



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

INSTITUTO METRÓPOLE DIGITAL

BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

GRAFOS

FLAWBERT LORRAN DA SILVA COSTA

GABRIELLE DE VASCONCELOS BORJA

JADY LIMA DA SILVA

SEBASTIÃO FELLIPE PINTO LOPES

WENDY MILLER MOREIRA

**RELATÓRIO DE PESQUISA - IMPLEMENTAÇÃO DE ALGORÍTMOS EM
GRAFOS E DÍGRAFOS**

NATAL/RN

OUTUBRO/2025

SUMÁRIO

Resumo.....	3
Introdução.....	3
Referencial Teórico.....	3
Implementações.....	4
Funcionalidades Implementadas.....	4
Relatório de Atividades dos Participantes.....	6
Resultados Obtidos.....	7
Conclusões.....	17

Link do repositório com os códigos no GitHub:
<https://github.com/gabrielleborja/Projeto-Grafos>

Resumo

Este documento apresenta o progresso de um projeto aplicado, desenvolvido para a primeira etapa da matéria de Grafos da Universidade Federal do Rio Grande do Norte (UFRN). O objetivo principal foi criar, usando a linguagem C++, diversos algoritmos e ferramentas cruciais para o trabalho com grafos e dígrafos, desde a sua modelagem e apresentação até a realização de pesquisas e a avaliação de atributos particulares. O projeto, realizado em equipe, visou colocar em prática os ensinamentos teóricos da disciplina, tais como lista de adjacência, matriz de adjacência, matriz de incidência, e outros, em sistemas computacionais eficazes, sempre com foco na perspectiva dos grafos.

Introdução

O estudo dos grafos se destaca como um campo crucial na matemática, na ciência da computação e na tecnologia, com usos que abrangem desde as redes de computadores até a bioinformática e a engenharia, passando pela logística. Para quem almeja uma carreira nesta área, o conhecimento para modelar e solucionar problemas usando estruturas de grafos é indispensável.

Este projeto surgiu como avaliação da primeira unidade da matéria de Grafos, com o intuito de consolidar o saber teórico adquirido em aula, por meio da execução prática de algoritmos. O grupo criou um conjunto de códigos em C++, seguindo as orientações do professor, que englobam as funções obrigatórias e as opcionais especificadas, incluindo tanto grafos não direcionados quanto dígrafos. O trabalho foi feito em equipe, com divisão de tarefas que possibilitou a colaboração de todos os membros no processo de criação, execução e análise dos códigos.

Referencial Teórico

Por definição, um grafo $G = (N, M)$ é formado por um conjunto de vértices (N) e um conjunto de pares ordenados (M), arestas. Essa dupla de conjuntos finitos (N, M) tal que cada elemento de M define uma relação entre exatamente dois elementos distintos de N e não existem dois elementos distintos de M tais que definem a mesma relação entre o mesmo par de elementos de N .

A representação computacional de um grafo pode ser feita de várias maneiras, sendo as mais comuns a Matriz de Adjacências, a Lista de Adjacências e a Matriz de Incidência. Cada representação possui vantagens e desvantagens em termos de uso de memória e eficiência para diferentes operações. Nas questões, o grupo implementou soluções das três maneiras existentes. Para melhor entender cada uma delas, podemos fazer uma definição mais precisa de cada um deles:

- **Matriz de Adjacências:** Uma matriz quadrada onde as linhas e colunas representam os vértices do grafo. Uma entrada $M[i][j]$ é igual a 1 se existe uma aresta entre os vértices i e j , e 0 caso contrário.
- **Lista de Adjacências:** Para cada vértice, se armazena uma lista de seus vértices vizinhos. Esta representação normalmente é mais eficiente em termos de memória para grafos avulsos.
- **Matriz de Incidências:** Uma matriz onde as linhas representam os vértices e as colunas representam as arestas. A entrada $M[i][j]$ indica a relação entre o vértice i e a aresta j .

Além de examinar diagramas, a análise de grafos engloba múltiplos algoritmos que investigam suas qualidades. Entre eles, destacam-se a Busca em Largura (BFS) e a Busca em Profundidade (DFS), que são abordagens metódicas para percorrer as arestas e os vértices de um grafo.

Implementações

Todo o projeto foi implementado utilizando a linguagem C++, escolhida pelo grupo pela eficiência e controle sobre o gerenciamento da memória. Foram utilizadas classes para representar grafos e dígrafos e atender os requisitos solicitados pela atividade.

Funcionalidades Implementadas

A seguir está uma melhor descrição das funcionalidades que foram implementadas seguindo a descrição do projeto:

- Para **Grafos**:
 - Criação e conversão:
 - Questões 1 e 2: Foram implementadas funções para criar um grafo a partir de uma lista de adjacências e de uma matriz de adjacências. O código *Q01.cpp* permite a criação interativa do grafo, enquanto *Q02.cpp* lê a matriz de um arquivo *matriz.txt*.

- Questão 3: Criada a funcionalidade para gerar um grafo a partir de uma matriz de incidência, como foi visto no arquivo *questao3.cpp*.
- Questão 4: Nessa questão, foi implementada a conversão entre matriz e lista de adjacências ou vice-versa, permitindo a flexibilidade da representação do grafo, como visto no arquivo *q4.cpp*.
- Propriedades do grafo:
 - Questão 5: Calcula o grau de cada vértice, como visto no arquivo *questão5.cpp*.
 - Questão 6: Foi desenvolvida a verificação de adjacências entre dois vértices (*Q6.cpp*).
 - Questões 7 e 8: Funções para determinar o número total de vértices e arestas (*Q07.cpp* e *Q08.cpp*).
 - Questão 11: Implementada a verificação de conectividade do grafo utilizando a busca em largura (*questao11.cpp*).
- Algoritmos de busca e análise:
 - Questão 12: Nessa questão foi implementada a verificação se um grafo é bipartido, utilizando uma adaptação da busca em largura para colorir os vértices (*Q12.cpp*).
 - Questão 13: Implementação da Busca em Largura (BFS) a partir de um vértice inicial especificado pelo usuário (*q13.cpp*).
 - Questão 14: Implementação da Busca em Profundidade (DFS), identificando as arestas de retorno (*Q14.cpp*).
 - Questão 15: Determinação de pontos de articulação e blocos biconexos utilizando a função *lowpt* (*Q15.cpp*).
- Para **Dígrafos**:
 - Questão 16 e Questão 17: Foram implementadas as representações de um dígrafo a partir de matriz de adjacências (*Q16.cpp*) e matriz de incidência (*q17.cpp*).

- Questão 18: Foi implementado uma aplicação que com base em um dígrafo, retorna o seu subjacente (*Q18.cpp*).
- Questão 19: Implementação da Busca em Largura (BFS) considerando as ordens das arestas de um determinado dígrafo.
- Questão 20: Implementação da Busca em Profundidade (DFS) para dígrafos, com a classificação completa das arestas: árvores, retorno, avanço, cruzamento. Além disso, determina também o tempo de descoberta e finalização (*Q20.cpp*).

Relatório de Atividades dos Participantes

As questões de código implementadas foram divididas igualmente entre os membros do grupo de forma aleatória, de modo que todos pudessem contribuir com questões que envolvessem o conhecimento de grafos e dígrafos, além de um conhecimento geral de todo o conteúdo lecionado na matéria durante a primeira unidade. Além da divisão de questões, todo o grupo contribuiu para esse relatório e a gravação do vídeo de testes de dos códigos ficou na responsabilidade de um membro, de modo que a divisão final do trabalho ficou da seguinte forma:

- Flawbert Lorrán da Silva Costa: Questão 4, questão 9, questão 13, questão 17 e desenvolvimento do relatório.
- Gabrielle de Vasconcelos Borja: Questão 1, questão 8, questão 15, questão 20 e gravação do vídeo de teste de códigos.
- Jady Lima da Silva: Questão 2, questão 7, questão 14, questão 18 e desenvolvimento do relatório.
- Sebastião Fellipe Pinto Lopes: Questão 6, questão 10, questão 12, questão 16 e desenvolvimento do relatório.
- Wendy Miller Moreira: Questão 3, questão 5, questão 11, questão 19 e desenvolvimento do relatório.

Resultados Obtidos

1. Criação do Grafo a partir da Lista de Adjacências

```
*gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q01
Digite o número de vértices: 6
Digite o nome do vértice 1: 1
Digite a lista de vértices adjacentes: ('fin' para acabar)
2
4
5
fin
u
Digite o nome do vértice 2: 2
Digite a lista de vértices adjacentes: ('fin' para acabar)
1
3
fin
u
Digite o nome do vértice 3: 3
Digite a lista de vértices adjacentes: ('fin' para acabar)
fin
u
Digite o nome do vértice 4: 4
Digite a lista de vértices adjacentes: ('fin' para acabar)
6
5
fin
u
Digite o nome do vértice 5: 5
Digite a lista de vértices adjacentes: ('fin' para acabar)
1
3
4
fin
u
Digite o nome do vértice 6: 6
Digite a lista de vértices adjacentes: ('fin' para acabar)
2
fin
u
Grafo a partir da Lista de Adjacência:
u
1 - 2
1 - 4
1 - 5
2 - 3
2 - 6
3 - 5
4 - 5
u
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$
```

2. Criação do Grafo a partir da Matriz de Adjacências

```
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos> ./Q02 matriz.txt
Grafo:
1-2
1-4
1-5
1-6
2-1
2-3
2-4
2-5
3-2
3-3
3-4
4-3
4-5
5-1
5-2
5-4
6-1
6-2
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos>
```

3. Criação do Grafo a partir da Matriz de Incidência

```
PS C:\Users\11451479417\Desktop\Projeto-Grafos> ./questao3.exe
--- Questão 3: grafo a partir da matriz de incidência ---
grafo criado - lista de adjacência):
A -> B C
B -> A C D
C -> A B D
D -> B C
```

4. Conversão de matriz de adjacência para lista de adjacências e vice-versa.

```
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q04
0 grafo é direcionado? (1 - Sim, 0 - Não): 0
Digite o número de vértices: 5
Digite os rótulos dos vértices (ex: A B C ou 1 2 3 ...):
a b c d e

Escolha uma opção:
1 - Converter Matriz de Adjacência para Lista de Adjacência
2 - Converter Lista de Adjacência para Matriz de Adjacência
Opção: 1
Digite a matriz (5x5):
0 0 0 1 0
1 0 1 1 0
1 1 1 1 0
1 0 1 0 1
0 0 1 1 0

Lista de Adjacência:
a: d
b: a c d
c: a b c d
d: a c e
e: c d

gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q04
0 grafo é direcionado? (1 - Sim, 0 - Não): 0
Digite o número de vértices: 5
Digite os rótulos dos vértices (ex: A B C ou 1 2 3 ...):
a b c d e

Escolha uma opção:
1 - Converter Matriz de Adjacência para Lista de Adjacência
2 - Converter Lista de Adjacência para Matriz de Adjacência
Opção: 2
Digite o número de arestas: 5
Digite as arestas (formato: origem destino):
a b
c d
e c
b e
a d

Matriz de Adjacência:
a b c d e
a: 0 1 0 1 0
b: 1 0 0 0 1
c: 0 0 0 1 1
d: 1 0 1 0 0
e: 0 1 1 0 0
```


5. Função que calcula o grau de cada vértice.

```
PS C:\Users\11451479417\Desktop\Projeto-Graros> ./questao5.exe
--- Questão 5: Grau dos Vértices ---

> Lendo arquivo: GRAFO_0.txt
Grau de cada vértice:
a -> 1
b -> 3
c -> 3
d -> 2
e -> 2
f -> 3
g -> 2
h -> 2

> Lendo arquivo: GRAFO_1.txt
Grau de cada vértice:
a -> 1
b -> 5
c -> 4
d -> 3
e -> 2
f -> 1
i -> 1
h -> 1

> Lendo arquivo: GRAFO_2.txt
Grau de cada vértice:
1 -> 3
2 -> 4
3 -> 3
5 -> 2
6 -> 3
4 -> 2
7 -> 1
8 -> 3
9 -> 2
10 -> 2
11 -> 1
```

```
> Lendo arquivo: GRAFO_3.txt
Grau de cada vértice:
a -> 4
b -> 1
c -> 2
e -> 2
f -> 4
d -> 4
g -> 3
q -> 3
j -> 3
h -> 3
i -> 3
l -> 3
m -> 3
k -> 3
o -> 2
n -> 3
p -> 2
```

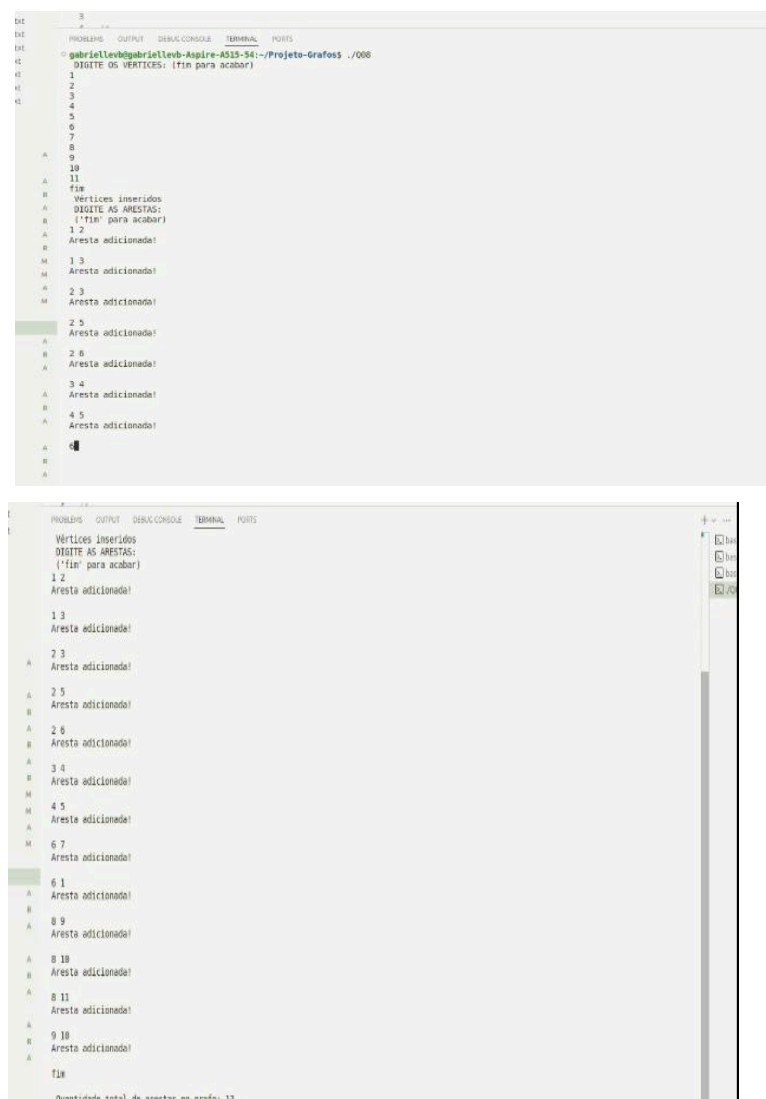
6. Função que determina se dois vértices são adjacentes.

```
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos> ./Q06 GRAFO_0.txt
Digite o primeiro vertice: a
Digite o segundo vertice: b
Os vertices sao adjacentes
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos> |
```

7. Função que determina o número total de vértices.

```
(base) PS C:\Users\jady1\OneDrive\Documentos\GitHub\grafos\exercicios\exercicio07> ./Q07.exe
a b c d e f g h
Numero de vertices: 8
|
```

8. Função que determina o número total de arestas.



```

8
DIGITE OS VERTICES: (fin para acabar)
1
2
3
4
5
6
7
8
9
10
11
fin
Vertices inseridos
DIGITE AS ARESTAS:
('fin' para acabar)
1 2
Aresta adicionada!
1 3
Aresta adicionada!
2 3
Aresta adicionada!
2 5
Aresta adicionada!
2 6
Aresta adicionada!
3 4
Aresta adicionada!
4 5
Aresta adicionada!
fin
Quantidade total de arestas no grafo: 13
```

9. Inclusão de um novo vértice usando Lista de Adjacências e Matriz de Adjacências.

```
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q09
O grafo é direcionado? (1 - Sim, 0 - Não): 0

Digite o número de arestas: 3
Digite as arestas (formato: origem destino):
a b
b c
c a

Grafo inicial:
Lista de Adjacência:
a: b c
b: a c
c: b a

Digite o rótulo do novo vértice: d
Quantas arestas o novo vértice possui? 1
Digite as arestas (formato: origem destino) envolvendo o novo vértice:
d a

Lista de Adjacência após inclusão:

Lista de Adjacência:
a: b c d
b: a c
c: b a
d: a

Matriz de Adjacência:
   a b c d
a: 0 1 1 1
b: 1 0 1 0
c: 1 1 0 0
d: 1 0 0 0
```

10. Exclusão de um vértice existente usando Lista de Adjacências e Matriz de Adjacências.

```
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos> ./Q10 GRAFO_0.txt
Grafo antes da exclusao:
GRAFO LISTA DE ADJACENCIA
a -> b
b -> a c d
c -> b d e
d -> b c
e -> c f
f -> e g h
g -> f h
h -> f g
-----

Digite o vertice a ser excluido: c
Vertice 'c' foi excluido.

Grafo apos a exclusao:
GRAFO LISTA DE ADJACENCIA
a -> b
b -> a d
d -> b
e -> f
f -> e g h
g -> f h
h -> f g
-----
```

11. Função que determina se um grafo é conexo ou não.

```
PS C:\Users\11451479417\Desktop\Projeto-Grafos> ./questao11
--- Questão 11: Verificar se um Grafo é Conexo ---

> Lendo arquivo: GRAFO_0.txt
Resultado: O grafo é conexo.

> Lendo arquivo: GRAFO_1.txt
Resultado: O grafo é conexo.

> Lendo arquivo: GRAFO_2.txt
Resultado: O grafo não é conexo.

> Lendo arquivo: GRAFO_3.txt
Resultado: O grafo é conexo.
```

12. Determinar se um grafo é bipartido

```
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos> ./Q12 GRAFO_0.txt
GRAFO LISTA DE ADJACENCIA
a -> b
b -> a c d
c -> b d e
d -> b c
e -> c f
f -> e g h
g -> f h
h -> f g
-----
Resultado: O grafo nao e bipartido.
```

13. Busca em Largura, a partir de um vértice específico.

```
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q13
Digite as arestas (formato: origem destino, ex: A B ou 1 2). Use 0 0 para encerrar:
1 2
1 3
2 3
2 5
2 6
3 4
4 5
6 7
6 1
8 9
8 10
8 11
9 10
0 0

Digite o vértice inicial para a busca: 2

Ordem de visita a partir do vértice 2:
2 1 3 5 6 4 7 gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$
```

14. Busca em Profundidade, com determinação de arestas de retorno, a partir de um vértice em específico

```
(base) PS C:\Users\jadyl\OneDrive\Documentos\GitHub\grafos\exercicios\exercicio14> .\Q14.exe

Digite o vertice inicial da busca: a
Aresta de retorno: c -> a
Vertice h visitado.
Vertice g visitado.
Vertice f visitado.
Vertice c visitado.
Vertice e visitado.
Vertice b visitado.
Vertice d visitado.
Vertice a visitado.

Status final:
a -> visitado
b -> visitado
c -> visitado
d -> visitado
e -> visitado
f -> visitado
g -> visitado
h -> visitado
```

15. Determinação de articulações e blocos (biconectividade), utilizando obrigatoriamente a função lowpt.

```
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q15
Digite o número de vértices: 13
Digite o nome do vértice 1: 1
Digite a lista de vértices adjacentes: ('fim' para acabar)
2
fim
Digite o nome do vértice 2: 2
Digite a lista de vértices adjacentes: ('fim' para acabar)
3
fim
Digite o nome do vértice 3: 3
Digite a lista de vértices adjacentes: ('fim' para acabar)
1
4
fim
Digite o nome do vértice 4: 4
Digite a lista de vértices adjacentes: ('fim' para acabar)
5
fim
Digite o nome do vértice 5: 5
Digite a lista de vértices adjacentes: ('fim' para acabar)
6
7
fim
Digite o nome do vértice 6: 7
Digite a lista de vértices adjacentes: ('fim' para acabar)
6
9
fim
Digite o nome do vértice 7: 8
Digite a lista de vértices adjacentes: ('fim' para acabar)
4
10
fim
Digite o nome do vértice 8: 9
Digite a lista de vértices adjacentes: ('fim' para acabar)
8
fim
Digite o nome do vértice 9: 6
Digite a lista de vértices adjacentes: ('fim' para acabar)
7
fim
Digite o nome do vértice 10: 1

Digite o nome do vértice 5: 5
Digite a lista de vértices adjacentes: ('fim' para acabar)
6
7
fim
Digite o nome do vértice 6: 7
Digite a lista de vértices adjacentes: ('fim' para acabar)
6
9
fim
A Digite o nome do vértice 7: 8
A Digite a lista de vértices adjacentes: ('fim' para acabar)
A 4
R 10
A fim
R Digite o nome do vértice 8: 9
A Digite a lista de vértices adjacentes: ('fim' para acabar)
R 8
A fim
R Digite o nome do vértice 9: 6
R Digite a lista de vértices adjacentes: ('fim' para acabar)
M 7
M fim
A Digite o nome do vértice 10: 10
A Digite a lista de vértices adjacentes: ('fim' para acabar)
M fim
M Digite o nome do vértice 11: 11
A Digite a lista de vértices adjacentes: ('fim' para acabar)
R 12
A fim
R Digite o nome do vértice 12: 12
A Digite a lista de vértices adjacentes: ('fim' para acabar)
A 13
A fim
R Digite o nome do vértice 13: 13
A Digite a lista de vértices adjacentes: ('fim' para acabar)
A 12
A fim
A Digite o primeiro vértice da busca: 1
A Bloco biconexo: (8,10) | Vértices: 10 8
R Bloco biconexo: (8,4) (9,8) (7,9) (6,7) (5,6) (4,5) | Vértices: 4 5 6 7 8 9
M Bloco biconexo: (3,4) | Vértices: 3 4
M Bloco biconexo final: (3,1) (2,3) (1,2) | Vértices: 1 2 3
Pontos de articulação:
3
4
8
```

16. Representação do Digrafo a partir da Matriz de Adjacências.

```
PS D:\Documents\UFRN\ufrn 2025.2\grafos\U1\Projeto-Grafos> ./Q16 DIGRAFO_0.txt
DIGRAFO MATRIZ DE ADJACENCIA
  a b d c e f g h
a 0 1 1 0 0 0 0 0
b 0 0 0 1 1 0 0 0
d 0 0 0 0 1 0 0 0
c 1 0 0 0 0 1 0 0
e 0 0 0 0 0 1 0 0
f 0 0 0 0 0 0 1 0
g 0 0 0 0 0 0 0 1
h 0 0 0 0 0 0 0 0
-----
```

17. Representação do Dígrafo a partir da Matriz de Incidência.

```
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q17
0 grafo é direcionado? (1 - Sim, 0 - Não): 1
Digite as arestas (formato: origem destino, use 0 0 para encerrar):
a b
b c
b d
c d
c e
e f
f g
f h
g h
0 0

Matriz de Incidência (8 x 9):

      e1  e2  e3  e4  e5  e6  e7  e8  e9
a    -1   0   0   0   0   0   0   0   0
b     1  -1  -1   0   0   0   0   0   0
c     0   1   0  -1  -1   0   0   0   0
d     0   0   1   1   0   0   0   0   0
e     0   0   0   0   1  -1   0   0   0
f     0   0   0   0   0   1  -1  -1   0
g     0   0   0   0   0   0   1   0  -1
h     0   0   0   0   0   0   0   1   1

Legenda das Arestas:
e1: a -> b
e2: b -> c
e3: b -> d
e4: c -> d
e5: c -> e
e6: e -> f
e7: f -> g
e8: f -> h
e9: g -> h

(Obs: valor '2' indica um laço no vértice correspondente)
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ █
```

18. Determinação do Grafo subjacente.

```
(base) PS C:\Users\jady1\OneDrive\Documentos\GitHub\grafos\exercicios\exercicio18> .\Q18.exe
Vertices digitados: a b c d e f g h
Grafo subjacente:
a - b
a - c
a - d
b - c
b - e
c - f
d - e
e - f
f - g
g - h
```

19. Busca em largura em dígrafo

```
PS C:\Users\11451479417\Desktop\Projeto-Grafos> ./questao19
--- Questão 19: Busca em Largura (BFS) para Dígrafos ---

> Lendo arquivo: DIGRAFO_0.txt
Ordem de visitação (BFS) a partir de 'a': a b d c e f g h

> Lendo arquivo: DIGRAFO1.txt
Ordem de visitação (BFS) a partir de '1': 1 2 3 4 5 6 8 7 10 9

> Lendo arquivo: DIGRAFO2.txt
Ordem de visitação (BFS) a partir de '1': 1 2 3 4 5 6 8 7 10 9 12 13

> Lendo arquivo: DIGRAFO3.txt
Ordem de visitação (BFS) a partir de 'a': a b e f c g d
PS C:\Users\11451479417\Desktop\Projeto-Grafos>
```

20. Busca em profundidade, com determinação de profundidade de entrada e de saída de cada vértice, e arestas de árvore, retorno, avanço e cruzamento.

```
Q20.cpp > main()
1 //questão 20 - Busca em profundidade, com determinação de profundidade de entrada e de saída de
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
gabriellevb@gabriellevb-Aspire-A515-54:~/Projeto-Grafos$ ./Q20
Digite o numero total de vertices: 13
Digite o numero total de arestas: 17
Digite as arestas (formato origem -> destino).
** Os vertices devem ser nomeados de 1 a 13.**
Aresta 1: 1 2
Aresta 2: 2 3
Aresta 3: 3 1
Aresta 4: 3 4
Aresta 5: 4 5
Aresta 6: 5 6
Aresta 7: 5 8
Aresta 8: 6 7
Aresta 9: 7 6
Aresta 10: 7 9
Aresta 11: 8
```



```
Q20.cpp > main()
1 //questão 20 - Busca em profundidade, com determinação de profundidade de entrada e de saída de
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Digite o numero total de vertices: 13
Aresta 5: 4 5
Aresta 6: 5 6
Aresta 7: 5 8
Aresta 8: 6 7
Aresta 9: 7 6
Aresta 10: 7 9
Aresta 11: 8 4
Aresta 12: 8 10
Aresta 13: 9 8
Aresta 14: 10 12
Aresta 15: 11 12
Aresta 16: 12 13
Aresta 17: 13 12

Grafo criado com sucesso!

Iniciando Busca em Profundidade (DFS)
--- Classificacao das Arestas ---
(1 -> 2) - Aresta de Arvore
(2 -> 3) - Aresta de Arvore
(3 -> 1) - Aresta de Retorno (ciclo)
(3 -> 4) - Aresta de Arvore
(4 -> 5) - Aresta de Arvore
(5 -> 6) - Aresta de Arvore
(6 -> 7) - Aresta de Arvore
(7 -> 6) - Aresta de Retorno (ciclo)
(7 -> 9) - Aresta de Arvore
(9 -> 8) - Aresta de Arvore
(8 -> 4) - Aresta de Retorno (ciclo)
(8 -> 10) - Aresta de Arvore
(10 -> 12) - Aresta de Arvore
(12 -> 13) - Aresta de Arvore
(13 -> 12) - Aresta de Retorno (ciclo)
(5 -> 8) - Aresta de Avanco
(11 -> 12) - Aresta de Cruzamento

--- Tempos de Descoberta e Finalizacao ---
Vertice 1: Descoberta = 1, Finalizacao = 12
Vertice 2: Descoberta = 2, Finalizacao = 11
Vertice 3: Descoberta = 3, Finalizacao = 10
Vertice 4: Descoberta = 4, Finalizacao = 9
Vertice 5: Descoberta = 5, Finalizacao = 8
Vertice 6: Descoberta = 6, Finalizacao = 7
Vertice 7: Descoberta = 7, Finalizacao = 6
Vertice 8: Descoberta = 9, Finalizacao = 4
Vertice 9: Descoberta = 8, Finalizacao = 5
Vertice 10: Descoberta = 10, Finalizacao = 3
Vertice 11: Descoberta = 13, Finalizacao = 13
```

Conclusões

O presente trabalho cumpriu com sucesso os objetivos de implementar e validar os algoritmos fundamentais da primeira unidade da teoria dos grafos. A utilização da estrutura de dados base provou-se eficiente e flexível. As implementações foram capazes de processar corretamente os arquivos de teste fornecidos, distinguindo grafos conexos de desconexos, calculando propriedades dos vértices e executando buscas em grafos direcionados, dentre outros fatores. O projeto serviu como uma valiosa experiência prática na aplicação de conceitos teóricos em soluções de software funcionais.