

Grafos – Caminhos Mínimos

Técnicas de Programação Avançada

Mateus Conrad B. da Costa

Ifes - Campus Serra

December 17, 2024

Caminho Mínimo e Otimização

- O problema do caminho mínimo está associado à **otimização**.
- Representa a menor solução possível ou melhor maneira de atingir um objetivo.
- Seleciona a solução mínima com relação a alguma variável.

Caminho Mínimo

- Em um grafo, o caminho mínimo é a menor relação entre dois vértices.
- Dado um vértice origem u e destino v , $C_{\min}(u, v)$ representa o menor peso entre os caminhos.
- **Exemplo:** Problema dos Odres de Vinho.

Caminho Mínimo - Origem Única

- Em um grafo ponderado $G = (V, A)$, dado um vértice de origem $s \in V$:
- Deseja-se obter os caminhos mínimos $\delta(s, v)$ para todos os vértices $v \in V$.

Problemas Mapeáveis no Problema Origem Única

- **Destino único:** Encontrar caminhos mínimos para um destino t pode ser transformado no problema origem única ao inverter a direção das arestas.
- **Par de vértices:** Pode ser resolvido pelo algoritmo de origem única.
- **Todos os pares:** Aplicar o algoritmo origem única para cada $v \in V$.

Representando Caminhos Mínimos

- Utiliza-se um vetor chamado **predecessor**.
- Para cada vértice v , $\text{predecessor}[v]$ indica o vértice anterior no caminho.
- Ao final do algoritmo, o vetor predecessor contém os caminhos mínimos.

Explicação do Relaxamento

- O processo de **relaxamento** testa se um caminho alternativo melhora a distância atual calculada.
- O vetor de distâncias $d[v]$ representa o menor custo conhecido até v . Se encontrarmos um caminho mais curto, atualizamos $d[v]$.
- **Importância:** O relaxamento é a base de algoritmos como Bellman-Ford e Dijkstra.

Algorithm 1 Relaxamento de uma aresta (u, v)

Require: Aresta (u, v) com peso $w(u, v)$, vetor de distâncias d ,
vetor de predecessores pred .

- 1: **if** $d[v] > d[u] + w(u, v)$ **then**
 - 2: $d[v] \leftarrow d[u] + w(u, v)$
 - 3: $\text{pred}[v] \leftarrow u$
 - 4: **end if**
-

Explicação do Algoritmo Bellman-Ford

- O algoritmo Bellman-Ford encontra caminhos mínimos a partir de um vértice de origem s em grafos com pesos negativos.
- Ele relaxa todas as arestas $|V| - 1$ vezes, garantindo a convergência para os menores pesos possíveis.
- Ao final, ele verifica se existe ciclo negativo no grafo.
- **Complexidade:** $O(V \cdot E)$.

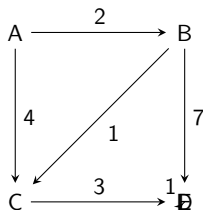
Algoritmo de Bellman-Ford

Algorithm 2 Bellman-Ford

Require: Grafo $G = (V, E)$, pesos w , vértice origem s

```
1: inicializaOrigemUnica( $G, pred, d$ )
2: for  $i = 1$  até  $|V| - 1$  do
3:   for cada aresta  $(u, v)$  em  $E$  do
4:     relax( $u, v, w$ )
5:   end for
6: end for
7: for cada aresta  $(u, v)$  em  $E$  do
8:   if  $d[v] > d[u] + w(u, v)$  then
9:     return FALSE
10:  end if
11: end for
12: return TRUE
```

Exemplo de Grafo e Vetor Predecessor



Vetor Predecessores:

- $\text{pred}[A] = \text{null}$
- $\text{pred}[B] = A$
- $\text{pred}[C] = B$
- $\text{pred}[D] = C$
- $\text{pred}[E] = D$

Explicação da Ordenação Topológica

- A **ordenação topológica** organiza os vértices de um grafo acíclico direcionado (DAG) de forma linear.
- Se existe uma aresta (u, v) , então u aparece antes de v na ordenação.
- É frequentemente usada como passo inicial para encontrar caminhos mínimos em DAGs.

Ordenação Topológica com DFS

Algorithm 3 ObtemOrdenaçãoTopológica(Grafo G)

Require: Grafo $G = (V, E)$ acíclico direcionado

- 1: Inicializa pilha S
 - 2: Marca todos os vértices como não visitados
 - 3: **for** cada vértice $v \in V$ **do**
 - 4: **if** v não visitado **then**
 - 5: DFS-Visit(v)
 - 6: **end if**
 - 7: **end for**
-

Ordenação Topológica com DFS

Algorithm 4 DFS-Visit(v)

- 1: **Saída:** Lista S com vértices ordenados
 - 2: **Procedimento** DFS-Visit(v):
 - 3: Marca v como visitado
 - 4: **for** cada $u \in \text{Adj}(v)$ **do**
 - 5: **if** u não visitado **then**
 - 6: DFS-Visit(u)
 - 7: **end if**
 - 8: **end for**
 - 9: Empilha v em S
-

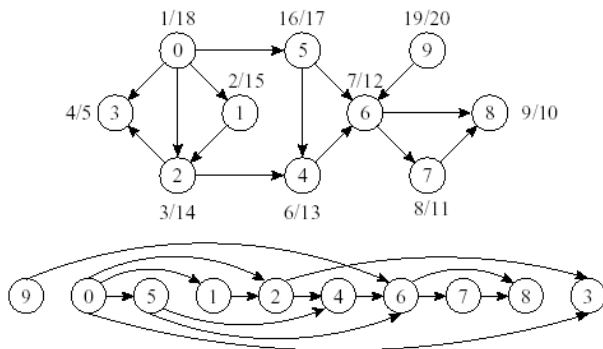
Caminho Mínimo em GAOs

Algorithm 5 Caminho Mínimo em GAOs

Require: Grafo $G = (V, E)$ acíclico, pesos w , vértice origem s

- 1: $LOT \leftarrow \text{ObtemOrdenaçãoTopologica}(G)$
 - 2: $\text{inicializaOrigemUnica}(G, \text{pred}, d)$
 - 3: **for** cada vértice $u \in LOT$ **do**
 - 4: **for** cada vértice $v \in \text{Adj}(u)$ **do**
 - 5: $\text{relax}(u, v, w)$
 - 6: **end for**
 - 7: **end for**
-

Exemplo de Ordenação Topológica



Explicação do Algoritmo Dijkstra

- O algoritmo de Dijkstra encontra os caminhos mínimos em grafos com pesos não negativos.
- Ele utiliza uma fila de prioridade para escolher o vértice com menor distância conhecida e relaxa suas arestas.
- **Complexidade:** $O((V + E) \log V)$ com heap binário.

Algoritmo de Dijkstra

Algorithm 6 Dijkstra

Require: Grafo $G = (V, E)$, pesos não negativos w , origem s

1: inicializaOrigemUnica($G, pred, d$)

2: Fila de prioridade $Q \leftarrow V$

3: **while** Q não está vazia **do**

4: $u \leftarrow \text{retiraMínimo}(Q)$

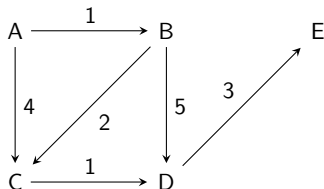
5: **for** cada $v \in \text{Adj}(u)$ **do**

6: $\text{relax}(u, v, w)$

7: **end for**

8: **end while**

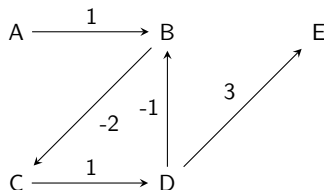
Exemplo de Dijkstra (sem ciclos negativos)



Vetor Predecessores (Final):

- $\text{pred}[A] = \text{null}$
- $\text{pred}[B] = A$
- $\text{pred}[C] = B$
- $\text{pred}[D] = C$
- $\text{pred}[E] = D$

Exemplo de Dijkstra (com ciclos negativos)



Vetor Predecessores (Final):

- $\text{pred}[A] = \text{null}$
- $\text{pred}[B] = D$ (loop detectado)
- $\text{pred}[C] = B$
- $\text{pred}[D] = C$
- $\text{pred}[E] = D$