

## Exercícios: Aplicações de BFS e DFS

Estrutura de Dados (CCA410)  
Centro Universitário FEI  
2º Semestre de 2025

### 1 Exercício 1: BFS - Encontrar Caminho Mais Curto

#### 1.1 Objetivo

Aplicar o algoritmo BFS para encontrar o caminho mais curto em um grafo não-ponderado.

#### 1.2 Cenário

Você foi contratado para desenvolver um sistema de navegação para um campus universitário. O mapa do campus pode ser representado como um grafo onde os vértices são localizações importantes e as arestas são caminhos diretos entre elas.

#### 1.3 Grafo do Campus

**Vértices (Localizações):**

- A = Portaria Principal
- B = Biblioteca
- C = Auditório
- D = Secretaria
- E = Sala de Aula
- F = Laboratório de Informática
- G = Cantina
- H = Quadra de Esportes
- I = Estacionamento
- J = Centro de Convivência

**Arestas (Caminhos diretos):**

A-B, A-D, B-C, B-E, C-G, C-H, D-E, D-I, E-F, E-G, F-J, G-H, G-J, H-I, I-J

#### 1.4 Parte A: Implementação Manual

1. Desenhe o grafo representando o campus com base nas conexões fornecidas.
2. Execute BFS manualmente partindo da Portaria Principal (A) para encontrar o caminho mais curto até o Centro de Convivência (J):
  - Use a tabela abaixo para registrar cada passo
  - Registre o estado da fila a cada iteração
  - Marque as cores dos vértices (BRANCO, CINZA, PRETO)

Passo	Vértice Atual	Fila	Vizinhos Descobertos	Distâncias Atualizadas
0	-	[A]	-	A.d = 0
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

3. Identifique o caminho de A para J e sua distância mínima.

Caminho: \_\_\_\_\_

Distância mínima: \_\_\_\_\_

## 1.5 Parte B: Análise Comparativa

Execute BFS também partindo de A para encontrar os caminhos mais curtos para:

- Cantina (G): Caminho = \_\_\_\_\_, Distância = \_\_\_\_\_
- Estacionamento (I): Caminho = \_\_\_\_\_, Distância = \_\_\_\_\_
- Laboratório de Informática (F): Caminho = \_\_\_\_\_, Distância = \_\_\_\_\_

Análise: Qual localização está mais próxima da portaria?

## 1.6 Parte C: Modificação do Algoritmo

Modifique o algoritmo BFS para:

1. Parar quando encontrar o vértice destino J
2. Reconstruir o caminho completo usando um array de predecessores
3. Retornar tanto a distância quanto a sequência de vértices

Escreva o pseudocódigo da função modificada: BFS\_CaminhoMaisCurto(G, origem, destino):

## 1.7 Parte D: Questões Teóricas

1. Por que BFS garante encontrar o caminho mais curto em grafos não-ponderados?
2. Se as arestas tivessem pesos diferentes, BFS ainda funcionaria? Justifique.

## 2 Exercício 2: DFS - Detecção de Ciclos

### 2.1 Objetivo

Aplicar o algoritmo DFS para detectar ciclos em grafos direcionados.

### 2.2 Cenário

Você está desenvolvendo um sistema de gerenciamento de dependências para um projeto de software. As dependências podem ser representadas como um grafo direcionado onde cada vértice é um módulo e cada aresta representa "depende de". Ciclos neste grafo indicam dependências circulares, que devem ser detectadas e corrigidas.

### 2.3 Grafo de Dependências

Módulos (Vértices):

- A = ModuloAuth
- B = ModuloDB
- C = ModuloAPI
- D = ModuloUI
- E = ModuloConfig
- F = ModuloLogger
- G = ModuloUtils
- H = ModuloCache

Dependências (Arestas direcionadas):

A→B, A→F, B→H, B→G, C→A, C→D, D→A, D→G, E→F, E→G, F→G, H→G

### 2.4 Parte A: Representação e Análise

1. Desenhe o grafo direcionado com as dependências.
2. Identifique visualmente se existem ciclos óbvios.
3. Liste todas as dependências diretas de cada módulo:

- A depende de: \_\_\_\_\_
- B depende de: \_\_\_\_\_
- C depende de: \_\_\_\_\_
- D depende de: \_\_\_\_\_
- E depende de: \_\_\_\_\_
- F depende de: \_\_\_\_\_
- G depende de: \_\_\_\_\_
- H depende de: \_\_\_\_\_

### 2.5 Parte B: Execução Manual do DFS

Execute DFS manualmente para detectar ciclos:

1. Use a tabela para registrar tempos de descoberta e finalização:

Vértice	Cor	Tempo Descoberta	Tempo Finalização	Predecessor
A				
B				
C				
D				
E				
F				
G				
H				

2. **Registre cada back edge encontrada** (aresta que vai para um vértice CINZA):

Back edge: \_\_\_\_\_ → \_\_\_\_\_

Ciclo detectado: \_\_\_\_\_

3. **Classifique todas as arestas:**

- Tree edges (arestas da árvore DFS): \_\_\_\_\_
- Back edges (causam ciclos): \_\_\_\_\_
- Forward edges (para descendentes): \_\_\_\_\_
- Cross edges (entre subárvores): \_\_\_\_\_

## 2.6 Parte C: Implementação do Detector de Ciclos

Modifique o algoritmo DFS para criar um detector de ciclos que:

1. Retorne `true` se encontrar pelo menos um ciclo
2. Armazene informações sobre todos os ciclos encontrados
3. Para cada ciclo, identifique os vértices que o compõem: `DetectaCiclo_DFS(G)`

## 2.7 Parte D: Análise de Resultados

1. **Quantos ciclos** existem no grafo de dependências?
2. **Quais módulos** estão envolvidos em dependências circulares?
3. **Proponha uma solução** para quebrar os ciclos mantendo o máximo de dependências possível.

## 2.8 Parte E: Questões Teóricas

1. Por que back edges em DFS indicam ciclos em grafos direcionados?
2. Como você modificaria o algoritmo para grafos não-direcionados?
3. Qual é a diferença entre detectar ciclos e encontrar componentes fortemente conexos?

# 3 Critérios de Avaliação

## 3.1 Observações:

- Mostre todos os passos da execução manual
- Justifique suas escolhas de implementação
- Compare os resultados obtidos com as características teóricas dos algoritmos
- Discuta limitações e possíveis melhorias

**3.2 Entrega:**

- Desenhos dos grafos
- Tabelas preenchidas
- Pseudocódigo comentado
- Respostas às questões teóricas