialmente é atribuído a d[v] um valor muito alto (∞), tal que o algoritmo possa no decorrer da sua execução encontrar valores menores de d[v]. Assim, para todo v ∈ V, d [v] = ∞ e d [s] = 0, uma vez que a distância do vértice de origem a ele mesmo é nula. Este procedimento é idêntico ao do Algoritmo de Dijkstra.

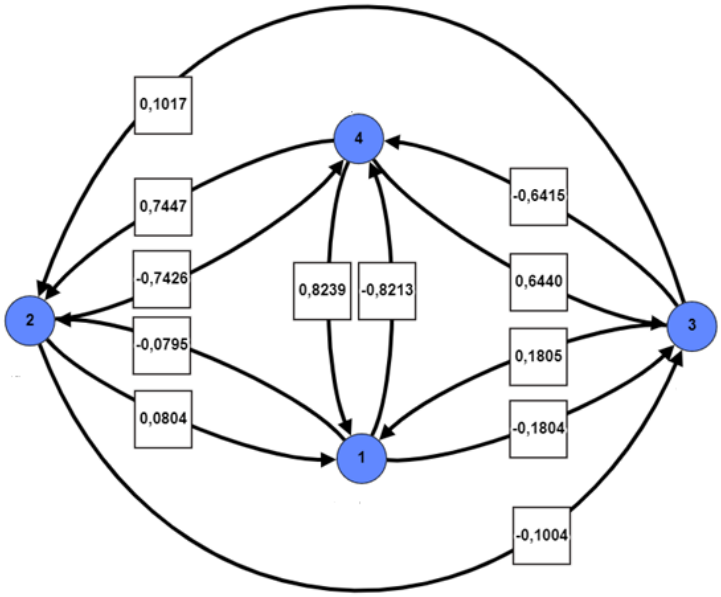
Para encontrar o caminho mínimo, a cada iteração, o algoritmo verifica se pode ser encontrado um caminho mais curto de um vértice de origem s até um vértice v passando pelo vértice corrente v'. Em caso positivo, atualiza-se o valor do menor caminho a v (d[v]), o qual terá como pai ou predecessor o vértice v' (π[v]), vizinho de v. Assim, para cada arco (u,v) do grafo representado na lista de adjacência, o algoritmo irá comparar o valor de d[v] com o valor de d[u] + w[u,v], onde w[u,v] é o custo do arco (u,v), e associará a d[v] o menor valor entre eles.

Como demonstrado por Moreno & Ramírez (2011), com no máximo de $n - 1$ iterações, sendo $n$ a ordem do grafo, é possível encontrar a solução do problema. Se a iteração-teste feita após o procedimento acima resultar em

mudanças dos custos, haverá um circuito negativo no grafo, indicando que as alterações não irão acabar, impossibilitando determinar o caminho mínimo entre s e v, não havendo, assim, solução para o problema.

Para ilustrar o funcionamento do Algoritmo de Bellmann-Ford, vamos determinar o caminho mínimo entre o vértice 4 e os demais vértices do grafo D = (V,A) da Figura 58.

**Figura 58 –** Problema Envolvendo Caminho Mínimo (Bellman-Ford)




Fonte: Autor

Para executar o algoritmo, inicialmente teremos que proceder à leitura dos custos de cada arco, conforme dados do Quadro 6:

**Quadro 6** – Custos dos Arcos

| w(1, 2) | w(1, 3) | w(1, 4) | w(2, 1) | w(2, 3) | w(2, 4) | w(3, 1) | w(3, 2) |
|---|---|---|---|---|---|---|---|
| -0,0795 | -0,1804 | -0,8213 | 0,0804 | -0,1004 | -0,7426 | 0,1805 | 0,1017 |

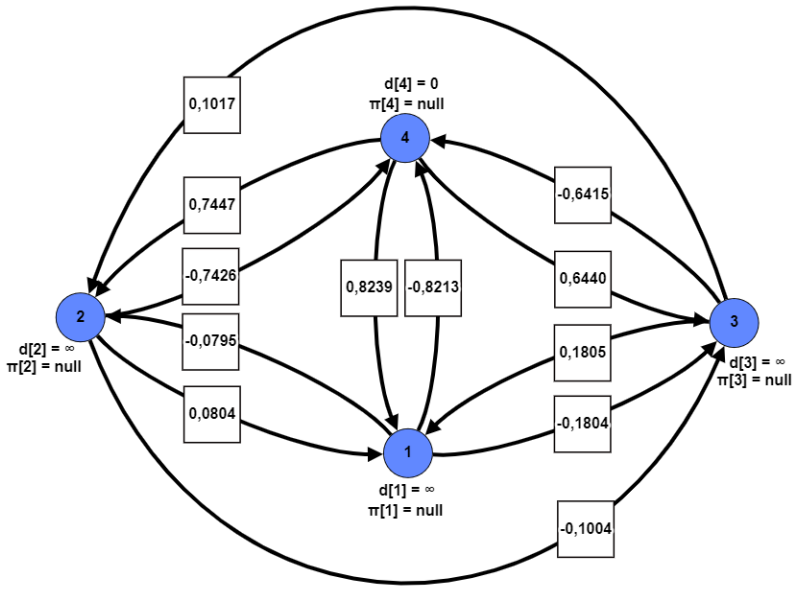
| w(3, 4) | w(4, 1) | w(4, 2) | w(4, 3) |
|---|---|---|---|
| -0,6415 | 0,8239 | 0,7447 | 0,6440 |

Fonte: Autor

Conforme enunciado do problema, s = 4.

Após a inicialização do algoritmo e execução dos comandos atribuídos às linhas de 6 a 10, tem-se a seguinte representação de grafo (Figura 59):

**Figura 59 –** Representação do Grafo antes da Primeira Iteração




Fonte: Autor

Sendo n = 4 a ordem do grafo, o algoritmo realizará 3 iterações para encontrar o caminho mínimo do vértice de origem 4 aos demais vértices do grafo, caso exista.

A execução das instruções do bloco de comando de repetição, a cada iteração, resultará nos valores de d[v] e π[v] ilustrados a seguir:

**1ª Iteração:**

Para i = 1, tem-se:

(1, 2) ∴ (d[2] = ∞) ⊁ (d[1] + w(1, 2) = ∞ - 0,0795 = ∞),

(1, 3) ∴ (d[3] = ∞) ⊁ (d[1] + w(1, 3) = ∞ - 0,1804 = ∞),

(1, 4) ∴ (d[4] = 0) ⊁ (d[1] + w(1, 4) = ∞ - 0,8213 = ∞),

(2, 1) ∴ (d[1] = ∞) ⊁ (d[2] + w(2, 1) = ∞ + 0,0804 = ∞),

(2, 3) ∴ (d[3] = ∞) ⊁ (d[2] + w(2, 3) = ∞ - 0,1004 = ∞),

(2, 4) ∴ (d[4] = 0) ⊁ (d[2] + w(2, 4) = ∞ - 0,7426 = ∞),

(3, 1) ∴ (d[1] = ∞) ⊁ (d[3] + w(3, 1) = ∞ + 0,1805 = ∞),

(3, 2) ∴ (d[2] = ∞) ⊁ (d[3] + w(3, 2) = ∞ + 0,1017 = ∞),

(3, 4) ∴ (d[4] = 0) ⊁ (d[3] + w(3, 4) = ∞ – 0,6415 = ∞),

(4, 1) ∴ (d[1] = ∞) > (d[4] + w(4, 1) = 0 + 0,8239 = 0,8239), então d[1] = 0,8239 e

π[1] = 4 [grafo modificado conforme Figura 60],

(4, 2) ∴ (d[2] = ∞) > (d[4] + w(4, 2) = 0 + 0,7447 = 0,7447), então d[2] = 0,7447 e

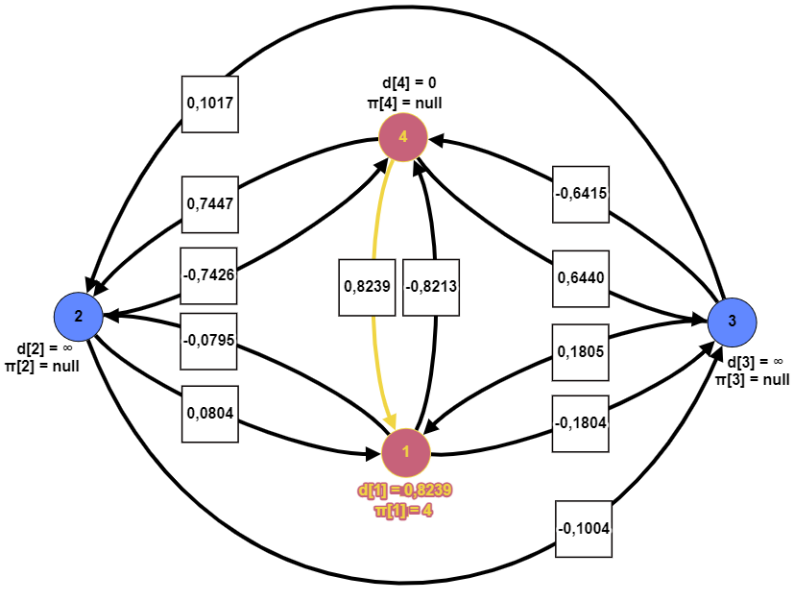
π[2] = 4 [grafo modificado conforme Figura 61],

(4, 3) ∴ (d[3] = ∞) > (d[4] + w(4, 3) = 0 + 0,6440 = 0,6440), então d[3] = 0,6440 e

π[3] = 4 [grafo modificado conforme Figura 62].

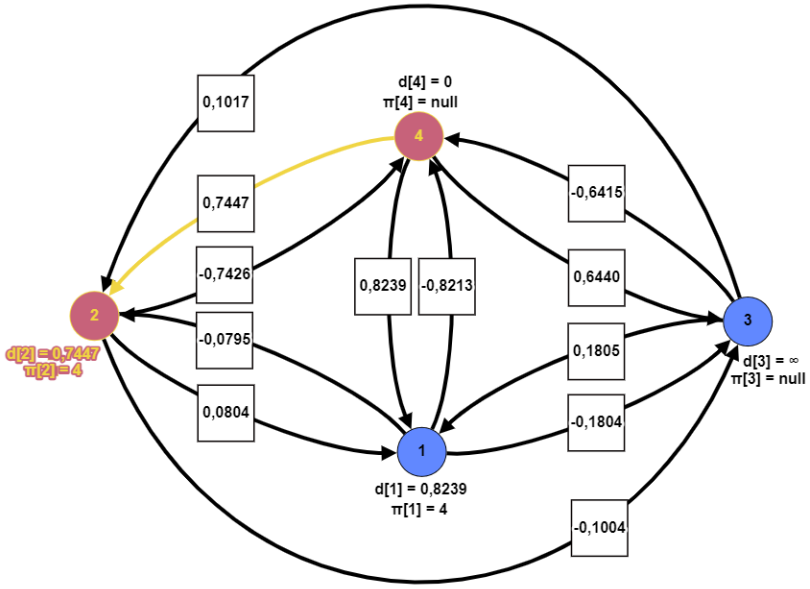
Os grafos das Figuras 60 a 62 ilustram a situação:

**Figura 60 –** Representação do Grafo após Análise do Arco (4, 1)



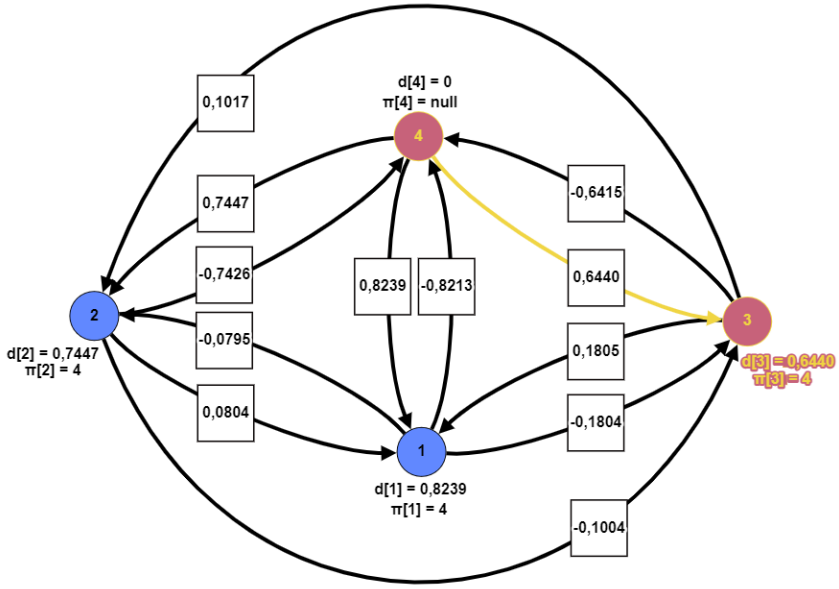

Fonte: Autor

**Figura 61 –** Representação do Grafo após Análise do Arco (4, 2)




Fonte: Autor

**Figura 62 –** Representação do Grafo após Análise do Arco (4, 3)




Fonte: Autor

Ao término da 1ª iteração, tem-se:

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** |
|---|---|---|---|---|
| ***d*** | 0,8239 | 0,7447 | 0,6440 | *0* |
| **π** | 4 | 4 | 4 | *null* |

**2ª Iteração:**

Para i = 2, tem-se:

(1, 2) ∴ (d[2] = 0,7447) > (d[1] + w(1, 2) = 0,8239 - 0,0795 = 0,7444), então
d[2] = 0,7444 e π[2] = 1 [grafo modificado conforme Figura 63],

(1, 3) ∴ (d[3] = 0,6440) > (d[1] + w(1, 3) = 0,8239 - 0,1804 = 0,6435), então
d[3] = 0,6435 e π[3] = 1 [grafo modificado conforme Figura 64],

(1, 4) ∴ (d[4] = 0) ⊁ (d[1] + w(1, 4) = 0,8239 - 0,8213 = 0,0026),

(2, 1) ∴ (d[1] = 0,8239) ⊁ (d[2] + w(2, 1) = 0,7444 + 0,0804 = 0,8248),

(2, 3) ∴ (d[3] = 0,6435) ⊁ d[2] + w(2, 3) = 0,7444 - 0,1004 = 0,6440),

(2, 4) ∴ (d[4] = 0) ⊁ (d[2] + w(2, 4) = 0,7444 - 0,7426 = 0,0018),

(3, 1) ∴ (d[1] = 0,8239) ⊁ (d[3] + w(3, 1) = 0,6435 + 0,1805 = 0,8240),

(3, 2) ∴ (d[2] = 0,7444) ⊁ (d[3] +w(3, 2) = 0,6435 + 0,1017 = 0,7452),

(3, 4) ∴ (d[4] = 0) ⊁ (d[3] +w(3, 4) = 0,6435 – 0,6415 = 0,0020),

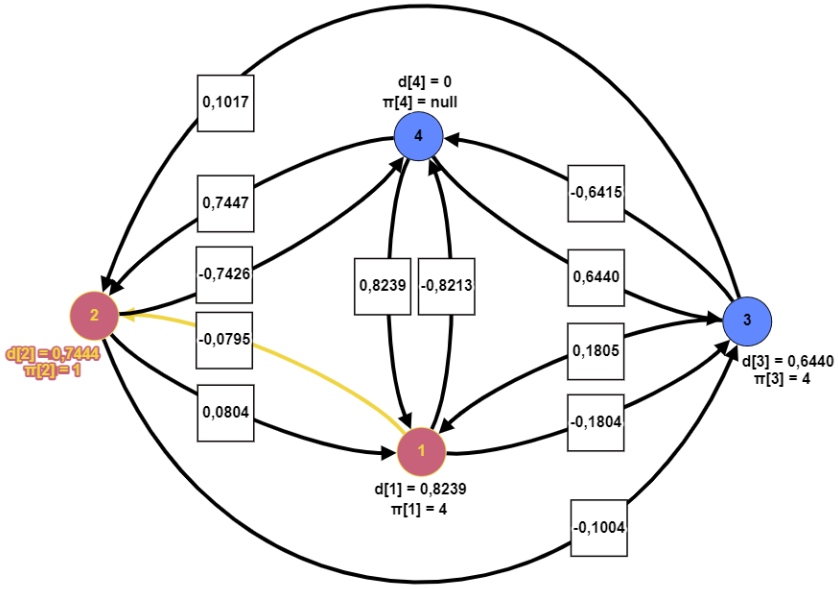
(4, 1) ∴ (d[1] = 0,8239) ⊁ (d[4] + w(4, 1) = 0 + 0,8239 = 0,8239),

(4, 2) ∴ (d[2] = 0,7444) ⊁ (d[4] + w(4, 2) = 0 + 0,7447 = 0,7447),

(4, 3) ∴ (d[3] = 0,6435) ⊁ (d[4] + w(4, 3) = 0 + 0,6440 = 0,6440).

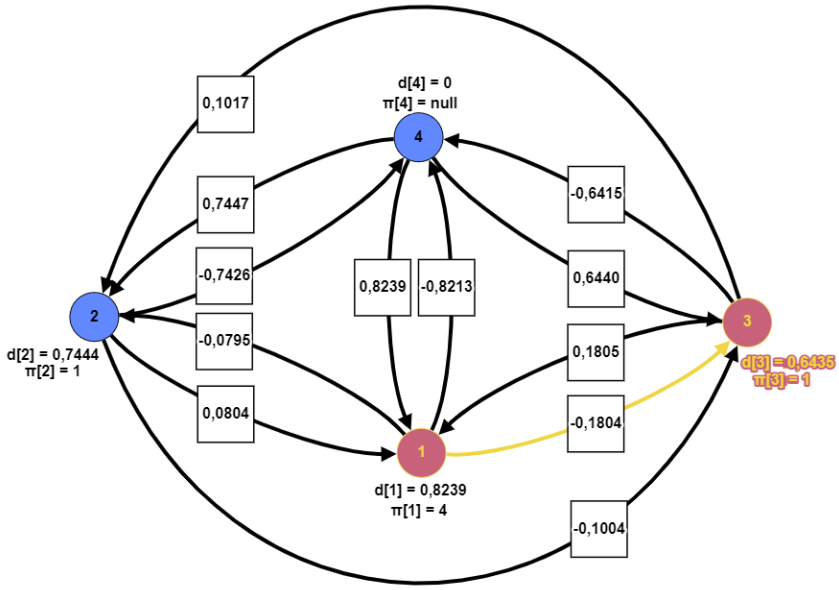
Os grafos das Figuras 63 e 64 ilustram a situação:

**Figura 63 –** Representação do Grafo após Análise do Arco (1, 2)




Fonte: Autor

**Figura 64 –** Representação do Grafo após Análise do Arco (1, 3)




Fonte: Autor

Ao término da 2ª iteração, tem-se:

| v | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| d | 0,8239 | 0,7444 | 0,6435 | 0 |
| π | 4 | 1 | 1 | *null* |

**3ª Iteração:**

Para i = 3, tem-se:

(1, 2) ∴ (d[2] = 0,7444) ⊁ (d[1] + w(1, 2)= 0,8239 - 0,0795 = 0,7444),

(1, 3) ∴ (d[3] = 0,6435) ⊁ (d[1] + w(1, 3) = 0,8239 - 0,1804 = 0,6435),

(1, 4) ∴ (d[4] = 0) ⊁ (d[1] + w(1, 4) = 0,8239 - 0,8213 = 0,0026),

(2, 1) ∴ (d[1] = 0,8239) ⊁ (d[2] + w(2, 1) = 0,7444 + 0,0804 = 0,8248),

(2, 3) ∴ d[3] = 0,6435) ⊁ d[2] + w(2, 3) = 0,7444 - 0,1004 = 0,6440),

(2, 4) ∴ (d[4] = 0) ⊁ (d[2] + w(2, 4) = 0,7444 - 0,7426 = 0,0018),

(3, 1) ∴ (d[1] = 0,8239) ⊁ (d[3] + w(3, 1) = 0,6435 + 0,1805 = 0,8240),

(3, 2) ∴ (d[2] = 0,7444) ⊁ (d[3] +w(3, 2) = 0,6435 + 0,1017 = 0,7452),

(3, 4) ∴ (d[4] = 0) ⊁ (d[3] +w(3, 4) = 0,6435 – 0,6415 = 0,0020),

(4, 1) ∴ (d[1] = 0,8239) ⊁ (d[4] + w(4, 1) = 0 + 0,8239 = 0,8239),

(4, 2) ∴ (d[2] = 0,7444) ⊁ (d[4] + w(4, 2) = 0 + 0,7447 = 0,7447),

(4, 3) ∴ (d[3] = 0,6435) ⊁ (d[4] + w(4, 3) = 0 + 0,6440 = 0,6440).

Ao término da 3ª e última iteração, tem-se:

| v | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| d | 0,8239 | 0,7444 | 0,6435 | 0 |
| π | 4 | 1 | 1 | *null* |

Como os valores de d[v] e π[v] após a 3ª alteração são os mesmos valores obtidos após a 2ª iteração, pode-se afirmar que os valores de d[v] e π[v] também serão os mesmos no teste a ser feito para a verificação de ciclo negativo. Em não havendo ciclo negativo, a solução do problema é o resultado obtido após a última iteração.

Para escrever ou registrar o caminho de um vértice de origem s a outro vértice v do grafo, pode-se utilizar o mesmo procedimento recursivo ilustrado na Figura 54. Logo, utilizando o algoritmo para escrever o caminho de vértice 4 ao

vértice 3, será escrito na tela do monitor o caminho 4 1 3, cujo custo total será igual a 0,6435 (d[3]).

Os códigos em Pascal do Algoritmo de Bellman_Ford ajustado pelo procedimento ***Escreva_caminho*** constam no programa ***Program Bellman_Ford*** (Figura 65), o qual tem sua versão completa no Apêndice B.

**Figura 65 –** Programa em Pascal baseado no Algoritmo de Bellman-Ford

```
1   Program Bellman_Ford;
2   uses
3     Crt;   // A linha Uses Crt informa ao compilador que ele deve incluir a biblioteca Crt no seu programa
4           // // Vai habilitar funções como gotoxy e clrscr
5   Const
6     MAX = 100;
7
8   Type
9     matriz = array [0..MAX, 0..MAX] of real;
10    vetor = array [0..MAX] of real;
11    vetor1 = array [0..MAX] of integer;
12
13    conjunto = record
14          tam: integer ;
15          vet: array [ 0..MAX+1] of integer ;
16    end;
17
18  var
19    ciclo_negativo: string;
20    w: matriz;
21    d: vetor;
22    pai: vetor1;
23    lin, col, i, j, k, t, resp, ordem, tam, vert: integer;
24    custo: real;
25
26  procedure tela_inicial;
27  // escreve na tela do monitor informações sobre o programa e recomendações ao usuário
...   (...)
38
39  procedure ler_grafo (var k: integer);
40  // procedimento para a leitura dos arcos ponderados do grafo (vértices de origem e destino e Custo)
41  // gotoxy (x, y): representa a coordenada (x, y) do cursor na tela do monitor
...   (...)
53
54  Procedure Criar_grafo_ponderado;
55  // Procedimento para criar o grafo ponderado. O usuário informará a ordem e o tamanho do grafo.
55     Em seguida o procedimento chamará
56  // o procedimento  ler_grafo. Após a leitura dos dados, testes serão feitos para verificar se o vértice
56     de origem e/ou destino fazem parte
57  // do grafo, no caso as variáveis lin e col terão que ser menores ou iguais a ordem do grafo. Os
57     dados serão armazenados em uma matriz
58  // de valores, a qual representará o grafo ponderado
...   (...)
124
125 Procedure Escrever_caminho (vert: integer);
126 // O procedimento determina o caminho do vertice de origem s ao vertice de destino v
...   (...)
138
139 Procedure Determinar_caminho;
140 // Solicita dado do vértice de destino para imprimir o caminho até ele
...   (...)
```

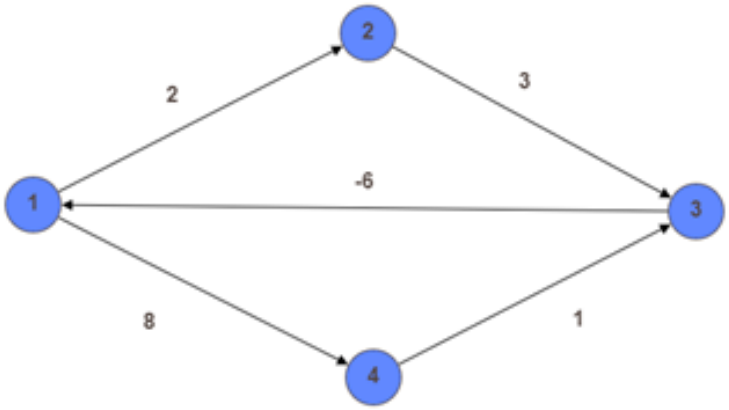
```
151
152  Procedure Imprimir_Iteracao;
153  // Permite a impressão dos valores atualizados de d[i] e pai[i] em cada iteração
 ...  (...)
162
163  Procedure definir_predecessor_inicial (ordem: integer; t:integer);
164  // Inicialização do Algoritmo de Bellman-Ford
 ...  (...)
175
176  Procedure Verificar_ciclo;
177  // Verifica a existência de ciclo negativo
 ...  (...)
195
196  //Programa Principal
197  begin
198    tela_inicial;
199    Criar_grafo_ponderado;
200    definir_predecessor_inicial(ordem, t);
201
202    for i:=1 to ordem -1 do   // Executa as iterações
203    begin
204      writeln (' Iteracao ', i);
205      writeln (' Arcos v d[v] pi[v]');
206      for j:= 1 to ordem do
207        for k:= 1 to ordem do
208          if w[j,k] <> 0 then
209          begin
210            if d[k] > (d[j] + w[j,k]) then
211            begin
212              d[k] := d[j] + w[j,k];
213              pai[k] := j;
214              if d[k] > 900000 then
215              begin
216                d[k]:=1000000;
217                pai[k] := 0;
218              end;
219            end;
220            writeln (' w[',j,',',k,'] ',k,' ', d[k]:9:4, ' ', pai[k]);
221          end;
222      writeln;
223      Imprimir_Iteracao;   // imprime os valores atualizados de d[v] e pai[v] em cada iteração
224      writeln;
225      writeln ('Digite ENTER para continuar');
226      readln;
227    end;
228
229    Verificar_ciclo;      // verfica a existência de ciclo negativo
230    if ciclo_negativo = 'NAO' then
231      Determinar_caminho;      // imprime o caminho do vértice s de origem ao vértice de destino v
232  end.
```

Fonte: Autor

Como mencionado na introdução deste tópico, o Algoritmo de Bellman-Ford realiza uma iteração-teste, a fim de verificar se o problema proposto tem solução. As mudanças dos custos ocorridas na iteração-teste indicam a existência de um circuito negativo no grafo, o que implica em alterações dos custos que jamais irão acabar, impossibilitando a determinação do caminho mínimo e, consequentemente, a solução do problema.

Para ilustrar, vamos determinar o caminho mínimo entre o vértice 1 e os demais vértices do grafo D = (V, A) da Figura 66.

**Figura 66 –** Problema Envolvendo Grafo com Ciclo Negativo (Bellman Ford)




Fonte: Autor

Na execução do algoritmo, após cada iteração, as variáveis d[v] e π[v] assumirão os seguintes valores:

**1ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** |
|---|---|---|---|---|
| ***d*** | *-1* | *2* | *5* | *8* |
| ***π*** | *3* | *1* | *2* | *1* |

**2ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** |
|---|---|---|---|---|
| ***d*** | *-2* | *1* | *4* | *7* |
| ***π*** | *3* | *1* | *2* | *1* |

**3ª Iteração**

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** |
|---|---|---|---|---|
| ***d*** | *-3* | *0* | *3* | *6* |
| ***π*** | *3* | *1* | *2* | *1* |

Quando da execução da iteração-teste, a variável Ciclo_negativo assumirá o valor “verdadeiro”, indicando que o grafo possui ciclo negativo, não havendo, portanto, solução para o problema. A sequência 1 – 2 – 3 – 1, de custo total igual a -1, representa o ciclo negativo do grafo.

Observe que, caso houvesse solução, esta se daria ao término da terceira iteração. Assim, o custo total do caminho mínimo entre os vértices 1 e 4 seria igual a 6. Todavia, o passeio 1 – 2 – 3 – 1 – 2 – 3 – 1 – 2 – 3 – 1 – 2 – 3 – 1 – 4 tem custo total igual a 4, inferior a 6. Aqui repetiu-se a sequência 1 – 2 – 3 – 1 quatro vezes. Repetindo a sequência um número maior de vezes e adicionando o vértice 4 à sequência, o custo total sempre diminuirá, indicando a falta de solução para o problema.

Como bem ressaltado por Moreno & Ramírez (2011), o Algoritmo de Bellman-Ford também pode ser utilizado para resolver problemas de caminho

mínimo em grafos com arcos não-negativos, todavia deve ser evitada a sua utilização, uma vez que é executado em um tempo maior que o Algoritmo de Dijkstra, tornado-se menos eficiente que este.

Isto ocorre porque o tempo de execução do Algoritmo de Dijkstra tem ordem de complexidade[11] $O(n^2)$; enquanto o tempo de execução do Algoritmo de Bellman-Ford, complexidade O(nm), onde n e m representam, respectivamente, a ordem e o tamanho do grafo. Em um grafo conexo, não definido como árvore, onde $m \geq n$, tem-se que $nm \geq n^2$, logo $O(nm) \geq O(n^2)$. Portanto, como regra geral, o Algoritmo de Dijkstra é mais eficiente que o Algoritmo de Bellman-Ford.

[11] Complexidade de tempo: medida que expressa a eficiência em termos de tempo de execução. Na análise de algoritmos verifica-se o número de operações consideradas relevantes que foram realizadas pelo algoritmo. Em seguida, expressa-se esse número como uma função de n, sendo n um parâmetro que caracteriza o tamanho da entrada do algoritmo (NOGUEIRA JÚNIOR, 2017). Quanto maior n maior será O(n).

## 4. ATIVIDADES PROPOSTAS

Um dos objetivos norteadores do ensino médio introduzidos pelo artigo 35 da Lei nº 9.394, de 20 de dezembro de 1996, conhecida como a Lei de Diretrizes e Bases da Educação Nacional (LDB), é “a compreensão dos fundamentos científico-tecnológicos dos processos produtivos, relacionando a teoria com a prática, no ensino de cada disciplina”.

Segundo Teixeira *et al.* (2017), ao trazer ao aluno o conteúdo do seu mundo real, o seu interesse é despertado naquilo que se está estudando, fazendo com que o relacionamento entre teoria e prática se traduza em uma aprendizagem significativa para o aluno.

Pretende-se, com os exercícios propostos, atingir este objetivo, fazendo com que, ao final do processo de resolução dos exercícios, os alunos dominem os conceitos básicos tanto da linguagem de programação Pascal como da Teoria dos Grafos, em especial, dos algoritmos voltados para resolução dos problemas envolvendo caminhos mínimos.

Devido às políticas de inclusão digital, acredita-se que os alunos terão acesso a computadores, notebooks ou tablets para a realização das tarefas, uma vez que precisarão trabalhar com Programas em Pascal. Como a linguagem está disponível para uso, também, em smartphones, acredita-se que os alunos, em última instância, poderão fazer uso do próprio aparelho, através de uma autorização prévia dos professores ou da direção da escola, nos estados da federação em que o uso é proibido.

Informações sobre a instalação e uso do software livre Pascal podem ser obtidos no Apêndice C.

## 4.1. CÁLCULO DA MÉDIA PONDERADA

Pretende-se que o aluno compreenda a noção de algoritmo, linguagem de programação e programa, em especial o Pascal, identificando a similaridade nas formas de estruturação do programa em Pascal (Figura 67.a) e do seu algoritmo (Figura 67.b). Deve-se abordar os conceitos elementares de fluxo de execução do programa, comandos de interação com o usuário (entrada e saída de dados), comandos de atribuição e uso de expressões que permitam a realização de cálculos aritméticos e lógicos, comandos de manipulação de dados da memória com o uso de variáveis e estruturas de repetição de comandos e de desvio do fluxo do programa.

**Figura 67 –** Algoritmo e Programa em Pascal para Cálculo da Média Ponderada

a)

```
1  Program Media_Disciplina;
2  var
3     i, n: integer;
4     nota, peso, soma, media: real;
5
6  begin
7     writeln ('Este programa calcula a media
7              das notas de um aluno numa
7              disciplina.');
8     writeln ('O aluno sera aprovado na
8              disciplina se a sua media nao
8              for inferior a 5,00');
9     writeln;
10    soma := 0;
11    write ('Informe a quantidade de
11           avaliacoes submetidas ao aluno: ');
12    readln (n);
13    writeln;
14
15    for i:=1 to n do
16    begin
17       write ('Entre com a nota da ', i, 'a.
17              avaliacao: ');
18       readln (nota);
19       write ('Entre com o peso da ', i, 'a.
19              avaliacao: ');
20       readln (peso);
21       soma := soma + nota * peso;
22       writeln;
23    end;
24
25    media := soma / n;
26
27    if media >= 5 then
28       writeln ('O aluno foi APROVADO com
```

b)

```
1  Algoritmo Media_Disciplina;
2  variáveis
3     i, n: inteiro;
4     nota, peso, soma, media: real;
5
6  início
7     escreva ("Este programa calcula a media
7              das notas de um aluno numa
7              disciplina.");
8     escreva ("O aluno sera aprovado na
8              disciplina se a sua media nao for
8              inferior a 5,00");
9     escreva ("  ");
10    soma = 0;
11    escreva ("Informe a quantidade de
11           avaliacoes submetidas ao aluno: ");
12    leia (n);
13    escreva (" ");
14
15    Para i=1 a n faça
16       escreva ("Entre com a nota da ", i, "a.
16                avaliacao: ");
17       leia (nota);
18       escreva ("Entre com o peso da ", i, "a.
18                avaliacao: ");
19       leia (peso);
20       soma = soma + nota * peso;
21       escreva ("  ");
22    fim para;
23
24    media = soma / n;
25
26    se media >= 5 então
27       escreva ("O aluno foi APROVADO com
27                media igual a ", media)
```

```
28              media igual a ', media:6:2)
29    else
30      writeln ('O aluno foi REPROVADO,
30              uma vez que teve media igual
30              a ', media:6:2);
31  end.
```

```
28    senão
29      escreva (“O aluno foi REPROVADO,
29              uma vez que teve media igual
29              a “, media);
30    fim se;
31  fim.
```

Fonte: Autor

O programa calcula a média das notas ponderadas pelos seus pesos obtidas pelo aluno numa disciplina. No final do ano-calendário, caso o aluno tenha obtido uma média não inferior a 5,00, ele será aprovado naquela disciplina; caso contrário, reprovado.

**Problema proposto**:

Em uma turma de Matemática do 1º ano do ensino médio 5 alunos ficaram de recuperação. Durante o ano letivo, eles realizaram 8 avaliações: duas provas de peso 2, quatro testes de peso 1, um trabalho de peso 1 e a prova de recuperação de peso 1. A média final do aluno será calculada com base nessas 8 avaliações. Caso o aluno tenha média não inferior a 5,00, ele ser aprovado na matéria; caso contrário, reprovado. As notas obtidas pelos alunos constam no Quadro 7:

**Quadro 7 –** Avaliações dos Alunos

| | Teste 1 | Teste 2 | Prova 1 | Teste 3 | Prova 4 | Trabalho 1 | Prova 2 | Prova de Recuperação |
|---|---|---|---|---|---|---|---|---|
| ANA | 5 | 4 | 3 | 5 | 4 | 7 | 6 | 10 |
| CARLOS | 6 | 5 | 5 | 5 | 3 | 6 | 3 | 6 |
| LUCAS | 4 | 5 | 4 | 4 | 5 | 6 | 5 | 9 |
| MARIA | 4 | 5 | 4 | 5 | 4 | 5 | 5 | 7 |
| PAULO | 7 | 6 | 5 | 4 | 3 | 5 | 4 | 9 |

Fonte: Autor

Propõe-se dividir a turma em grupo de 4 alunos, ficando cada grupo responsável por responder às perguntas sobre um determinado aluno constante do Quadro 7. Os grupos analisarão o código do programa ***Program Media_Disciplina*** e do respectivo algoritmo e responderão:

1- Qual a quantidade de avaliações submetidas ao aluno e qual a variável do programa que a armazena?

Os alunos foram submetidos a 8 avaliações entre provas, testes e trabalho. No programa ***Program Media_Disciplina***, a variável ***n***, do tipo inteira, armazena o valor da quantidade de avaliações digitada pelo usuário do programa.

2- Para cada iteração e após a inserção de cada uma das notas obtidas pelo aluno e do peso da avaliação, quais foram os valores armazenados nas variáveis *soma* e *peso_tot*?

Os valores das variáveis *soma* e *peso_tot* a cada iteração do programa constam no Quadro 8:

**Quadro 8 –** Valores Armazenados em Variáveis do Programa

| Aluno | Variáveis | i = 1 | i = 2 | i = 3 | i = 4 | i = 5 | i = 6 | i = 7 | i = 8 |
|---|---|---|---|---|---|---|---|---|---|
| ANA | soma | 5 | 9 | 15 | 20 | 24 | 31 | 43 | 53 |
| | peso_tot | 1 | 2 | 4 | 5 | 6 | 7 | 9 | 10 |
| CARLOS | soma | 6 | 11 | 21 | 26 | 29 | 35 | 41 | 47 |
| | peso_tot | 1 | 2 | 4 | 5 | 6 | 7 | 9 | 10 |
| LUCAS | soma | 4 | 9 | 17 | 21 | 26 | 32 | 42 | 51 |
| | peso_tot | 1 | 2 | 4 | 5 | 6 | 7 | 9 | 10 |
| MARIA | soma | 4 | 9 | 17 | 22 | 26 | 31 | 41 | 48 |
| | peso_tot | 1 | 2 | 4 | 5 | 6 | 7 | 9 | 10 |
| PAULO | soma | 7 | 13 | 23 | 27 | 30 | 35 | 43 | 52 |
| | peso_tot | 1 | 2 | 4 | 5 | 6 | 7 | 9 | 10 |

Fonte: Autor

3- Qual foi a média obtida pelo aluno?

A média dos alunos são as seguintes (Quadro 9):

**Quadro 9 –** Média dos Alunos

| | ANA | CARLOS | LUCAS | MARIA | PAULO |
|---|---|---|---|---|---|
| Média | 5,3 | 4,7 | 5,1 | 4,8 | 5,2 |

Fonte: Autor

4- O aluno foi aprovado ou reprovado em Matemática?

Os alunos Ana, Lucas e Paulo foram aprovados, pois a média de suas avaliações em Matemática foi superior a 5,00. Carlos e Maria foram reprovados, uma vez que a média de suas avaliações na disciplina foi inferior a 5,00.

## 4.2. CÁLCULO DAS MÉDIAS QUADRÁTICA, ARITMÉTICA, GEOMÉTRICA E HARMÔNICA

Pretende-se, através do programa ilustrado na Figura 68, que o aluno compreenda as noções de subprogramas ou sub-rotinas. Serão mostradas aos alunos as estruturas de uma função e de um procedimento, indicando a similaridade com a estrutura de um programa. O professor deverá abordar a finalidade do uso de cada tipo de subprograma, definir o que são variáveis locais e globais, esclarecer como um subprograma é chamado pelo programa principal ou por outro subprograma e como ocorre a passagem dos parâmetros.

**Figura 68 –** Algoritmo e Programa em Pascal para Cálculo de Médias

```
1  Program Media;
2  var
3    n: integer;
4    numero, mq, ma, mg, mh: real;
5    soma_arit, prod_geom, soma_harm,
5                              soma_quad: real;
6
7  procedure media_aritm (num: real;
7                           var soma_arit: real);
8  begin
9    soma_arit := soma_arit + num;
10 end;
11
12 procedure media_geomet (num: real;
12                          var prod_geom: real);
13 begin
14   prod_geom := prod_geom * num;
15 end;
16
17 procedure media_harmon (num: real;
17                          var soma_harm: real);
18 begin
19   soma_harm := soma_harm+(1/num);
20 end;
21
22 procedure media_quadrat (num: real;
22                          var soma_quad: real);
23 begin
24  // sqr (x) retorna o quadrado de um
24 número racional x.
25   soma_quad := soma_quad + sqr (num);
26 end;
27
28 begin
29   n := 0;
30   soma_arit := 0;
31   prod_geom := 1;
```

```
1  Algoritmo Media;
2  variaveis
3    n: inteiro;
4    numero, mq, ma, mg, mh: real;
5    soma_arit, prod_geom, soma_harm,
5                              soma_quad: real;
6
7  procedimento media_aritm (num: real;
7                           variável soma_arit: real);
8  inicio
9    soma_arit = soma_arit + num;
10 fim;
11
12 procedimento media_geomet (num: real;
12                          variável prod_geom: real);
13 inicio
14   prod_geom = prod_geom * num;
15 fim;
16
17 procedimento media_harmon (num: real;
17                          variável soma_harm: real);
18 inicio
19   soma_harm = soma_harm+(1/num);
20 fim;
21
22 procedimento media_quadrat (num: real;
22                          variável soma_quad: real);
23 inicio
24   soma_quad = soma_quad + num * num;
25 fim;
26
27 inicio
28   n = 0;
29   soma_arit = 0;
30   prod_geom = 1;
31   soma_harm = 0;
32   soma_quad = 0;
```

```
   soma_harm := 0;
   soma_quad := 0;
   writeln ('Digite os numeros para o
                        calculo das medias');
   write ('Numero: ');
   readln (numero);

    while numero > 0 do
    begin
      n := n + 1;
      media_aritm (numero, soma_arit);
      media_geomet (numero,
                              prod_geom);
      media_harmon (numero,
                             soma_harm);
      media_quadrat (numero,
                            soma_quad);
      write ('Numero: ');
      readln (numero);
    end;

// sqrt (x retorna a raiz quadrada do
número racional x.
// exp (ln (x)/n) retorna a raiz n-ésima de
x.
   mq := sqrt (soma_quad/n);
   ma := soma_arit/n;
   mg :=  exp (ln (prod_geom)/n):
   mh := n/soma_harm;

// cada número a ser impresso ocupará
um espaço de 10 caracteres e terá 2
casas decimais.

   writeln ('Media quadratica: ', mq:10:2);
   writeln ('Media aritmetica: ', ma:10:2);
   writeln ('Media geometrica: ', mg:10:2);
   writeln ('Media harmonica: ', mh:10:2);
end.
```

```
   escreva ("Digite os numeros para o calculo
                  das medias");
   escreva ("Numero: ");
   leia (numero);

   enquanto numero > 0 faça
      n = n + 1;
      media_aritm (numero, soma_arit);
      media_geomet (numero, prod_geom);
      media_harmon (numero, soma_harm);
      media_quadrat (numero, soma_quad);
      escreva ("Numero: ");
      leia (numero);
   fim enquanto;

   // a ^ b equivale a elevado a b
   mq =  (soma_quad/n) ^ (1/2);
   ma = soma_arit/n;
   mg =   (prod_geom) ^ (1/n);
   mh = n/soma_harm;

  escreva ("Media quadratica: ", mq);
   escreva ("Media aritmetica: ", ma);
   escreva ("Media geometrica: ", mg);
   escreva ("Media harmonica: ",mh;
fim.
```

Fonte: Autor

**Problema proposto:**

Calcular as médias aritmética, geométrica, harmônica e quadrática entre os seguintes números:

a) 1, 3 e 6

b) 2, 2, 4 e 16

c) 1, 1, 2, 3, 5

d) 10 e 10

e) 6, 6, 6 e 6

f) 3, 3 e 3

Propõe-se dividir a turma em grupo de 4 alunos. Cada grupo deverá resolver uma das questões relacionadas aos itens de “a” a “c” e uma das questões relacionadas aos itens de “d” a “f”. Deve-se ressaltar aos grupos que o encerramento do programa dar-se-á pela digitação de um número menor ou igual a zero. Os grupos analisarão o código do programa ***Program Media_Disciplina*** e do respectivo algoritmo e responderão as perguntas propostas. Com base nos resultados apresentados pelos grupos, os quais deverão ser expostos no quadro negro, deve-se solicitar aos alunos que comparem os valores das médias de cada sequência e informem a conclusão a que chegaram sobre a relação entre as médias.

1- Quais os valores das variáveis *n*, *soma_arit, prod_geom, soma_harm e soma_quad* a cada chamada dos procedimentos ***media_aritm, media_geomet, media_harmon*** e ***media_quadrat*** no intervalo entre a leitura do primeiro e do último número inteiro da sequência?

Os valores das variáveis constam no Quadro 10.

**Quadro 10 –** Valores Armazenados em Variáveis do Programa

***item a)***

| Variáveis | Valores | | | |
|---|---|---|---|---|
| num | | 1 | 3 | 6 |
| n | 0 | 1 | 2 | 3 |
| soma_arit | 0 | 1,00 | 4,00 | 10,00 |
| prod_geom | 1 | 1,00 | 3,00 | 18,00 |
| soma_harm | 0 | 1,00 | 1,33 | 1,50 |
| soma_quad | 0 | 1,00 | 10,00 | 46,00 |

***item b)***

| Variáveis | Valores | | | | |
|---|---|---|---|---|---|
| num | | 2 | 2 | 4 | 16 |
| n | 0 | 1 | 2 | 3 | 4 |
| soma_arit | 0 | 2,00 | 4,00 | 8,00 | 24,00 |
| prod_geom | 1 | 2,00 | 4,00 | 16,00 | 256,00 |
| soma_harm | 0 | 0,50 | 1,00 | 1,25 | 1,31 |
| soma_quad | 0 | 4,00 | 8,00 | 24,00 | 280,00 |

***item c)***

| Variáveis | Valores | | | | | |
|---|---|---|---|---|---|---|
| num | | 1 | 1 | 2 | 3 | 5 |
| n | 0 | 1 | 2 | 3 | 4 | 5 |
| soma_arit | 0 | 1,00 | 2,00 | 4,00 | 7,00 | 12,00 |
| prod_geom | 1 | 1,00 | 1,00 | 2,00 | 6,00 | 30,00 |
| soma_harm | 0 | 1,00 | 2,00 | 2,50 | 2,83 | 3,03 |
| soma_quad | 0 | 1,00 | 2,00 | 6,00 | 15,00 | 40,00 |

***item d)***

| Variáveis | Valores | | |
|---|---|---|---|
| num | | 10 | 10 |
| n | 0 | 1 | 2 |
| soma_arit | 0 | 10,00 | 20,00 |
| prod_geom | 1 | 10,00 | 100,00 |
| soma_harm | 0 | 0,10 | 0,20 |
| soma_quad | 0 | 100,00 | 200,00 |

***item e)***

| Variáveis | Valores | | | | |
|---|---|---|---|---|---|
| num | | 5 | 5 | 5 | 5 |
| n | 0 | 1 | 2 | 3 | 4 |
| soma_arit | 0 | 5,00 | 10,00 | 15,00 | 20,00 |
| prod_geom | 1 | 5,00 | 25,00 | 125,00 | 625,00 |
| soma_harm | 0 | 0,20 | 0,40 | 0,60 | 0,80 |
| soma_quad | 0 | 25,00 | 50,00 | 75,00 | 100,00 |

***Item f)***

| Variáveis | Valores | | | |
|---|---|---|---|---|
| num | | 3 | 3 | 3 |
| n | 0 | 1 | 2 | 3 |
| soma_arit | 0 | 3,00 | 6,00 | 9,00 |
| prod_geom | 1 | 3,00 | 9,00 | 27,00 |
| soma_harm | 0 | 0,33 | 0,67 | 1,00 |
| soma_quad | 0 | 9,00 | 18,00 | 27,00 |

Fonte: Autor

2- Quais os valores das médias aritmética, geométrica, harmônica e quadrática impressos na tela do monitor?

Os valores das médias constam no Quadro 11:

**Quadro 11 –** Médias Calculadas pelo Programa

| Médias | item a) | item b) | item c) | item d) | item e) | item f) |
|---|---|---|---|---|---|---|
| Média Quadrática | 3,92 | 8,37 | 2,83 | 10,00 | 5,00 | 3,00 |
| Media Aritmética | 3,33 | 6,00 | 2,40 | 10,00 | 5,00 | 3,00 |
| Média Geométrica | 2,62 | 4,00 | 1,97 | 10,00 | 5,00 | 3,00 |
| Média Harmônica | 2,00 | 3,05 | 1,65 | 10,00 | 5,00 | 3,00 |

Fonte: Autor

3- Qual a relação entre as médias aritmética, geométrica, harmônica e quadrática?

Ao comparar as médias, deve-se concluir que:

Se $x_1$, $x_2$, ..., $x_n$ são números positivos e $M_Q$, $M_A$, $M_G$ e $M_H$ são, respectivamente, suas médias quadrática, aritmética, geométrica e harmônica, então $M_Q \geq M_A \geq M_G \geq M_H$. Além disso, as médias são iguais se, e somente se, $x_1 = x_2 = x_3 = \ldots = x_n$.

## 4.3. ORGANIZAÇÃO E LOCALIZAÇÃO DE UM NÚMERO EM UMA LISTA

Pretende-se que o aluno compreenda as noções de vetor, a motivação para o seu uso e sua sintaxe. Para tal, utilizar-se-á o programa ***Program lista*** para a recepção de *n* números inteiros a serem digitados pelos usuários (Figura 69). Ele armazena os números numa única variável do tipo vetor. Em seguida, a ***procedure ordem_inversa*** será chamada e os números armazenados no vetor serão impressos na tela do monitor na ordem inversa de sua entrada. Após, o programa chamará a ***procedure ordem_crescente*** que ordenará os números em ordem crescente, alterando a sua posição no vetor (parâmetro passado por referência) e os imprimindo na tela do monitor. Por fim, o programa procurará na lista um número a ser digitado pelo usuário. Caso o localize, será impressa na tela do monitor a posição deste número no vetor; caso contrário, será impressa na tela do monitor a não localização do número na lista.

**Figura 69 –** Programa para Organizar uma Lista de Números e Localizar um Número em uma Lista

```
1  Program lista;
2  type
3    vetor = array [1..1000] of integer;
4  var
5    i, n, num: integer;
6    vet: vetor;
7
8  procedure ordem_inversa (vet1: vetor);
9  var
10   k: integer;
11 begin
12   write ('Segue a lista na ordem inversa de sua entrada: ');
13   for k := n downto 1 do
14    write (vet1[k], '  ');
15   writeln;
16 end;
17
18 procedure ordem_crescente (var vet2: vetor);
19 var
20   j, k, aux: integer;
21 begin
22   for j := 1 to n do
23    for k := j + 1 to n do
24      if vet2[j] > vet2[k] then
25      begin
26       aux := vet2[j];
27       vet2[j] := vet2[k];
28       vet2[k] := aux;
29      end;
```

```
30    write ('Lista ordenada: ');
31    for j := 1 to n do
32     write (vet[j], '  ');
33  end;
34
35  procedure procura (num: integer; vet3: vetor);
36  var
37    k: integer;
38    nao_localizado: boolean;
39  begin
40    nao_localizado := true;
41    for k:=1 to n do
42     if vet3[k] = num then
43     begin
44       nao_localizado := false;
45       writeln ('O numero ', num, ' foi localizado. Esta na posicao ', k, ' da lista ordenada.');
46     end;
47    if nao_localizado then
48     writeln ('O numero ', num, ' nao foi localizado na lista.');
49  end;
50
51  begin
52    write ('Informe a quantidade de numeros inteiros a serem digitados: ');
53    readln (n);
54    for i := 1 to n do
55    begin
56     write ('Digite o ', i, '. numero: ');
57     readln (num);
58     vet[i] := num;
59    end;
60    ordem_inversa (vet);
61    ordem_crescente (vet);
62    writeln;
63    write ('Digite o numero a ser localizado: ');
64    readln (num);
65    procura (num, vet);
66  end.
```

Fonte: Autor

**Problema proposto:**

O professor deverá dividir a turma em grupo de 4 alunos. Cada grupo deverá analisar o código do programa ***Program lista*** e responder às seguintes questões:

1- Quais os valores assumidos pela variável indexada *vet* após a inserção de cinco números inteiros, nesta ordem: 6, 3, 8, 2 e 5?

Os valores assumidos pela variável são (Quadro 12):

**Quadro 12 –** Valores assumidos pela variável *vet*

| Variável | vet[1] | vet[2] | vet[3] | vet[4] | vet[5] |
|---|---|---|---|---|---|
| Valores | 6 | 3 | 8 | 2 | 5 |

Fonte: Autor

2- Qual a sequência de valores impressos quando da execução da ***procedure ordem_inversa***?

Sequência: 5 2 8 3 6

3- Após a execução da ***procedure ordem_inversa***, os valores armazenados na variável indexada *vet* foram alterados? Por quê?

Não, uma vez que se trata de parâmetro passado por valor (não leva a palavra ***var*** antes da lista de parâmetros), ou seja, o parâmetro passado por valor recebeu uma cópia do valor da variável usada como argumento na chamada do subprograma. Qualquer manipulação do parâmetro não modificará o valor da variável usada como argumento.

4- Quais são os valores assumidos pelas variáveis *aux*, *vet2[j]* e *vet2[k]* a cada variação ou iteração de *j* e *k* quando da execução da ***procedure ordem_crescente***?

Os valores assumidos pela variável são (Quadro 13):

**Quadro 13 –** Valores assumidos por variáveis

| j | k | vet2 [1] | vet2 [2] | vet2 [3] | vet2 [4] | vet2 [5] |
|---|---|---|---|---|---|---|
| Antes das iterações | | 6 | 3 | 8 | 2 | 5 |
| 1 | 2 | 3 | 6 | 8 | 2 | 5 |
| 1 | 3 | 3 | 6 | 8 | 2 | 5 |
| 1 | 4 | 2 | 6 | 8 | 3 | 5 |
| 1 | 5 | 2 | 6 | 8 | 3 | 5 |
| 2 | 3 | 2 | 6 | 8 | 3 | 5 |
| 2 | 4 | 2 | 3 | 8 | 6 | 5 |
| 2 | 5 | 2 | 3 | 8 | 6 | 5 |
| 3 | 4 | 2 | 3 | 6 | 8 | 5 |
| 3 | 5 | 2 | 3 | 5 | 8 | 6 |
| 4 | 5 | 2 | 3 | 5 | 6 | 8 |

Fonte: Autor

5- Qual a sequência de valores impressos quando da execução da ***procedure ordem_crescente***?

Sequência: 2 3 5 6 8

6- Após a execução da ***procedure ordem_crescente****,* os valores armazenados na variável indexada *vet* foram alterados? Por quê?

Sim. Trata-se de parâmetro passado por referência (leva a palavra ***var*** antes da lista de parâmetros), ou seja, quando o subprograma é chamado, é passado para ele uma referência da variável, sendo possível alterar o conteúdo da variável original usando-se esta referência, o que ocorreu. Assim vet[i] = vet2[i].

7- Ao executar a ***procedure procura***, para verificar se o número 5 faz parte de um dos valores assumidos pela variável indexada *vet,* qual a mensagem que aparecerá na tela do monitor?

*O numero 5 foi localizado. Esta na posicao 3 da lista ordenada.*

## 4.4. INSERÇÃO DE ELEMENTOS EM UMA MATRIZ

Pretende-se que o aluno compreenda as noções de matriz, a motivação para o seu uso e sua sintaxe. Para tal, utilizar-se-á o programa em Pascal ***Program matriz_simetrica*** para a construção de uma matriz simétrica (Figura 70). Como a matriz simétrica é uma matriz quadrada A de ordem n, que satisfaz $A^t = A$, ou seja, os elementos $a_{ij}$ são iguais aos elementos $a_{ji}$, os alunos construirão a matriz A inserindo elementos $a_{ij}$, tais que $i \leq j$.

**Figura 70 –** Programa para Inserir Elementos em uma Matriz

```
 1  program matriz_simetrica;
 2  type
 3    matriz = array [1..50, 1.. 50] of real;
 4  var
 5    mat: matriz;
 6    i, j, n: integer;
 7
 8  begin
 9    write ('Ordem da matriz: ');
10    readln (n);
11    writeln ('Informe os valores dos elementos da matriz');
12
13    for i := 1 to n do
14      for j:= i to n do
15      begin
16        write ('a', i, j, ' = ');
17        readln (mat[i,j]);
18        mat[j,i] := mat[i,j];
19      end;
20
21    writeln ('Matriz A:');
22
23    for i := 1 to n do
24    begin
25      for j:= 1 to n do
26        write (mat[i,j]:8:2);
27      writeln;
28    end;
29
30  end.
```

Fonte: Autor

**Problema proposto:**

O professor deverá dividir a turma em grupo de 4 alunos. Cada grupo deverá analisar o código do programa ***Program matriz_simetrica*** e utilizá-lo para criar a matriz A simétrica de ordem 4, inserindo os seguintes elementos na matriz:

$a_{11} = 2$, $a_{12} = -3$, $a_{13} = 5$, $a_{14} = 1$, $a_{22} = -1$, $a_{23} = 0$, $a_{24} = 0$, $a_{33} = 1$, $a_{34} = 6$ e $a_{44} = -2$. Cada grupo deverá responder às seguintes questões:

1- Quando da execução das instruções de linhas de 13 a 19, o que será impresso na tela do monitor quando i = 3 e j = 4?

Será impresso *a34 = .*

2- Qual número deverá ser informado em seguida pelo usuário do programa?

Deverá ser informado como entrada de dados o número *6*.

3- Em seguida, qual o valor será atribuído ao elemento da matriz? Que elemento é esse?

Será atribuído o valor *6* ao elemento *a43*.

4- Inserindo-se os comandos de linhas 30 a 33 em substituição ao comando de linha 30 do programa ***program matriz_simetrica***, a matriz continuaria sendo simétrica? Como seria representada a nova matriz?

```
30    for i := 1 to n do
31      mat[i,i] = 1;
32
33  end.
```

Sim, pois somente os elementos da diagonal principal seriam alterados.

Representação da matriz:

$$A = \begin{bmatrix} 1{,}00 & -3{,}00 & 5{,}00 & 1{,}00 \\ -3{,}00 & 1{,}00 & 0{,}00 & 0{,}00 \\ 5{,}00 & 0{,}00 & 1{,}00 & 6{,}00 \\ 1{,}00 & 0{,}00 & 6{,}00 & 1{,}00 \end{bmatrix}$$

## 4.5. CRIAÇÃO E MANIPULAÇÃO DE CONJUNTOS

Pretende-se que o aluno compreenda as noções de registros, a motivação para o seu uso e sua sintaxe. Para tal, utilizar-se-á o programa em Pascal ***Program conjunto*** ilustrado na Figura 8, complementado com o código do programa ilustrado na Figura 71. Caberá ao professor mostrar a lógica do programa descrita no tópico 6 do capítulo 1.

**Figura 71 –** Programa para Criação e Manipulação de Conjuntos

```
  1 Program Conjunto;
  2 // área de declarações (Figura 8)
 78
 79 begin
 80   inicializar_conjunto (c);
 81   writeln ('Apos a criacao de um conjunto');
 82   writeln('Conjunto e vazio? ', conjunto_vazio (c), ' ', 'Cardinalidade: ', c.tam);
 83   writeln;
 84
 85   writeln ('Insira n elementos no conjunto: ');
 86   write ('Quantidade a ser inserida: ');
 87   readln (n);
 88
 89   for j := 1 to n do
 90   begin
 91     write (j,'. elemento:' );
 92     readln (elemento);
 93     inserir_no_conjunto (elemento, c);
 94   end;
 95
 96   writeln ('Elementos do conjunto: ');
 97
 98   for j := 1 to c.tam do
 99     write (c.v[j], ' ');
100
101   writeln;
102   writeln('Cardinalidade: ', c.tam);
103   writeln;
104
105   writeln ('Insira um elemento que ja exista no conjunto: ');
106   readln (elemento);
107   inserir_no_conjunto (elemento, c);
108   writeln('Cardinalidade: ', c.tam);
109   writeln;
110
111   if pertence (elemento, c) then
112     writeln ('A cardinalidade não alterou, pois o elemento pertente ao conjunto')
113   else
114     writeln ('A cardinalidade não alterou, pois o elemento nao pertente ao conjunto');
115
116   writeln;
117
118   writeln ('Retire um elemento do conjunto');
```

```
119    write ('Elemento: ');
120    readln (elemento);
121    remover_do_conjunto (elemento, c);
122    writeln ('Elementos do conjunto apos retirada do elemento ‘, elemento, ‘: ‘);
123
124    for j := 1 to c.tam do
125      write (c.v[j], ' ');
126
127     writeln;
128     writeln('Cardinalidade: ', c.tam);
129  end.
```

Fonte: Autor

**Problema proposto:**

O professor deverá dividir a turma em grupo de 4 alunos e, juntamente com eles, responder às perguntas, momento em que explicará o funcionamento do programa ***Program conjunto***.

Para resolver as questões a seguir, utilize o programa ***Program conjunto***.

1- Após a execução da **procedure *inicializar_conjunto*** é criado o conjunto C. Qual é a classificação e cardinalidade deste conjunto [execução das instruções de linhas 80 a 82 do programa principal]?

Conjunto vazio, de cardinalidade igual a zero.

2- Qual a ordem dos elementos do conjunto e a sua cardinalidade após a inserção de 6 elementos de sequência *1, 4, 7, 10, 2, 9* [execução das instruções de linhas 85 a 102 do programa principal]?

Ordem: 1 2 4 7 9 10.
Cardinalidade: 6

3- Após digitar o número *7* como elemento a ser inserido no conjunto, a cardinalidade do conjunto será alterada [execução das instruções de linhas 105 a 114 do programa principal]? Por quê?

Não. O programa verifica se um elemento pertence ao conjunto. Caso pertença, ele não é inserido novamente no conjunto, não alterando, assim, a sua cardinalidade.

4- Após digitar o número *2* como elemento a ser removido do conjunto, qual a nova ordem dos elementos do conjunto e a sua cardinalidade? [execução das instruções de linhas 118 a 128 do programa principal]?

Caso o elemento pertença ao conjunto, ele poderá ser removido de lá. Neste caso, o programa reordenará seus elementos e diminuirá de uma unidade a cardinalidade do conjunto. Logo:

Ordem: 1 4 7 9 10.
Cardinalidade: 5

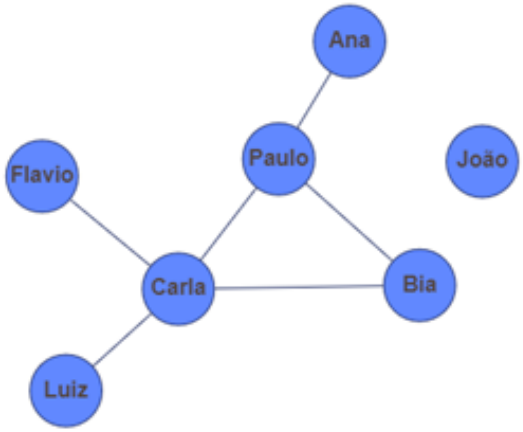
## 4.6. BÁSICO DA TEORIA DOS GRAFOS

O professor deverá apresentar um breve histórico sobre a origem da Teoria dos Grafos e sua importância atual. Em seguida, serão apresentados os conceitos básicos da teoria e exercícios para a sua fixação.

**Problemas propostos:**

1- Seja o grafo G = (V, E) da Figura 72. Os vértices e arestas do grafo representam, respectivamente, os alunos de um curso de espanhol e a relação de amizade entre eles.

**Figura 72 –** Grafo G = (V, E)




Fonte: Autor

Responda:

a) Qual a ordem e o tamanho do grafo?

A ordem (n) de um grafo é o número de vértices que ele possui, enquanto o tamanho (m) é o seu número de arestas ou arcos. Assim, tem-se n = 7 e m = 6.

b) Quem tem o maior número de amigos? O que representa este número na Teoria dos Grafos?

O vértice rotulado como CARLA tem o maior número de ligações ou de vértices adjacentes. Como as ligações representam relações de amizade entre duas pessoas, logo Carla possui o maior número de amigos, que é igual a 4. Este número

representa o grau do vértice, correspondendo ao número de vizinhos ou de vértices adjacentes que ele possui.

c) O que representaria a aresta no grafo complementar $\overline{G}$? Qual o seu tamanho?

Dado um grafo G, o grafo complementar $\overline{G}$ é um grafo que contém as ligações (arestas ou arcos) que não estão no grafo G em relação a um universo dado. Se as ligações em G representam as relações de amizade entre dois alunos, a sua falta representa a ausência dessas relações. Esta ausência em G estará representada em $\overline{G}$ por uma aresta ligando dois de seus vértices.

Esse universo, em grafos não orientados, como o da Figura 72, é o conjunto de todas as arestas possíveis em um grafo G1 de mesma ordem que G, chamado de grafo completo. Sendo n a ordem de G, então o conjunto de arestas do grafo completo G1 será:

$$C_2^n = \frac{n \,.\, (n-1)}{2}$$

Se o tamanho de G é *m*, então $\overline{G}$ terá o tamanho $m_1 = C_2^n - m$. Logo:

$$m_1 = \frac{n \,.\, (n-1)}{2} - m$$

Então: $m_1 = \frac{7 \,.\, 6}{2} - 6 = 15$

2- Considere a matriz de valores A de um grafo, que possui 6 vértices rotulados por números que variam de 1 a 6. O valor do elemento $a_{ij}$ da matriz, onde i e j representam os vértices do grafo, é igual ao custo da aresta ou arco w (i, j). Assim, de acordo com a figura, o custo w (3, 4) é igual a 400.

A =

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 200 | 100 | 0 | 0 | 0 |
| 2 | 150 | 0 | 250 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 400 | 500 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 100 |
| 5 | 0 | 0 | 600 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 500 | 0 |

Faça o que se pede:

a) O grafo é orientado ou não orientado? Por que motivo?

As matrizes de valores e a de adjacência de um grafo não orientado, diferentemente de um grafo orientado, são matrizes simétricas. Como a matriz de valores A não corresponde a uma matriz simétrica, esta matriz representa um grafo orientado.

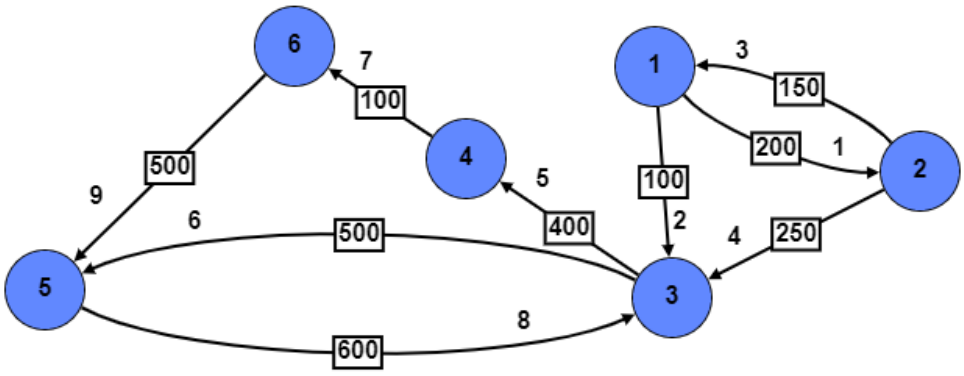
b) Represente o grafo em um diagrama.

Os elementos aij de uma matriz de valores, que representa um grafo orientado, assumirão o custo dos arcos que ligam os vértices i ao j, na direção de i para j. Não havendo ligação entre os vértices i e j, aij assumirá o valor igual a zero.

Os elementos da matriz diferentes de zero são $a_{12}$, $a_{13}$, $a_{21}$, $a_{23}$, $a_{34}$, $a_{35}$, $a_{46}$, $a_{53}$ e $a_{65}$, cujos valores representam, respectivamente, os custos dos arcos (1, 2), (1, 3), (2, 1), (2, 3), (3, 4), (3, 5), (4,6), (5,3) e (6, 5).

Assim, a matriz de valores A pode representar o grafo D = (V, A) da Figura 73.

**Figura 73 –** Grafo D = (V, A) representado pela Matriz de Valores




Fonte: Autor

c) Represente a matriz de adjacência do grafo.

Assim como a matriz de valores, a matriz de adjacência também é uma matriz cujas linhas e colunas são formadas pelos n vértices do grafo, tratando-se, portanto, de uma matriz quadrada de ordem n.

Sendo B = $[b_{ij}]$ a matriz de adjacência que representa o grafo orientado D = (V, A), o valor de $b_{ij}$ dependerá da existência ou não de uma ligação entre os vértices i e j e da orientação em que tais vértices estão ligados. Por convenção, o elemento $b_{ij}$ da matriz B receberá:

- 1, se os vértices i e j forem adjacentes, com orientação de i para j
- 0, nos demais casos.

Assim, o grafo D = (V, A), que é representado pela matriz de valores dada, é representado pela seguinte matriz de adjacência:

$$B = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$

d) Qual é a soma dos valores da linha 3 da matriz de adjacência? O que este número representa na Teoria dos Grafos?

$L_3 = b_{31} + b_{32} + b_{33} + b_{34} + b_{35} + b_{36} = 0 + 0 + 0 + 1 + 1 + 0 = 2.$

Este número representa o semigrau exterior $d^+(3)$, que é o número de sucessores do vértice 3.

e) Qual é a soma dos valores da coluna 3 da matriz de adjacência? O que este número representa na Teoria dos Grafos?

C3 = b13 + b23 + b33 + b43 + b53 + b63 = 1 + 1 + 0 + 0 + 1 + 0 = 3.

Este número representa o semigrau interior d-(3), que é o número de antecessores do vértice 3.

f) Qual é o grau do vértice 3?

Como o grafo é orientado, o grau de um vértice é a soma dos seus

subgraus. Assim, o grau do vértice 3 é: $d(3) = d^+(3) + d^-(3) = 2 + 3 = 5$ (soma dos valores de $L_3$ com $C_3$).

g) Qual é a relação entre o somatório dos valores de todas as linhas e colunas da matriz de adjacência e o tamanho do grafo representando por esta matriz?

O somatório dos valores de todas as linhas (soma dos semigraus exteriores) e de todas as colunas (soma dos semigraus interiores) da matriz de adjacência de um grafo orientado ou não orientado representa a soma dos graus de todos os vértices do grafo. Esta soma é o dobro do número de arestas ou arcos do grafo, ou seja, é o dobro do tamanho do grafo, conforme Teorema 1**.**

Assim $\sum L + \sum C = 2\,m \rightarrow 9 + 9 = 2\,m \rightarrow m = \frac{18}{2} = 9$.

h) Represente a Matriz de incidência do grafo?

A matriz de incidência é uma matriz n x m formada por n linhas que correspondem a cada um dos vértices do grafo, e por m colunas que correspondem a cada uma das ligações entre os vértices.

Por convenção, o elemento $c_{ij}$ da matriz de incidência $C = [c_{ij}]_{nxm}$ será, no caso de grafo orientado, igual a:

- +1, quando o vértice i for de saída em relação ao arco j
- -1, quando o vértice i for de entrada em relação ao arco j
- 0, caso contrário aos anteriores

Assim, o grafo D = (V, A), que é representado pela matriz de valores dada, também pode ser representado pela matriz de incidência:

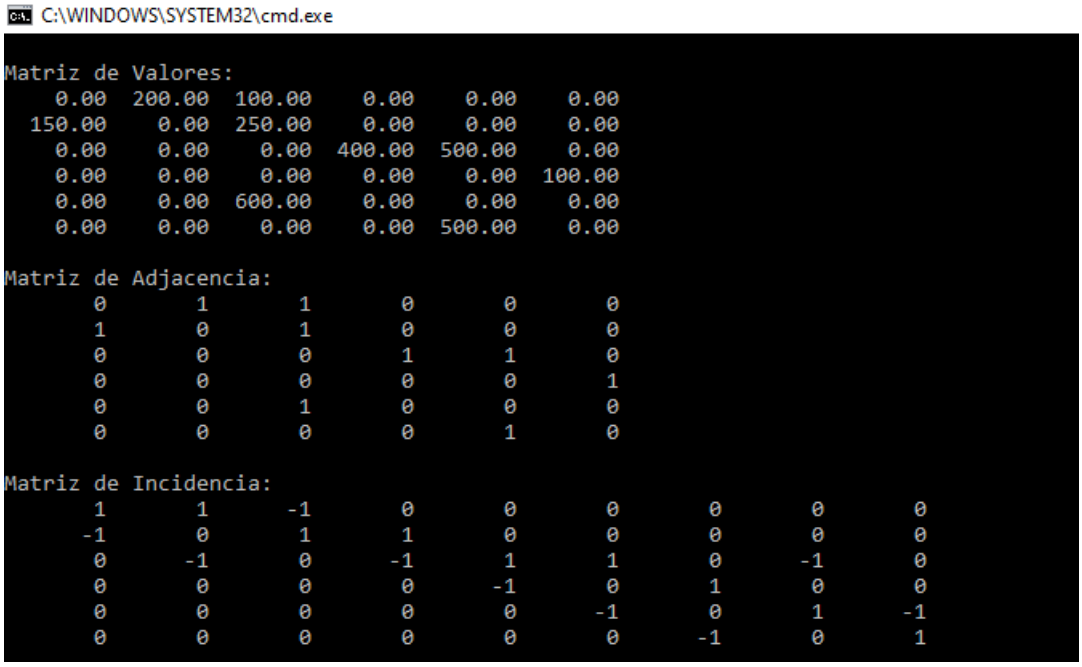
C =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | -1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | -1 | -1 | 1 | 1 | -1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 0 | -1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 |

i) Utilize o programa ***Program Representacao_de_Grafo***, ilustrado na Figura 44 para comparar as matrizes encontradas nos itens “c” e “h” com as geradas pelo programa.

Matrizes geradas pelo programa (Figura 74):

**Figura 74 –** Matrizes Impressas pelo Programa




Fonte: Autor

3- Uma cidade possui 9 bairros, identificados pelos números de 1 a 9. Como a cidade é cortada por um rio, muitas vezes o acesso a um determinado bairro dar-se-á pelo uso de uma embarcação. Curiosamente, só há estradas que ligam dois bairros, quando a soma entre os números que identificam tais bairros é um número múltiplo de 3. Por exemplo, existem estradas que ligam os bairros nºs 2 e 4 (soma igual a 6) ou nºs 1 e 2 (soma igual a 3).

Faça o que se pede:

a) Existe a possibilidade de você, motorista, sair com seu automóvel do bairro nº 4 e chegar no bairro nº 6?

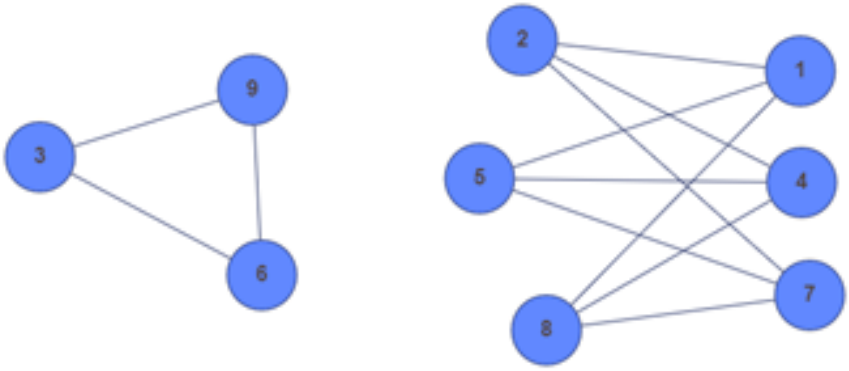
A situação descrita pode ser representada por um grafo não-dirigido G = (V, E), onde V é o conjunto dos bairros da cidade e E é o conjunto das estradas que ligam dois bairros sob determinada condição.

Observe que:

- Os vértices 1, 4 e 7 podem se ligar aos vértices 2, 5 e 8 e vice-versa;
- Os vértices 1, 4 e 7 não podem se ligar entre si;
- Os vértices 2, 5 e 8 não podem se ligar entre si;
- Os vértices 3, 6 e 9 se ligam entre si.

Então, o grafo pode estar assim representado (Figura 75):

**Figura 75 –** Grafo da Cidade




Fonte: Autor

De acordo com o grafo, não existe a possibilidade de você, motorista, sair com seu automóvel do bairro nº 4 e chegar ao bairro nº 6.

b) Caso você utilizasse um grafo para modelar a situação descrita, este grafo seria conexo ou não conexo? Justifique.

Diz-se que um grafo não orientado G = (V, E) é conexo se existe pelo menos um caminho entre cada par de vértices do grafo. No caso de não existir um caminho que ligue pelo menos dois vértices do grafo, diz-se que o grafo é não conexo. Com base na resposta do item “a”, o grafo é não conexo.

c) Em caso de grafo não conexo, onde poderia ser adicionada uma aresta para o grafo se tornar conexo? Qual o nome especial dado a esta aresta?

Para que o grafo se torne conexo, pode-se adicionar uma aresta ligando um dos vértices 3, 6 ou 9 a um dos demais vértices. Por exemplo, ligando o vértice 6 ao vértice 5.

O nome desta aresta é ponte, caracterizando-se como indispensável para a conexão de um grafo.

d) Cite quatro percursos que um motorista poderia tomar para sair do bairro nº 1 e chegar no bairro nº 4.

Para se chegar ao bairro nº 4 partindo do bairro nº 1, pode-se tomar os percursos 1 – 2 – 4, 1 – 5 – 4, 1 – 8 – 4, 1 – 2 – 7 – 8 – 4.

## 4.7. ALGORITMO DE DIJKSTRA

Inicialmente, o professor deverá abordar o conceito de caminho mínimo, os casos em que são aplicados, os algoritmos utilizados para determiná-lo, dentre eles o Algoritmo de Dijkstra e o de Bellman-Ford, ressaltando a diferença entre eles.

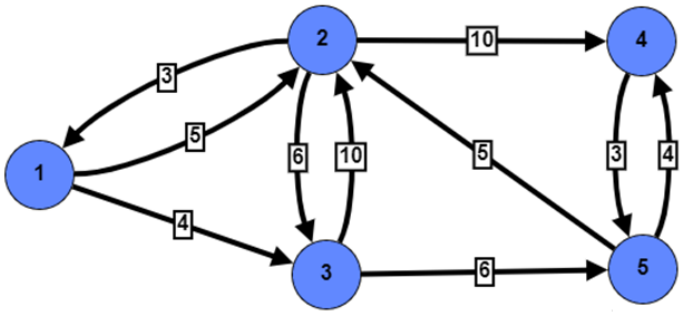
Propõe-se que o professor divida a turma em grupo de 4 alunos. Em seguida apresente o problema a turma e o ***Algoritmo de Dijkstra***, ilustrado na Figura 46, utilizado para solucioná-lo. O professor passará a explicar genericamente o funcionamento do algoritmo. Caberá ao grupo analisar mais profundamente o algoritmo e encontrar a solução para o problema. Em seguida, o professor corrigirá o exercício, procurando sanar as dúvidas dos alunos. Por fim, apresentará o **procedimento *Escreva_caminho***, ilustrado na Figura 54, que escreve na tela do monitor o caminho mínimo a ser percorrido de um vértice de origem a um vértice de destino, e o programa em Pascal desenvolvido a partir de ambos os algoritmos, descrito no Anexo A, fazendo uma correlação entre os códigos.

**Problema proposto:**

1- Um prestador de serviços da área de tecnologia da informação (TI), sempre sendo contratado para a realização de serviços altamente técnicos por empresas localizadas em cidades do interior de um pequeno estado brasileiro, observou que a demanda por tais serviços era alta, todavia a sua oferta era inexistente. Técnicos de fora da cidade eram contratados de forma reiterada, o que encarecia o valor da prestação de serviços.

O mapa da região está representado pelo grafo direcionado D = (V, A) da figura.

**Figura 76** – Problema envolvendo o Algoritmo de Dijkstra




Fonte: Autor

Cada vértice v ∈ V do grafo representa uma cidade do interior, e cada arco a ∈ A representa uma via que liga as cidades, com um custo $c_a$ a ele associado (distância em Km entre as cidades). A direção dos arcos corresponde ao sentido das vias.

O analista de TI vislumbrou uma oportunidade de negócio e resolveu se instalar na cidade 1, próxima das demais, opção que se deu em face do seu baixo custo de vida.

Supondo que você fosse o analista de TI, que atendesse diariamente um único cliente, quais percursos tomaria para chegar aos clientes localizados nas cidades 2, 3, 4 e 5, percorrendo a menor distância possível?

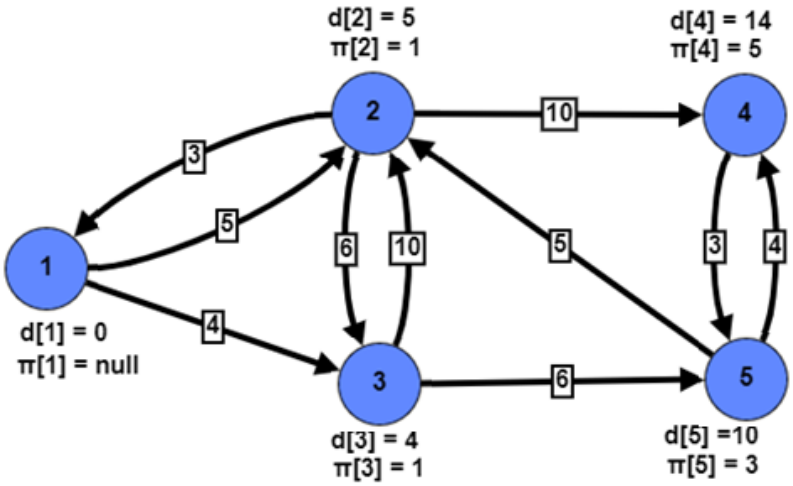
**Solução:**

Todos os passos para a solução do problema estão descritos no capítulo 3.1. Quando do encerramento do algoritmo, tem-se:

| ***v*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** |
|---|---|---|---|---|---|
| ***d*** | *0* | *5* | *4* | *14* | *10* |
| ***π*** | *null* | *1* | *1* | *5* | *3* |
| *S = {1, 3, 2, 5, 4} e Q = Ø* | | | | | |

A Figura 77 mostra o grafo que corresponde as menores distâncias ligando a cidade 1 as demais.

**Figura 77 –** Solução do Problema envolvendo o Algoritmo de Dijkstra




Fonte: Autor

Assim, partindo da cidade 1, tomar-se-iam os seguintes caminhos para se chegar as demais cidades:

- Cidade 2 ∴ caminho: (1, 2) distância percorrida: 5 Km
- Cidade 3 ∴ caminho: (1, 3) distância percorrida: 4 Km
- Cidade 4 ∴ caminho: (1, 3), (3, 5), (5, 4) distância percorrida: 14 Km
- Cidade 5 ∴ caminho: (1, 3), (3, 5) distância percorrida: 10 Km

## 4.8. ALGORITMO DE BELLMAN-FORD

Propõe-se que o professor divida a turma em grupo de 4 alunos. Em seguida, apresente o problema a turma e o ***Algoritmo de Bellman-Ford***, ilustrado na Figura 57, utilizado para solucioná-lo. O professor passará a explicar genericamente o funcionamento do algoritmo. Caberá ao grupo analisar mais profundamente o algoritmo e encontrar a solução para o problema. Em seguida, o professor corrigirá o exercício, procurando sanar as dúvidas dos alunos. Logo depois, reapresentará o **procedimento *Escreva_caminho***, e, por fim, apresentará o programa em Pascal desenvolvido a partir de ambos os algoritmos, descrito no Anexo B, fazendo uma correlação entre os códigos.

**Problema proposto:**

1- Uma empresa nacional, que trabalha com importação de mercadorias, diariamente compra moeda estrangeira para liquidar as operações mercantis realizadas em diversos países do globo terrestre.

   Por exemplo, ao adquirir mercadorias do Canadá liquidará a operação com o dólar canadense. Caso a mercadoria seja adquirida da Inglaterra, a liquidação será feita com libras esterlinas.

   A empresa possui um setor responsável pela procura da melhor cotação na liquidação das operações. Supondo que tenha que liquidar uma operação de C$ 1.000,00 (mil dólares canadenses), o setor verificaria quais as opções disponíveis no mercado para a conversão do real para dólar canadense, a fim de desembolsar menos reais para liquidar a operação.

   O Quadro 14 traz a cotação entre as moedas.

**Quadro 14 –** Cotação entre Moedas

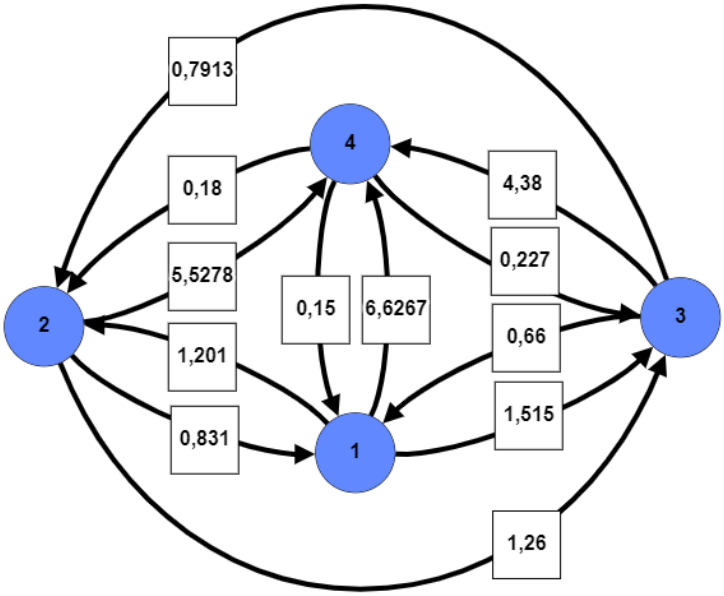
| Moeda | R$ | US$ | £ | C$ |
|---|---|---|---|---|
| R$ | 1,0000 | 0,1800 | 0,1500 | 0,2270 |
| US$ | 5,5278 | 1,0000 | 0,8310 | 1,2600 |
| £ | 6,6267 | 1,2010 | 1,0000 | 1,5150 |
| C$ | 4,3800 | 0,7913 | 0,6600 | 1,0000 |

Fonte: Autor

Dadas as moedas e as taxas de câmbio, que decisão a importadora deverá tomar?

O grafo da Figura 78 modela a situação, em que os vértices representam as moedas (1: £, 2: US$, 3: C$ e 4: R$); e os arcos, as taxas de câmbio. Deixou-se de representar os laços.

**Figura 78 –** Grafo Representando a Relação entre Moedas




Fonte: Autor

Vejamos apenas três opções dentre outras para a liquidação da operação:

1ª opção: Com R$ 1.000,00 compra-se C$ 227,00 (1.000,00 x 0,2270). Logo a empresa precisará de R$ 4.405,29 para comprar C$ 1.000,00.

2ª opção: Com R$ 1.000,00 compra-se US$ 180,00, que comprará C$ 226,80 (1.000,00 x 0,1800 x 1,2600). Assim, a empresa precisará de R$ 4.409,17 para comprar C$ 1.000,00.

3ª opção: Com R$ 1.000,00 compra-se £ 150,00, que comprará C$ 227,25 (1.000,00 x 0,1500 x 1,5150). Assim, a empresa precisará de R$ 4.400,44 para comprar C$ 1.000,00.

Assim, a melhor opção para liquidar a operação de importação seria a terceira.

Observe que, para calcularmos o câmbio envolvendo as moedas (moeda de origem – moeda final), multiplicamos as taxas de câmbio envolvidas nas operações, representadas pelos custos dos arcos. Vê-se que quanto maior a taxa de câmbio final menos reais serão dispendidos para quitar a obrigação da importadora.

O exemplo dado aqui envolve apenas quatro moedas. Existem cinco possibilidades de conversão de real para dólar canadense:

- R$ x C$
- R$ x US$ x C$
- R$ x £ x C$
- R$ x US$ x £ x C$
- R$ x £ x US$ x C$

Poderia haver outras conversões, todavia são normalmente desvantajosas, com prejuízos na operação, como por exemplo a conversão R$ => US$ => R$ => 3, em que há uma compra de dólar americano e uma recompra de reais.

Nas transações envolvendo seis moedas, o número de permutações chega a 61.

Imagine casos reais que envolvam um número considerável de moedas. Ficaria impraticável um programa processar as informações e encontrar uma solução plausível em um tempo razoável.

Outra forma de encontrar a solução seria buscar um caminho mínimo no grafo. Todavia, a lógica do caminho mínimo é somar os custos dos arcos e não os multiplicar. Em outras palavras, para calcularmos o caminho mínimo entre um vértice de origem (moeda em real) e os demais vértices (demais moedas) do grafo, necessitaríamos alterar os custos dos arcos, a fim de poder somá-los. E qual seria a lógica dessa alteração?

Sabemos que dados dois números a e b, se a > b > 0 então log a > log b. Multiplicando ambos os lados da inequação por (-1) teremos -log a < - log b. Assim, se a e b representam a taxa de câmbio final, a melhor opção de liquidação será aquela de maior cotação *a* (valor máximo) ou -log a (valor mínimo).

Nessa situação, a é o produto de diversas taxas de câmbio, ou seja:

$a = i_1 * i_2 * ... * i_n$

Então $\log a = \log (i_1 * i_2 * ... * i_n)$

Logo $-\log a = -\log (i_1 * i_2 * ... * i_n) = -(\log i_1 + \log i_2 + ... + \log i_n) = -\log i_1 - \log i_2 - ... - \log i_n$.

Assim, quanto menor o valor de -log a, ou seja, quanto menor a soma $-\log i_1 - \log i_2 - ... - \log i_n$, menos recursos serão dispendidos para a liquidação da obrigação. Então, se ponderarmos os arcos do grafo pelo valor negativo do logaritmo da taxa de câmbio, o caminho mínimo de um vértice aos demais terá como custo total a soma dos arcos envolvidos no percurso.

Vamos agora alterar o valor dos arcos do grafo da Figura 78.

O Quadro 15 traz o valor negativo do logaritmo decimal da taxa de câmbio apresentada no Quadro 14 (-log a, onde a é a taxa de câmbio).

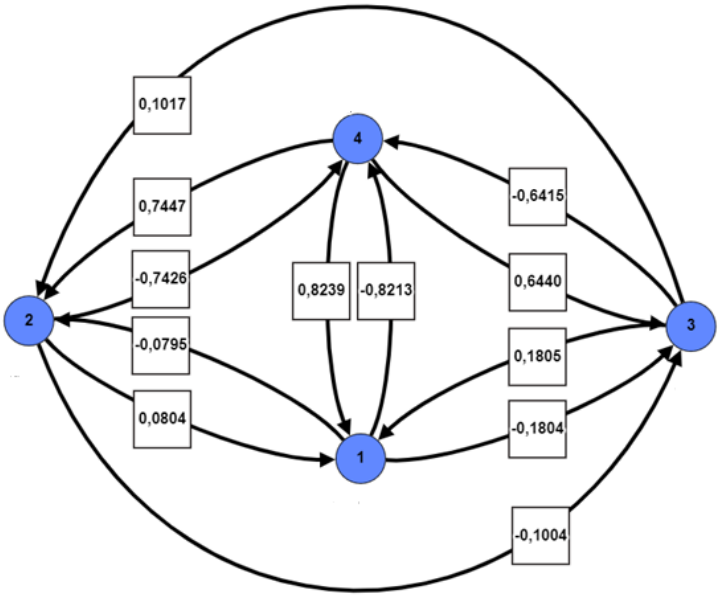
**Quadro 15 –** Cotação Logarítmica entre Moedas

| Moeda | R$ | US$ | £ | C$ |
|---|---|---|---|---|
| R$ | 0,0000 | 0,7447 | 0,8239 | 0,6440 |
| US$ | -0,7426 | 0,0000 | 0,0804 | -0,1004 |
| £ | -0,8213 | -0,0795 | 0,0000 | -0,1804 |
| C$ | -0,6415 | 0,1017 | 0,1805 | 0,0000 |

Fonte: Autor

O grafo da Figura 79 modela a nova situação:

**Figura 79 –** Problema envolvendo o Algoritmo de Belmman-Ford



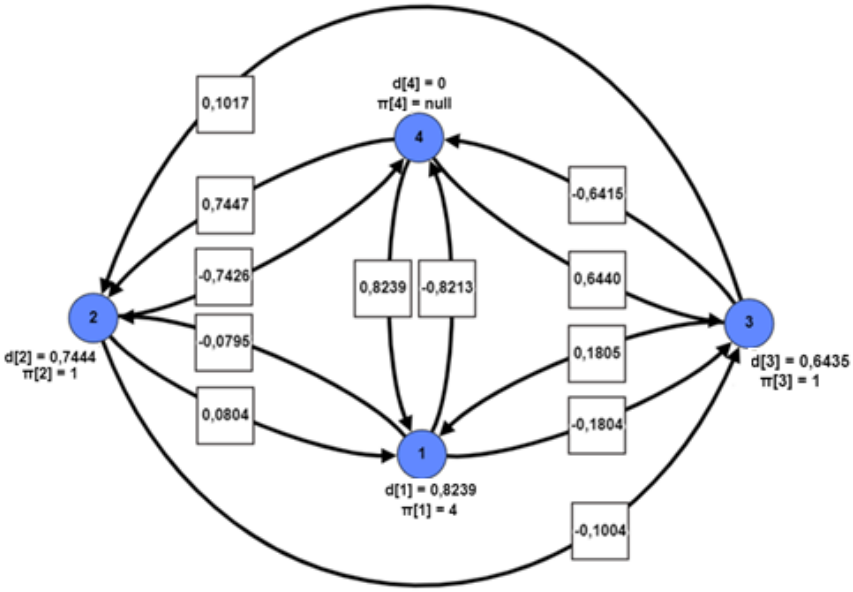

Fonte: Autor

Assim, pergunta-se: Qual o menor caminho do vértice 4 ao vértice 3?

**Solução:**

Todos os passos para a solução do problema estão descritos no capítulo 3.2. Quando do encerramento do algoritmo, tem-se como solução o grafo da Figura 80.

**Figura 80 –** Solução do Problema Envolvendo o Algoritmo de Belmman-Ford




Fonte: Autor

Assim, o menor caminho que leva o vértice de origem 4 ao vértice 3 será (4, 1), (1, 3), com custo total de 0,6435. Para calcularmos a taxa de câmbio total (x) envolvida nas operações, temos:

$$- (\log x) = d[3]$$

$$\text{Então } \log x = - d[3]$$

$$\text{Logo } x = 10^{-d[3]} = 10^{-0,6435}$$

$$\text{Então } x = 0,22725$$

Portanto, 0,22725 é a melhor cotação para liquidar operações envolvendo o real e o dólar canadense. Como a empresa possuía uma obrigação de C$ 1.000,00, para liquidá-la, terá que desembolsar R$ 4.400,48 [1.000,00 / 0,2275], que corresponde a 3ª opção descrita no enunciado do problema.

## 5. CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi elaborar uma proposta de ensino para as turmas de ensino médio abordando um tópico específico da Teoria dos Grafos que trata de caminhos mínimos, dos Algoritmos de Dijkstra e Bellman-Ford utilizados para determiná-los e dos programas em Pascal a eles relacionados, uma vez que o tema “Grafo” tem sua relevância na solução dos problemas que envolvem o nosso cotidiano. Inicialmente, mostrou-se tal importância ao citar trabalhos acadêmicos sobre a teoria e, mais especificamente, ao abordarmos Caminhos Mínimos em Grafos.

Em seguida, mostrou-se a viabilidade de a Teoria dos Grafos, mais especificamente caminhos mínimos e seus algoritmos, serem trabalhados nos itinerários formativos de Matemática e suas Tecnologias. Da Mesma forma, mostrou-se a viabilidade da linguagem de programação Pascal poder ser ofertada pelas escolas através dos itinerários formativos de Matemática e suas Tecnologias.

Procurando conciliar teoria e prática, foram propostos problemas envolvendo o cotidiano dos alunos, para serem resolvidos em grupo, pretendendo-se, ao final da etapa, o domínio dos conceitos básicos do Pascal e da Teoria dos Grafos, o entendimento dos algoritmos e dos programas em pascal relativos a caminhos mínimos com o fim de aplicá-los a casos práticos.

É bom ressaltar a dificuldade em encontrar em livros, aulas, trabalhos acadêmicos etc. um único exemplo prático relacionado ao Algoritmo de Bellman-Ford.

Espera-se, com o tema proposto, que os alunos tenham a curiosidade de conhecer mais a Teoria dos Grafos e possam utilizar o Pascal para criar seus próprios programas no auxílio de suas tarefas diárias, como na resolução dos problemas envolvendo matemática e física. Isso já garantiria o previsto na LDB: uma mudança comportamental do aluno na busca pelo novo.

Concluída esta etapa, pretende-se em trabalhos futuros abordar temas ligados à Teoria dos Grafos tão relevantes como o aqui abordado como Árvore Geradora Mínima e Problemas de Coloração. Há inúmeras aplicações práticas do

nosso cotidiano envolvendo os assuntos como o planejamento de rotas de voos por uma companhia aérea (FRANCO, 2019) e o controle de tráfego viário de uma cidade (HERNANDES, 2007).

## 6. REFERÊNCIA BIBLIOGRÁFICA


ALMEIDA, Angela Maria de Oliveira; CUNHA, Gleicimar Gonçalves. Representações sociais do desenvolvimento humano. **Psicologia: reflexão e crítica**, 2003. Disponível em: https://www.scielo.br/j/prc/a/rHJrvCntshLb7WSN3GVCz8n/?lang=pt&format=pdf. Acesso em 28 dez. 2021.

BOAVENTURA NETTO, Paulo Oswaldo. **Grafos**: Teoria, Modelos, Algoritmos. ISBN 978-85-212-0680-4. 5. Ed. São Paulo: Blucher, 2011.

BOAVENTURA NETTO, Paulo Oswaldo; JURKIEWICS, Samuel. **Grafos**: Introdução e Prática. ISBN 978-85-212-0473-2. São Paulo: Blucher, 2009.

BOYER, Carl B. **História da Matemática**. Tradução: Elza F. Gomide. São Paulo: Blucher, 1974. Disponível em: https://www.docsity.com/pt/boyer-carl-b-historia-da-matematica/4870774/. Acesso em: 01 mai. 2021.

CASTILHO, Marcos *et al*. **Algoritmos e Estrutura de Dados 1**. ISBN: 978-65-86233-62-9. Curitiba, 2020. Disponível em: https://www.inf.ufpr.br/marcos/livro_alg1/livro_alg1.pdf. Acesso em: 03 jul. 2021.

CASTILHO, Marcos *et al*. **Guia rápido de referência da linguagem Pascal**: Versão Free Pascal. 2009. Disponível em: https://www.inf.ufpr.br/cursos/ci055/pascal.pdf. Acesso em 31 jul. 2021.

CASTRO JUNIOR, Amaury Antonio. **Implementação e avaliação de algoritmos BSP/CGM para o fecho transitivo e problemas relacionados**. 2003. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Mato Grosso do Sul, Campo Grande, 2003. Disponível em: https://repositorio.ufms.br/bitstream/123456789/445/1/Amaury%20Antonio%20de%20Castro%20Junior.pdf. Acesso em: 21 dez. 2021.

DA SILVEIRA JUNIOR, Carlos Roberto et al. A identificação de conflitos em sala de aula utilizando Visualização de Informações. In: **Anais do XXXII Simpósio Brasileiro de Informática na Educação**. SBC, 2021. Disponível em: https://sol.sbc.org.br/index.php/sbie/article/view/18091/17925. Acesso em: 28 dez. 2021.

FEOFILOFF, Paulo. **Caminhos e ciclos em grafos**. 2008. Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/paths-and-cycles.html. Acesso em: 07 mai. 2021.

FRANCO, Lucas dos Santos. **Um método de planejamento de rotas de voo de vant multirotor para cobertura de áreas utilizando a meta-heurística ACO**. 2019. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos, São Carlos, 2019. Disponível em:

https://repositorio.ufscar.br/bitstream/handle/ufscar/12177/Dissertacao-Mestrado_Lucas-Franco.pdf?sequence=4&isAllowed=y. Acesso em: 20 dez. 2021.

HERNANDES, Fábio. O problema de coloração em grafos Fuzzy, **XXXIX SBPO**, Fortaleza, 2007. Disponível em http://www.din.uem.br/sbpo/sbpo2007/pdf/arq0249.pdf. Acesso em: 20 dez. 2021.

LOUREIRO, Antonio Alfredo Ferreira; GOUSSEVSKAIA, O. **Grafos**. 2015. Disponıvel em: http://homepages. dcc. ufmg. br/loureiro/md/md Grafos. pdf. Acesso em: 28 dez. 2021.

MANZANO, José Augusto N. G.; YAMATUMI, Wilson Y. **Programando em Turbo Pascal 7.0 & Free Pascal Compiler**: Guia Prático de Orientação e Desenvolvimento. 8. Ed. São Paulo: Erica, 2002.

MATHIAS, Ivo Mario, **Algoritmos e Programação I**. ISBN 978.85.8024.298.0. Ponta Grossa: UEPEG/ NUTEAD, 2017. Disponível em: https://educapes.capes.gov.br/bitstream/capes/176223/2/Algoritmos%20e%20Programa%C3%A7%C3%A3o%20I%20EBOOK.pdf. Acesso em: 29 jul. 2021.

MORENO, Eduardo; RAMÍREZ, Héctor. **Grafos: Fundamentos y Algoritmos**. ISBN 978-956-306-076-8. 1. ed. Santago: J. C. Saez, 2011.

MOTTA, Cézanne Alves Mendes; BRITO, George Lauro Ribeiro de. Modelagem e otimização de fluxo em uma rede real conectada, **Revista Desafios**, Palmas, 2017. Disponível em https://sistemas.uft.edu.br/periodicos/index.php/desafios/article/view/3248/9522. Acesso em: 20 dez. 2021.

NOGUEIRA JÚNIOR, Dárcio Costa. **Grafos e Problemas de Caminhos Mínimos**. 2017. Dissertação (Mestrado em Matemática) – Universidade Federal de Viçosa, Viçosa, 2017. Disponível em: https://www.locus.ufv.br/bitstream/123456789/11869/1/texto%20completo.pdf. Acesso em: 12 jan. 2021.

OLIVEIRA, Alicia Cavasso de *et al*. Aplicação do conceito de caminho mínimo em uma empresa de pequeno porte através do Algoritmo de Dijkstra, **XI FATECLOG**, Campinas, 2020. Disponível em https://fateclog.com.br/anais/2020/APLICA%C3%87%C3%83O%20DO%20CONCEITO%20DE%20CAMINHO%20M%C3%8DNIMO%20EM%20UMA%20EMPRESA%20DE%20PEQUENO%20PORTE%20ATRAV%C3%89S%20DO%20ALGORITMO%20DE%20DIJKSTRA.pdf. Acesso em: 20 dez. 2021.

PEREIRA, Silvio do Lago. **Linguagem Pascal**: Noções básicas usando Turbo Pascal, 2018. Disponível em https://www.ime.usp.br/~slago/slago-pascal.pdf. Acesso em: 20 dez. 2021.

RESE, Alex Luciano Roesler *et al.* Análise de Algoritmos da Árvore Geradora Mínima para o Problema de Reconfiguração de Redes de Distribuição. **Revista de Informática Aplicada**, 2017. Disponível em: https://seer.uscs.edu.br/index.php/revista_informatica_aplicada/article/view/6922/3013. Acesso em: 28 dez. 2021.

SANTOS, J. C.; MOTA, B. **História da matemática**: Teoria de grafos. Faculdade de Ciências do Porto, Porto, 2010. Disponível em: https://www.academia.edu/6342431/Teoria_de_Grafos_-_História. Acesso em: 10 jun. 2021.

SILVA, Anderson Alves da. **Uma abordagem heurística para o problema do carteiro chinês capacitado na coleta de lixo urbano**. 2020. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Pernambuco, Recife, 2020. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/39252/1/DISSERTA%c3%87%c3%83O%20Andersson%20Alves%20da%20Silva.pdf. Acesso em: 20 dez. 2021.

SOUZA, Audemir Lima de. **Teoria dos Grafos e Aplicações**. 2013. Dissertação (Mestrado em Matemática) – Universidade Federal do Amazonas, Manaus, 2013. Disponível em: https://tede.ufam.edu.br/bitstream/tede/4788/2/Disserta%C3%A7%C3%A3o%20-%20Audemir%20Lima%20de%20Souza.pdf. Acesso em: 12 jan. 2021.

TEIXEIRA, Lilian Aparecida *et al*. **Metodologia do Ensino da Matemática**. ISBN 978-85-8482-907-1. Londrina: Editora e Distribuidora Educacional S.A., 2017.

# APÊNDICE A – VERSÃO EM PASCAL DO ALGORITMO DE DIJKSTRA

```
1  Program Dijkstra;
2  uses
3     Crt; // A linha Uses Crt informa ao compilador que ele deve incluir a
       biblioteca Crt no seu programa
4          // Vai habilitar funções como gotoxy e clrscr
5  Const
6     MAX = 100;
7
8  Type
9     matriz = array [0..MAX, 0..MAX] of real;
10    matriz1 = array [0..MAX, 0..MAX] of integer;
11    vetor = array [0..MAX] of real;
12    vetor1 = array [0..MAX] of integer;
13
14    conjunto = record
15       tam: integer;
16       vet: array [ 0..MAX+1] of integer;
17    end;
18
19 var
20    S,Q: conjunto;
21    w: matriz;
22    mat_adj: matriz1;
23    d: vetor;
24    pai, vet_adj: vetor1;
25    lin, col, i, j, p, t, u, v, y, resp, ordem, tam, vert: integer;
26    custo: real;
27
28 procedure tela_inicial;
29 // escreve na tela do monitor informações sobre o programa e recomendações ao
   usuário
30 begin
31    Writeln
      ('=================================================================================
      ==================================');
32    Writeln ('   Este programa eh utilizado para encontrar o caminho minimo de um
      vertice de origem dado aos demais vertices ');
33    writeln ('   do grafo, utilizando o Algoritmo de Dijkstra.');
34    writeln;
35    Writeln ('   Durante a execucao do programa, serao  solicitados a  ordem e o
      tamanho do  grafo, o conjunto de  arcos e seus');
36    writeln ('   custos e o vertice de origem.');
37    Writeln
      ('=================================================================================
      ==================================');
38    writeln;
39 end;
40
41 procedure ler_grafo (var j: integer);
42 // procedimento para a leitura dos arcos ponderados do grafo (vértices de origem e
   destino e custo)
43 // gotoxy (x, y): representa a coordenada (x, y) do cursor na tela do monitor
44 begin
45    gotoxy (4,j);
46    write ('Vi: ');
47    read (lin);
48    gotoxy(18, j);
49    write ('Vj: ');
50    read (col);
51    gotoxy(34, j);
52    write ('w (',lin,', ',col, '): ');
53    read (custo);
54 end;
55
56 Procedure Criar_grafo_ponderado;
```

```
57  // Procedimento para criar o grafo ponderado. O usuário informará a ordem e o
    tamanho do grafo. Em seguida, o procedimento chamará
58  // o procedimento ler_grafo. Após a leitura dos dados, testes serão feitos para
    verificar se o vértice de origem e/ou destino fazem parte
59  // do grafo, no caso as variáveis lin e col terão que ser menores ou iguais a
    ordem do grafo. Os dados serão armazenados em uma matriz
60  // de valores, a qual representará o grafo ponderado
61  label
62     retorno; // permite o desvio do fluxo do programa ao ser chamado
63
64  var
65     k: integer;
66
67  begin
68     // Informação da ordem e do tamanho do grafo
69     writeln ('   Digite a ordem do grafo');
70     write ('   ordem: ');
71     readln (ordem);
72     writeln;
73     writeln ('   Digite o tamanho do grafo');
74     write ('   tamanho: ');
75     readln (tam);
76     Writeln
       ('=================================================================================
       ==================================');
77     writeln;
78
79     j:= 17;
80     writeln ('   Informe os rotulos dos vertices Vi e Vj que formam o arco (Vi, Vj)
       e o custo deste');
81
82     for k:=1 to tam do
83     begin
84        ler_grafo(j);
85        retorno:
86        if (lin > ordem) or (col > ordem) then
87        begin
88           // Constatação de erro
89           gotoxy (4, j+3);
90           writeln ('Erro: O valor de Vi ou Vj eh maior que a ordem do grafo. ');
91           gotoxy (4, j+5);
92           write ('Digite "1" para continuar ou outro numero para encerrar o
             programa ');
93           readln (resp);
94           if resp = 1 then
95           begin
96              gotoxy(4, j+3);
97              writeln (' ':100);
98              gotoxy(4, j+5);
99              writeln (' ':100);
100             gotoxy(4, j);
101             writeln (' ':100);
102             ler_grafo(j);
103             resp := 0;
104             goto retorno;
105          end
106          else
107          begin
108             clrscr;
109             writeln ('Programa encerrado pelo usuario');
110             halt;   // encerra o programa
111          end;
112       end;
113       w[lin, col] := custo;
114       // cria lista de adjacência
```

```
115         vet_adj [lin] := vet_adj [lin] + 1;
116         mat_adj [lin, vet_adj [lin]] := col;
117
118         j := j + 1;
119      end;
120
121      // solicita dado sobre o vértice de origem
122      Writeln
         ('=============================================================================
         ====================================');
123      writeln;
124      writeln ('   Informe o rotulo do vertice de origem');
125      write ('   Vertice s: ');
126      readln (t);
127      Writeln
         ('=============================================================================
         ====================================');
128      ClrScr;
129      Writeln
         ('=============================================================================
         ====================================');
130   end;
131
132   Procedure Imprimir_Iteracao;
133   // Permite a impressão dos valores atualizados de d[i] e pai[i] em cada iteração
134   begin
135         for i :=1 to ordem do
136            writeln ('   vertice ', i,'           d[', i, ']: ',d[i]:12:2,'           
               pi[', i, ']: ', pai[i]);
137         writeln;
138         writeln ('Tecle ENTER para continuar');
139         readln;
140         writeln;
141   end;
142
143   Procedure Escrever_caminho (vert: integer);
144   // O procedimento determina o caminho do vertice de origem s ao vertice de destino v
145   begin
146      if vert = t then
147         write ('   ', t)
148      else if pai[vert] = 0 then
149         writeln ('   Nao existe trajeto entre os vertices.')
150      else
151      begin
152         Escrever_caminho (pai[vert]);
153         write (' - ', vert);
154      end;
155   end;
156
157   Procedure Determinar_caminho;
158   // Solicita dado do vértice de destino para imprimir o caminho até ele
159   begin
160      Writeln
         ('=============================================================================
         ====================================');
161      writeln ('   Determinacao do caminho ate o vertice v de destino.');
162      writeln ('   Informe o vertice v de destino.');
163      write ('   v: ');
164      readln (vert);
165      Escrever_caminho (vert);
166      writeln;
167      Writeln
         ('=============================================================================
         ====================================');
168   end;
```

```
169
170 Procedure definir_predecessor_inicial (ordem: integer; t:integer);
171 // Inicialização do Algoritmo de Dijkstra
172 var v: integer;
173
174 begin
175    for v:=1 to ordem do
176    begin
177       d[v] := 1000000;
178       pai[v] := 0;
179    end;
180    d[t] := 0;
181 end;
182
183 procedure inicializar_conjunto (var c : conjunto);
184 // Cria um conjunto vazio
185 begin
186    c.tam:= 0;
187 end;
188
189 function conjunto_vazio (c : conjunto): boolean;
190 // Retorna true se o conjunto c eh vazio e false caso contrario
191 begin
192    conjunto_vazio:= c. tam = 0;
193 end;
194
195 function cardinalidade (c : conjunto) : integer;
196 // Retorna a cardinalidade do conjunto c
197 begin
198    cardinalidade:= c. tam;
199 end;
200
201 Procedure ordenar_Q (var c: conjunto);
202 // Ordena os elementos do conjunto c de acordo com a o custo de d[v]
203 var
204    i, j: integer;
205    aux: integer;
206
207 begin
208    for i:=1 to cardinalidade (c) -1 do
209      for j:=1+i to cardinalidade (c) do
210        if d[c.vet[i]] > d[c.vet[j]] then
211        begin
212          aux := c.vet[j];
213          c.vet[j] := c.vet[i];
214          c.vet[i] := aux;
215       end;
216 end;
217
218 procedure inserir_no_conjunto (x: integer ; var c : conjunto);
219 // Insere o elemento x (vértice) no conjunto c
220 var i : integer;
221
222 begin
223       i := c . tam;
224       c.vet[i+1] := x;
225       c.tam := c . tam + 1;
226 end;
227
228 procedure remover_do_conjunto (x: integer ; var c : conjunto);
229 // Remove o elemento x (vértice) do conjunto c
230 var i: integer;
231
232 begin
233    for i := 1 to c.tam - 1 do
```

```
234         c.vet[ i ] := c.vet[ i +1];
235      c.tam := c.tam - 1;
236   end;
237
238   // Programa Principal
239   begin
240      tela_inicial;
241      Criar_grafo_ponderado;
242      definir_predecessor_inicial(ordem, t);
243      inicializar_conjunto (S);
244      inicializar_conjunto (Q);
245
246      for i:=1 to ordem do  // insere os vértices no conjunto Q
247         inserir_no_conjunto (i, Q);
248
249      Ordenar_Q (Q);  // ordena Q de acordo com o custo de d[v]
250
251      y:=1;
252      while not conjunto_vazio (Q)do   // Executa as iterações enquanto Q não for vazio
253      begin
254         writeln ('   Iteracao: ', y);
255         u := Q.vet[1]; // u recebe sempre o 1° elemento de Q
256         remover_do_conjunto (Q.vet[1], Q); // remove o 1° elemento de Q
257         inserir_no_conjunto (u, S); // u é inserido em S
258
259         for p:=1 to vet_adj[u] do // seleciona os vizinhos de u de acordo com a
          lista de adjacência
260         begin
261           v := mat_adj [u, p];
262           if d[v] > d[u] + w[u,v] then
263              begin
264                 d[v] := d[u] + w[u,v];
265                 pai[v] := u;
266              end;
267         end;
268         Ordenar_Q (Q); // ordena os elementos de Q em vista da alteração dos custos
          de d[v]
269         y:= y+1;
270         Imprimir_Iteracao; // imprime os valores atualizados de d[v] e pai[v] em
          cada iteração
271      end;
272      Determinar_caminho; // imprime o caminho do vértice s de origem ao vértice de
         destino v
273   end.
```

# APÊNDICE B – VERSÃO EM PASCAL DO ALGORITMO DE BELLMAN-FORD

```
1  Program Bellman_Ford;
2  uses
3     Crt; // A linha Uses Crt informa ao compilador que ele deve incluir a
       biblioteca Crt no seu programa
4          // Vai habilitar funções como gotoxy e clrscr
5  Const
6     MAX = 100;
7
8  Type
9     matriz = array [0..MAX, 0..MAX] of real;
10    vetor = array [0..MAX] of real;
11    vetor1 = array [0..MAX] of integer;
12
13    conjunto = record
14       tam: integer ;
15       vet: array [ 0..MAX+1] of integer ;
16    end;
17
18 var
19    ciclo_negativo: string;
20    w: matriz;
21    d: vetor;
22    pai: vetor1;
23    lin, col, i, j, k, t, resp, ordem, tam, vert: integer;
24    custo: real;
25
26 procedure tela_inicial;
27 // escreve na tela do monitor informações sobre o programa e recomendações ao usuário
28 begin
29    Writeln
      ('=================================================================================
      ==================================');
30    Writeln ('   Este programa eh utilizado para encontrar o caminho minimo de um
      vertice de origem dado aos demais vertices ');
31    writeln ('   do grafo, utilizando o Algoritmo de de Bellman=Ford.');
32    writeln;
33    Writeln ('   Durante a execucao do programa, serao  solicitados a  ordem e o
      tamanho do  grafo, o conjunto de  arcos e seus');
34    writeln ('   custos e o vertice de origem.');
35    Writeln
      ('=================================================================================
      ==================================');
36    writeln;
37 end;
38
39 procedure ler_grafo (var k: integer);
40 // procedimento para a leitura dos arcos ponderados do grafo (vértices de origem e
   destino e custo)
41 // gotoxy (x, y): representa a coordenada (x, y) do cursor na tela do monitor
42 begin
43    gotoxy (4,k);
44    write ('Vi: ');
45    read (lin);
46    gotoxy(18, k);
47    write ('Vj: ');
48    read (col);
49    gotoxy(34, k);
50    write ('w (',lin,', ',col, '): ');
51    read (custo);
52 end;
53
54 Procedure Criar_grafo_ponderado;
55 // Procedimento para criar o grafo ponderado. O usuário informará a ordem e o
   tamanho do grafo. Em seguida, o procedimento chamará
56 // o procedimento ler_grafo. Após a leitura dos dados, testes serão feitos para
```

```
   verificar se o vértice de origem e/ou destino fazem parte
57 // do grafo, no caso as variáveis lin e col terão que ser menores ou iguais a
   ordem do grafo. Os dados serão armazenados em uma matriz
58 // de valores, a qual representará o grafo ponderado
59 label
60    retorno;
61
62 var
63    h: integer;
64
65 begin
66    // Informação da ordem e do tamanho do grafo
67    writeln ('   Digite a ordem do grafo');
68    write ('   ordem: ');
69    readln (ordem);
70    writeln;
71    writeln ('   Digite o tamanho do grafo');
72    write ('   tamanho: ');
73    readln (tam);
74    Writeln
      ('=================================================================================
      ==================================');
75    writeln;
76
77    k:= 17;
78    writeln ('   Informe os rotulos dos vertices Vi e Vj que formam o arco (Vi, Vj)
      e o custo deste');
79
80    for h:=1 to tam do
81    begin
82       ler_grafo(k);
83       retorno:
84       if (lin > ordem) or (col > ordem) then
85       begin
86          // Constatação de erro
87          gotoxy (4, k+3);
88          writeln ('Erro: O valor de Vi ou Vj eh maior que a ordem do grafo. ');
89          gotoxy (4, k+5);
90          write ('Digite "1" para continuar ou outro numero para encerrar a
            digitacao: ');
91          readln (resp);
92          if resp = 1 then
93          begin
94             gotoxy(4, k+3);
95             writeln
               ('                                ');
96             gotoxy(4, k+5);
97             writeln
               ('                                ');
98             gotoxy(4, k);
99             writeln
               ('                                ');
100            ler_grafo(k);
101            goto retorno;
102         end
103         else
104         begin
105            clrscr;
106            writeln ('Programa encerrado pelo usuario');
107            halt;   // encerra o programa
108         end;
109      end;
```

```
110         w[lin, col] := custo;
111         k := k+1;
112      end;
113
114      // solicita dado sobre o vértice de origem
115      Writeln
         ('=================================================================================
         ==================================');
116      writeln;
117      writeln ('   Informe o rotulo do vertice de origem');
118      write ('   Vertice s: ');
119      readln (t);
120      Writeln
         ('=================================================================================
         ==================================');
121      ClrScr;
122      Writeln
         ('=================================================================================
         ==================================');
123   end;
124
125   Procedure Escrever_caminho (vert: integer);
126   // O procedimento determina o caminho do vertice de origem s ao vertice de destino v
127   begin
128      if vert = t then
129         write ('   ', t)
130      else if pai[vert] = 0 then
131         writeln ('   Nao existe trajeto entre os vertices.')
132      else
133      begin
134         Escrever_caminho (pai[vert]);
135         write (' - ', vert);
136      end;
137   end;
138
139   Procedure Determinar_caminho;
140   // Solicita dado do vértice de destino para imprimir o caminho até ele
141   begin
142      Writeln
         ('=================================================================================
         ==================================');
143      writeln ('   Determinacao do caminho ate o vertice v de destino.');
144      writeln ('   Informe o vertice v de destino.');
145      write ('   v: ');
146      readln (vert);
147      Escrever_caminho (vert);
148      writeln;
149      Writeln
         ('=================================================================================
         ==================================');
150   end;
151
152   Procedure Imprimir_Iteracao;
153   // Permite a impressão dos valores atualizados de d[i] e pai[i] em cada iteração
154   var l: integer;
155
156   begin
157         writeln ('   Dados consolidados:');
158         for l:=1 to ordem do
159            writeln ('      vertice: ', l,' ..... d[', l, ']: ',d[l]:9:4,' .....
               pi[', l, ']: ', pai[l]);
160         writeln;
161   end;
162
163   Procedure definir_predecessor_inicial (ordem: integer; t:integer);
```

```
164 // Inicialização do Algoritmo de Bellman-Ford
165 var v: integer;
166
167 begin
168    for v:=1 to ordem do
169    begin
170       d[v] := 1000000;
171       pai[v] := 0;
172    end;
173    d[t] := 0;
174 end;
175
176 Procedure Verificar_ciclo;
177 // Verifica a existência de ciclo negativo
178 Label
179    termino;
180
181 begin
182    ciclo_negativo := 'NAO';
183    for j:= 1 to ordem do
184       for k:= 1 to ordem do
185          if w[j,k] <> 0 then
186             if d[k] > (d[j] + w[j,k]) then
187             begin
188                ciclo_negativo := 'SIM';
189                goto termino;
190             end;
191
192    termino:
193    writeln ('   Ciclo negativo: ',ciclo_negativo);
194 end;
195
196 // Programa Principal
197 begin
198    tela_inicial;
199    Criar_grafo_ponderado;
200    definir_predecessor_inicial(ordem, t);
201
202    for i:=1 to ordem - 1 do   // Executa as iterações
203    begin
204       writeln ('   Iteracao ', i);
205       writeln ('      Arcos      v      d[v]      pi[v]');
206       for j:= 1 to ordem do
207         for k:= 1 to ordem do
208            if w[j,k] <> 0 then
209            begin
210               if d[k] > (d[j] + w[j,k]) then
211               begin
212                  d[k] := d[j] + w[j,k];
213                  pai[k] := j;
214                  if d[k] > 900000 then
215                  begin
216                     d[k]:=1000000;
217                     pai[k] := 0;
218                  end;
219                end;
220                writeln ('      w[',j,',',k,']     ',k,'   ', d[k]:9:4, '      ',
                   pai[k]);
221             end;
222       writeln;
223       Imprimir_Iteracao;   // imprime os valores atualizados de d[v] e pai[v] em
          cada iteração
224       writeln;
225       writeln ('Digite ENTER para continuar');
226       readln;
```

```
227       end;
228
229       Verificar_ciclo;     // verfica a existência de ciclo negativo
230       if ciclo_negativo = 'NAO' then
231          Determinar_caminho;   // imprime o caminho do vértice s de origem ao
          vértice de destino v
232   end.
```

## APÊNDICE C – INSTALAÇÃO E USO DO SOFTWARE LIVRE PASCAL

Para a utilização do Pascal no sistema operacional *Windows*, deve-se instalar o *software* livre *Free Pascal*, inicialmente realizando o *download* de uma versão do compilador, através da página *https://www.freepascal.org/download.html*, selecionando a opção *Windows 32-bit* ou *Windows 64-bit*, em seguida *SourceForge* e, por fim, *Download Latest Version*. Os códigos podem ser editados no próprio compilador *Free Pascal*, ou através do *software* livre *Geany*, editor de texto mais amigável aos usuários. Este *software* pode ser baixado através do endereço *https://www.geany.org/download*, selecionando o arquivo da opção *Windows (64-bit)*. Versões mais antigas também funcionam em sistemas *Windows de 32 bits*.

Para escrever um novo programa, deve-se selecionar na Barra de Ferramentas o botão “Novo”. Em seguida, seleciona-se na Barra de Menu “Arquivo” e “Salvar Como”, digita-se o nome do arquivo com a extensão “.pas”. Assim, um programa que calcule a média entre dois números poderá ser salvo com o nome de “media.pas”. No Editor de Texto, deve-se escrever o código do programa para compilá-lo ao término, selecionando na Barra de Ferramentas o botão “Compilar”. Em não havendo qualquer erro identificado pelo compilador, o programa poderá ser executado, selecionando-se na Barra de Ferramentas o botão “Executar”.

Nos smartphones, deve-se instalar o aplicativo *Pascal N-IDE – Editor And Compiler – Programming*, editor Pascal e Compilador para *Android*. Para criar novo programa, deve-se clicar no primeiro botão da Barra de Menu e, em seguida, nos botões “+” e “New”, digitando o nome do arquivo de extensão “.pas”. Após o código do programa ser digitado, deve-se compilá-lo clicando no botão “✓” da Barra de Menu. Em não havendo nenhum erro identificado pelo compilador, o programa poderá ser executado clicando no botão “▷” da Barra de Menu.