PDF Resposta: Djikstra e Bellman-Ford

Pedro Schneider

1 Algoritmo de Djisktra

1. **Execute** representando o algoritmo a partir do vértice A.

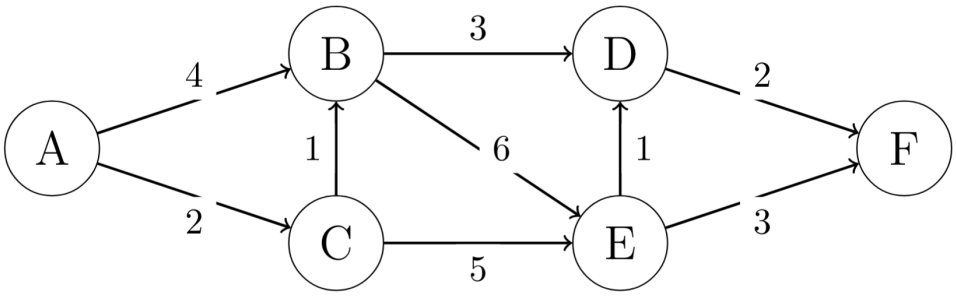


Figura 1: Grafo direcionado não-negativo

2. **Preencha** a tabela de execução:

Tabela 1: Execução do algoritmo Djikstra (do tipo **[**$v.d$**,** $v.pai$**]**)

Interação	**Vértice**	**A**	**B**	**C**	**D**	**E**	**F**
Inicial	-	0	[∞, *]	[∞, *]	[∞, *]	[∞, *]	[∞, *]
1	A	0	[4, A]	[2, A]	[∞, *]	[∞, *]	[∞, *]
2	C	0	[3, C]	[2, A]	[∞, *]	[7, C]	[∞, *]
3	B	0	[3, C]	[2, A]	[6, B]	[7, C]	[∞, *]
4	D	0	[3, C]	[2, A]	[6, B]	[7, C]	[8, D]
5	F	0	[3, C]	[2, A]	[6, B]	[7, C]	[8, D]
6	E	0	[3, C]	[2, A]	[6, B]	[7, C]	[8, D]

3. **Identifique o caminho** mínimo A para F.

Caminho: A-C-B-D-F

Distância mínima: 8

4. **Desenhe** a árvore de caminhos mínimos resultante

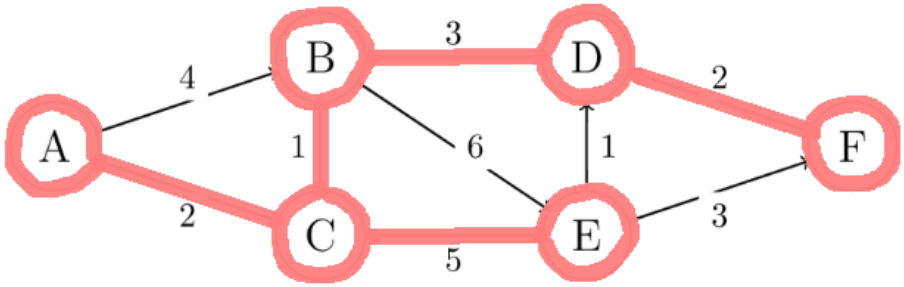


Figura 2: Árvore de caminhos mínimos do grafo acima

2 Algoritmo de Bellman-Ford

1. **Execute** o algoritmo de Bellman-Ford a partir do vértice S.

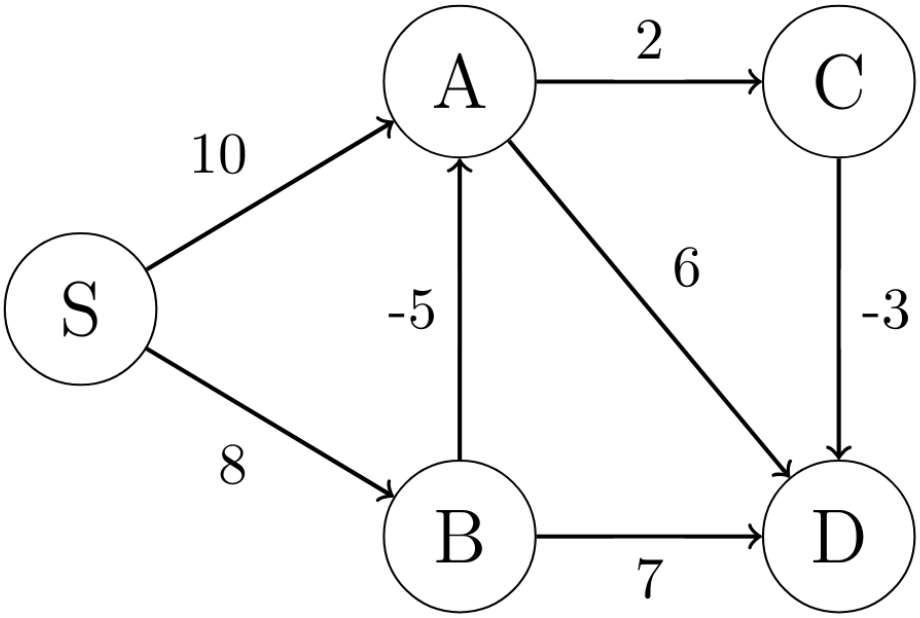


Figura 3: Grafo direcionado com pesos negativos

2. **Realize** $|V| - 1 = 4$ iterações de relaxamento

3. **Verifique** se existe ciclo negativo
   Resposta: Não

4. **Compare** o resultado com o que seria obtido pelo Djisktra
   Resposta: Como não há ciclos negativos, o resultado obtido é o mesmo usando ambos os algoritmos

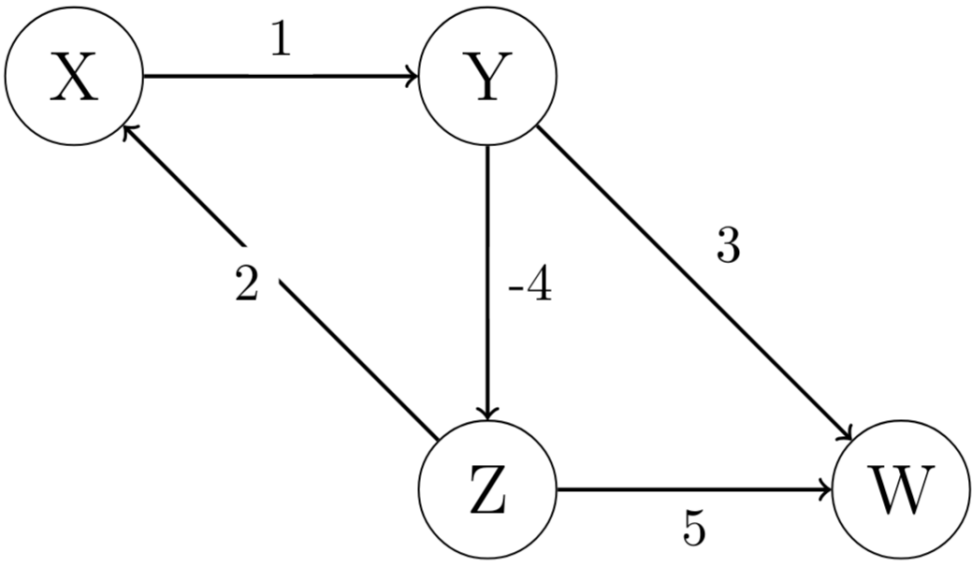
Tabela de execução

Tabela 2: Execução do algoritmo de Bellman-Ford (do tipo **[**$v.d$**,** $v.pai$**]**)

Interação	**S**	**A**	**B**	**C**	**D**
Inicial	0	[∞, *]	[∞, *]	[∞, *]	[∞, *]
1	0	[3, B]	[8, S]	[5, C]	[2, C]
2	0	[3, B]	[8, S]	[5, C]	[2, C]
3	0	[3, B]	[8, S]	[5, C]	[2, C]
4	0	[3, B]	[8, S]	[5, C]	[2, C]
Verificação de Ciclo Negativo					
Final	0	[3, B]	[8, S]	[5, A]	[2, C]

3 Detecção de Ciclo Negativo

Considere o grafo abaixo:



Pergunta: Este grafo possui ciclo negativo? Justifique sua resposta executando o algoritmo de Bellman-Ford.

Resposta: Sim, o ciclo $X \to Y \to Z \to X$. O algoritmo de Bellman-Ford itera sobre os vértices $|V| - 1$ vezes, se ao fim da iteração ainda houver a possibilidade de relaxamento de alguma aresta, ele retorna FALSO (ou seja, há ciclo negativo). Nesse caso, ao fim da 3a iteração, ainda é possivel relaxar a aresta $X \to Y$.

Análise do ciclo $X \to Y \to Z \to X$: Peso total = -1

4 Problema Prático

Uma empresa de logística precisa encontrar a rota de menor custo entre diferentes centros de distribuição. O grafo abaixo representa as conexões entre as cidades e os custos de transporte:

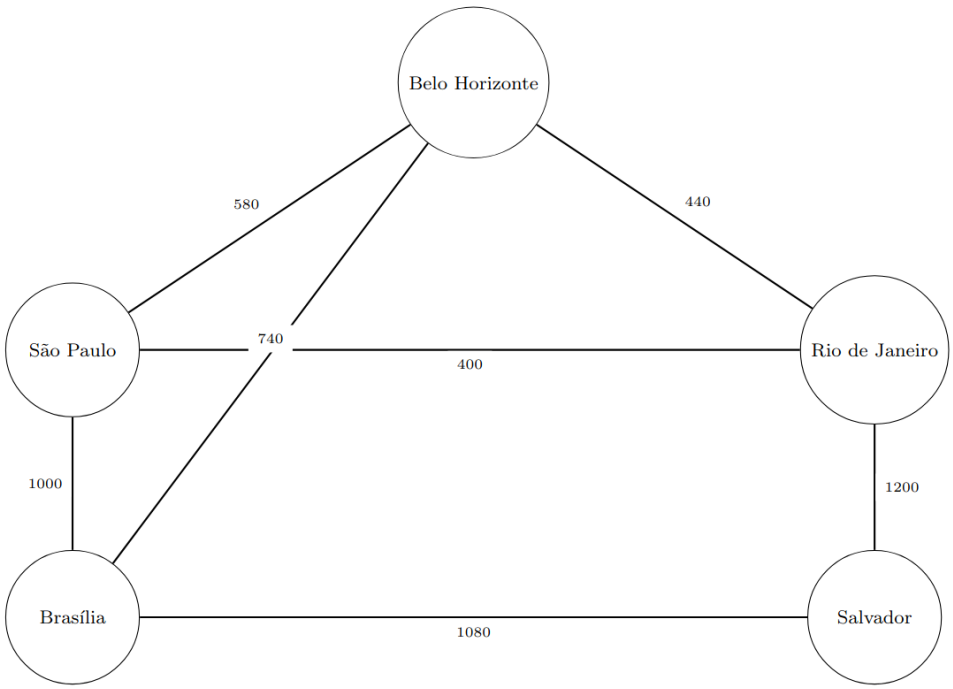


Figura 4: Grafo de logística

Tarefas:

1. Determine o caminho de menor custo de São Paulo até Salvador
   Resposta: São Paulo → Rio de Janeiro → Salvador

2. Qual algoritmo você usaria? Justifique
   Resposta: Djisktra, pois é um gráfico não-orientado sem pesos negativos e preciso encontrar uma rota ótima partindo de um ponto de origem definido.

3. Calcule o custo total da rota ótima
   Resposta: 1600

4. Liste todas as cidades no caminho ótimo
   Resposta: São Paulo, Rio de Janeiro, Salvador

5 Análise Comparativa

Complete a tabela comparativa abaixo:

Tabela 3: Comparação: Dijkstra vs Bellman–Ford

Critério	**Dijkstra**	**Bellman–Ford**
Pesos Negativos	Não lida	Lida
Detecção de Ciclos Negativos	Não detecta	Detecta
Complexidade Temporal	$O((V+E)\log V)$ com heap binário (ou $O(E+V\log V)$ com Fibonacci heap)	$O(V \cdot E)$
Estrutura de Dados Principal	Fila de prioridade / min-heap (priority queue)	Lista de arestas (array de arestas)
Estratégia	Gulosa: fixa o vértice de menor distância e o finaliza	Relaxamento repetido: relaxa todas as arestas $\|V\| - 1$ vezes e faz uma verificação extra
Melhor Aplicação	Grafos grandes (esparsos) com pesos não-negativos (rota mais curta rápida)	Grafos que podem conter arestas de peso negativo ou quando é preciso detectar ciclos negativos

6 Implementação em Pseudocódigo

Escreva o pseudocódigo para uma função que:

1. Receba um grafo e um vértice origem

2. Determine automaticamente qual algoritmo usar (Djisktra ou Bellman-Ford)

3. Retorne os caminhos mínimos ou detecte ciclos negativos

Listing 1: Pseudocódigo GrafosD_ou_BF(G, s)

```
GrafosD_ou_BF(G, s):
   # 1. Verificar se o grafo possui arestas de peso negativo
   possui_negativo <- FALSO
   para cada (u, v, w) em G.Arestas:
      se w < 0:
         possui_negativo <- VERDADEIRO
         interromper

   # 2. Escolher algoritmo apropriado
   se possui_negativo == FALSO:
      retornar Dijkstra(G, s)
   senao:
      returnar Bellman_Ford(G, s)


Dijkstra(G, s):
   para cada vertice v em G.V:
      v.dist <- +inf
      v.pai <- nulo
   s.dist <- 0

   Q <- conjunto com todos os vertices de G

   enquanto Q nao vazio:
      u <- vertice em Q com menor dist
      remover u de Q

      para cada (u, v, w) em G.ArestasSaindoDe(u):
         se v.dist > u.dist + w:
            v.dist <- u.dist + w
            v.pai <- u

   retornar distancias e predecessores

Bellman_Ford(G, s):
   para cada vertice v em G.V:
      v.dist <- +inf
```

```
38      v.pai <- nulo
39  s.dist <- 0
40
41  # Relaxar todas as arestas |V| - 1 vezes
42  para i de 1 ate |G.V| - 1:
43      para cada (u, v, w) em G.Arestas:
44          se v.dist > u.dist + w:
45              v.dist <- u.dist + w
46              v.pai <- u
47
48  # Verificacao de ciclo negativo
49  para cada (u, v, w) em G.Arestas:
50      se v.dist > u.dist + w:
51          imprimir "Ciclo negativo detectado!"
52          retornar ERRO
53
54  retornar distancias e predecessores
```