BFS

Grafo:

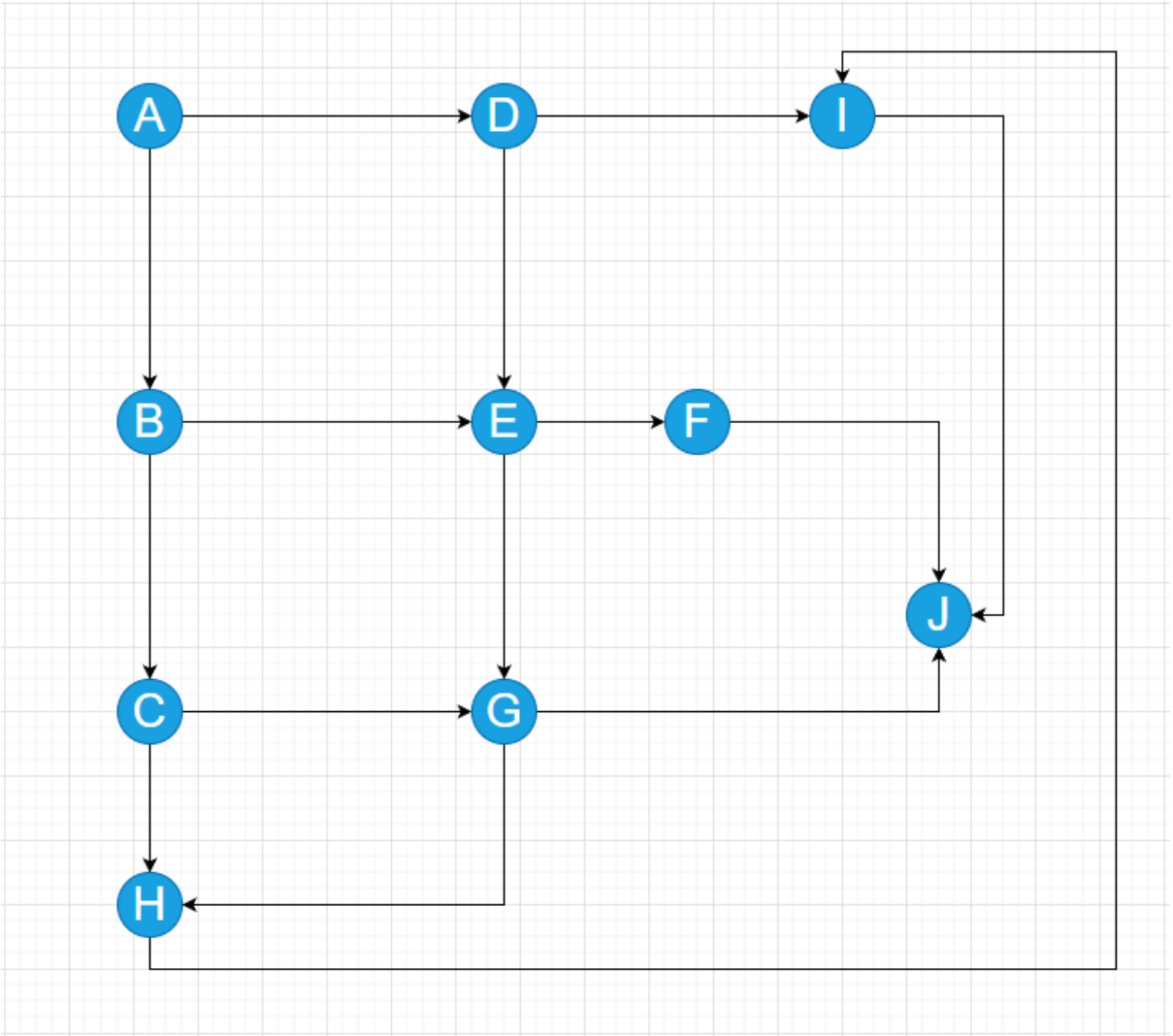


Tabela:

Passo	Vértice Atual	Fila	Vizinhos Descobertos	Dist ncias Atualizadas
0	A	[A]	B,D	A.d = 0
1	B	[D]	C,E	B.d = 1
2	D	[C,E]	I	D.d = 1
3	C	[E,I]	G,H	C.d = 2
4	E	[I,G,H]	F	E.d = 2
5	I	[G,H,F]	J	I.d = 3
6	G	[H,F,J]	-	G.d = 3
7	H	[F,J]	-	H.d = 3
8	F	[J]	-	F.d = 3
9	J	-	-	J.d = 3

Respostas:

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

Caminho: A->D->I->J

Distancia mınima: 3

1.5 Parte B: An´alise Comparativa

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

para: • Cantina (G): Caminho = A->B->C->G , Dist˜ancia = 3

• Estacionamento (I): Caminho = A->D->I, Dist˜ancia = 2

• Laboratório de Informática (F): Caminho =A->D->E->F , Dist˜ancia = 3 An´alise:

Qual localiza¸c˜ao est´a mais próxima da portaria? Biblioteca

1.6 Parte C: Modificac˜ao do Algoritmo

```
Função BFS_CaminhoMaisCurto(G, origem, destino):
  Para cada vértice v em G:
    v.cor = BRANCO
    v.dist = ∞
    v.pred = NULL

  origem.cor = CINZA
  origem.dist = 0
  origem.pred = NULL

  fila = [origem]

  Enquanto fila não estiver vazia:
    u = fila.pop(0)

    Se u == destino:
      caminho = []
      atual = destino
      Enquanto atual != NULL:
        Inserir atual no início de caminho
        atual = atual.pred
      Retornar (destino.dist, caminho)

    Para cada vizinho v de u:
      Se v.cor == BRANCO:
        v.cor = CINZA
        v.dist = u.dist + 1
        v.pred = u
        fila.append(v)

    u.cor = PRETO

  Retornar (∞, [])
```

1.7 Parte D: Quest˜oes Te´oricas

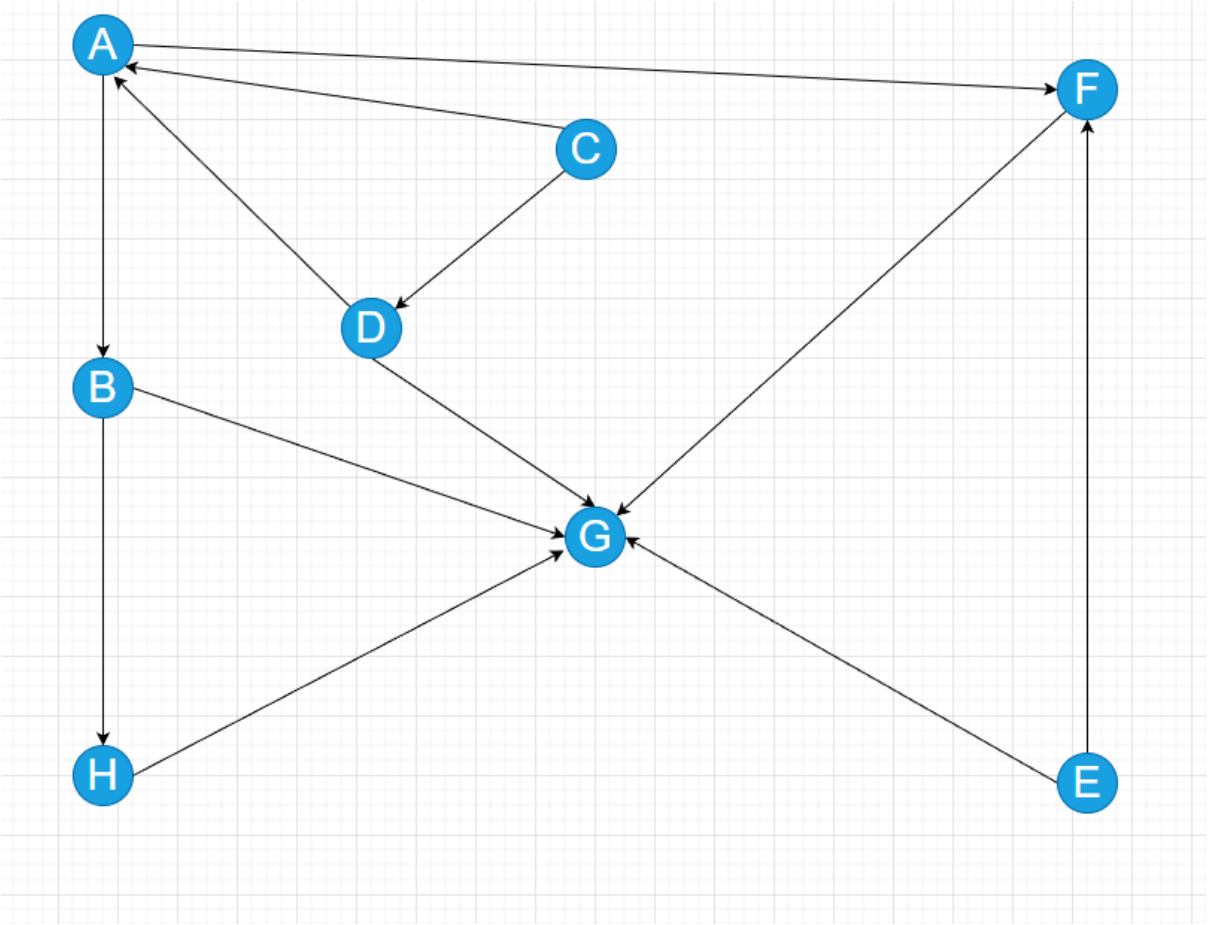
1. Por que BFS garante encontrar o caminho mais curto em grafos n˜ao-ponderados? Porque ele explora nível por nível, ou seja, ele vai eliminando todos os nós mais próximos do início até os mais longes, garantindo que o primeiro caminho encontrado seja o com menor arestas

2. Se as arestas tivessem pesos diferentes, BFS ainda funcionaria? Justifique.

Não, pois BFS ignora o peso.

DFS

Grafo:



Respostas:

2.4 Parte A: Representa¸c˜ao e An´alise

1. Desenhe o grafo direcionado com as depend^encias.
2. Identifique visualmente se existem ciclos ´óbvios. Não existe
3. Liste todas as depend^encias diretas de cada m´odulo:

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

2. Registre cada back edge encontrada (aresta que vai para um v´ertice CINZA):

Back edge: → C->A, D->A

Ciclo detectado: C->A->B->H->... e D->A->B-> ... ->D (dependência circular envolvendo D e A)

3. Classifique todas as arestas:

Aresta	Tipo
A → B	Tree edge
A → F	Tree edge
B → H	Tree edge
B → G	Tree edge
C → A	Back edge
C → D	Tree edge
D → A	Back edge
D → G	Forward edge
E → F	Forward edge
E → G	Forward edge
F → G	Forward edge
H → G	Forward edge

2.6 Parte C: Implementa¸c˜ao do Detector de Ciclos

```
Função DetectaCicloDFS(G):
    Para cada vértice v em G:
        v.cor = BRANCO
        v.pred = NULL

    ciclos = []

    Para cada vértice v em G:
        Se v.cor == BRANCO:
            DFS_Visita(v, ciclos)

    Se ciclos não estiver vazio:
        Retornar (true, ciclos)
    Senão:
        Retornar (false, [])

Função DFS_Visita(u, ciclos):
    u.cor = CINZA
    Para cada vizinho v de u:
        Se v.cor == BRANCO:
            v.pred = u
            DFS_Visita(v, ciclos)
        Se v.cor == CINZA:
            ciclo = [v]
            atual = u
            Enquanto atual != v:
                inserir atual no início de ciclo
                atual = atual.pred
            inserir v no final do ciclo
            ciclos.append(ciclo)
    u.cor = PRETO
```

2.7 Parte D: An´alise de Resultados

1. Quantos ciclos existem no grafo de depend˜encias? 2 ciclos
2. Quais m´odulos est˜ao envolvidos em depend˜encias circulares? A,B,C
3. Proponha uma solu¸c˜ao para quebrar os ciclos mantendo o m´aximo de depend˜encias poss´ıvel. Remover C->A ou D->A

2.8 Parte E: Quest˜oes Te´oricas

1. Por que back edges em DFS indicam ciclos em grafos direcionados?

   Porque ele vai para um vértice cinza que ainda está na pilha de recursão formando um loop.

2. Como voc˜e modificaria o algoritmo para grafos n˜ao-direcionados?

   Ignorar a aresta que vai para o predecessor.

3. Qual ´e a diferen¸ca entre detectar ciclos e encontrar componentes fortemente conexos?

   Detectar ciclo verifica se existe pelo menos uma dependência circular, já componentes fortemente conexos identifica todos os subconjuntos de vértices, onde cada vértice é alcançável a partir de qualquer outro no mesmo subconjunto.

tabela inteira da 2.5:

vertice	cor	desc	final	predecessor
C	CINZA	1	-	NULL
A	CINZA	2	-	C
B	CINZA	3	-	A
H	CINZA	4	5	B
B	PRETO	-	6	A
G	CINZA	7	8	B
A	PRETO	-	9	C
F	CINZA	10	11	A
A	PRETO	-	-	C
C	PRETO	-	12	-
D	CINZA	13	-	C
A	CINZA	14	15	D
G	CINZA	16	17	D
D	PRETO	-	18	C
E	CINZA	19	-	-
F	CINZA	20	21	E
G	CINZA	22	23	E
E	PRETO	-	24	-
F	PRETO	-	25	E
G	PRETO	-	26	E
H	PRETO	--	27	B