Arestas/ciclos de custo negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, w) não possui arestas de custo negativo.
- Quando (G, w) possui arestas negativas, o algoritmo de Dijkstra não funciona (Exercício).
- Uma das dificuldades com arestas negativas é a possível existência de ciclos de custo negativo ou simplesmente ciclos negativos.

Ciclos negativos — uma dificuldade

- Se um ciclo negativo C é atingível a partir da fonte s, em princípio o problema não tem solução pois o "caminho" pode passar ao longo do ciclo infinitas vezes obtendo caminhos cada vez menores.
- Naturalmente, podemos impor a restrição de que os caminhos tem que ser simples, sem repetição de vértices. Entretanto, esta versão do problema é NP-difícil.
- Assim, vamos nos restringir ao Problema de Caminhos Mínimos sem ciclos negativos.

O algoritmo de Bellman-Ford

O algoritmo de Bellman-Ford recebe um grafo orientado (G, w) (possivelmente com arestas de custo negativo) e um vértice origem s de G

Ele devolve um valor booleano

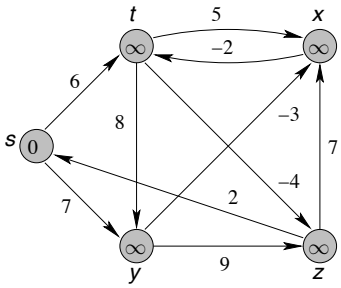
- FALSE se existe um ciclo negativo atingível a partir de s, ou
- TRUE e neste caso devolve também uma Árvore de Caminhos Mínimos com raiz s.

O algoritmo de Bellman-Ford

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  para i ← 1 até |V[G]| − 1 faça
3      para cada aresta (u, v) ∈ E[G] faça
4          RELAX(u, v, w)
5  para cada aresta (u, v) ∈ E[G] faça
6      se d[v] > d[u] + w(u, v)
7          então devolva FALSE
8  devolva TRUE
```

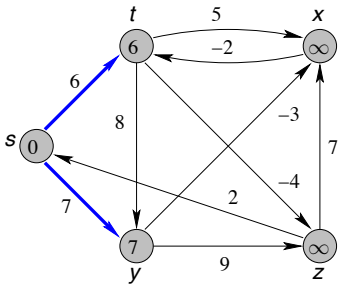
Complexidade de tempo: $O(VE)$

Exemplo (CLRS)



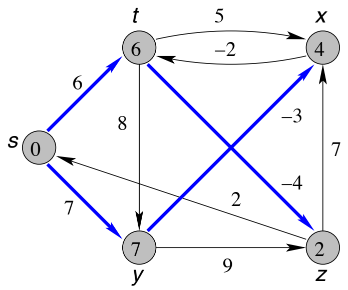
Ordem:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Exemplo (CLRS)



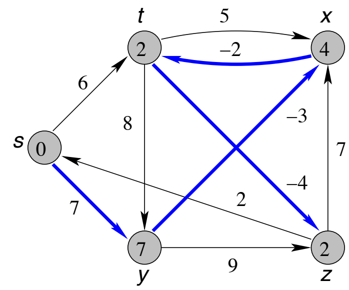
Ordem:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Exemplo (CLRS)



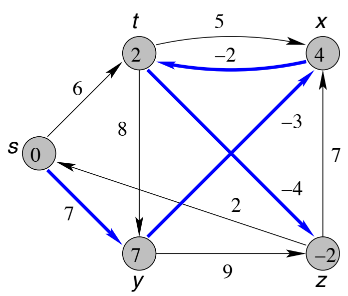
Ordem:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Exemplo (CLRS)



Ordem:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Exemplo (CLRS)



Ordem:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Corretude do algoritmo

Teorema 24.4 (CLRS).
Se (G, w) não contém ciclos negativos atingíveis por s, então no final

- o algoritmo devolve TRUE,
- $d[v] = dist(s, v)$ para $v \in V$ e
- $\pi[\,]$ define uma Árvore de Caminhos Mínimos.

Se (G, w) contém ciclos negativos atingíveis por s, então no final o algoritmo devolve FALSE.

Corretude do algoritmo

Primeiramente, vamos supor que o grafo não possui ciclos negativos atingíveis por s.

Relembrando...

(Lema 24.16, CLRS) Seja $P = (v_0 = s, v_1, \ldots, v_k)$ um caminho mínimo de v_1 a v_k e suponha que as arestas (v_0, v_1), (v_1, v_2),...,(v_{k-1}, v_k) são relaxadas nesta ordem. Então $d[v_k] = dist(s, v_k)$.

Corretude do algoritmo

Seja v um vértice atingível por s e seja

$$P = (v_0 = s, v_1, \ldots, v_k = v)$$

um caminho mínimo de s a v.

Note que P tem no máximo $|V| - 1$ arestas. Cada uma das $|V| - 1$ iterações do laço das linhas 2–4 relaxa todas as $|E|$ arestas.

Na iteração i a aresta (v_{i-1}, v_i) é relaxada.

Logo, pelo Lema 24.16, $d[v] = d[v_k] = dist(s, v)$.

Corretude do algoritmo

Se v é um vértice não atingível por s pode-se mostrar que $d[v] = \infty$ no final (Exercício).

Assim, no final do algoritmo $d[v] = dist(s, v)$ para $v \in V$.

Pode-se verificar que $\pi[\,]$ define uma Árvore de Caminhos Mínimos (veja CLRS para detalhes).

Corretude do algoritmo

Agora falta mostrar que BELLMAN-FORD devolve TRUE.

Seja (u, v) uma aresta de G. Então

$$\begin{aligned} d[v] &= dist(s, v) \\ &\leq dist(s, u) + w(u, v) \\ &= d[u] + w(u, v), \end{aligned}$$

e assim nenhum dos testes da linha 6 faz com que o algoritmo devolva FALSE. Logo, ele devolve TRUE.

Corretude do algoritmo

Agora suponha que (G, w) contém um ciclo negativo atingível por s.

Seja $C = (v_0, v_1, \ldots, v_k = v_0)$ um tal ciclo.

Então $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$.

Suponha por contradição que o algoritmo devolve TRUE. Então $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$ para $i = 1, 2 \ldots, k$.

Corretude do algoritmo

Somando as desigualdades ao longo do ciclo temos

$$\begin{aligned} \sum_{i=1}^{k} d[v_i] &\leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i). \end{aligned}$$

Como $v_0 = v_k$, temos que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.

Mas então $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$ o que contraria o fato do ciclo ser negativo.
Isto conclui a prova de corretude.

Caminhos mínimos entre todos os pares

O problema agora é dado um grafo (G, w) encontrar para todo para u, v de vértices um caminho mínimo de u a v.

Obviamente podemos executar $|V|$ vezes um algoritmo de Caminhos Mínimos com Mesma Origem.

- Se (G, w) não possui arestas negativas podemos usar o algoritmo de Dijkstra implementando a fila de prioridade como

  um vetor: $|V|.O(V^2 + E) = O(V^3 + VE)$ ou
  min-heap binário: $|V|.O(E \lg V) = O(VE \lg V)$ ou
  heap de Fibonacci: $|V|.O(V \lg V + E) = O(V^2 \lg V + VE)$.
- Se (G, w) possui arestas negativas podemos usar o algoritmo de Bellman-Ford: $|V|.O(VE) = O(V^2 E)$.

O algoritmo de Floyd-Warshall

Veremos agora um método direto para resolver o problema que é assintoticamente melhor se G é denso.

O algoritmo de Floyd-Warshall baseia-se em programação dinâmica e resolve o problema em tempo $O(V^3)$.

O grafo (G, w) pode ter arestas negativas, mas suporemos que não contém ciclos negativos.

Vamos adotar a convenção de que (i, j) não é uma aresta de G então $w(i, j) = \infty$.

Estrutura de um caminho mínimo

Seja $P = (v_1, v_2, \ldots, v_l)$ um caminho (simples).

Um vértice intermediário de P é qualquer vértice de P distinto de v_1 e v_l, ou seja, em $\{v_2, \ldots, v_{k-1}\}$.

Para simplificar, suponha que $V = \{1, 2, \ldots, n\}$.

Estrutura de um caminho mínimo

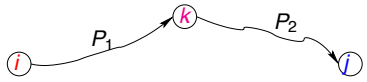
Sejam i e j dois vértices de G. Considere todos os caminhos cujos vértices intermediários pertencem a $\{1, \ldots, k\}$. Seja P um caminho mínimo entre todos eles.

O algoritmo de Floyd-Warshall explora a relação entre P e um caminho mínimo de i a j com vértices intermediários em $\{1, \ldots, k-1\}$.

Se k não é um vértice intermediário de P então P é um caminho mínimo de i a j com vértices intermediários em $\{1, \ldots, k-1\}$.

Estrutura de um caminho mínimo

- Se k é um vértice intermediário de P então P pode ser dividido em dois caminhos P_1 (com início em i e fim em k) e P_2 (com início em k e fim em j).



- P_1 é um caminho mínimo de i a k com vértices intermediários em $\{1, \ldots, k-1\}$
- P_2 é um caminho mínimo de k a j com vértices intermediários em $\{1, \ldots, k-1\}$.

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o custo de um caminho mínimo de i a j com vértices intermediários em $\{1, 2, \ldots, k\}$.

Quando $k = 0$ então $d_{ij}^{(0)} = w(i, j)$.

Temos a seguinte recorrência:

$$d_{ij}^{(k)} = \begin{cases} w(i,j) & \text{se } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{se } k \geq 1. \end{cases}$$

Assim, queremos calcular a matrix $D^{(n)} = (d_{ij}^{(n)})$ com $d_{ij}^{(n)} = dist(i, j)$.

Algoritmo de Floyd-Warshall

A entrada do algoritmo é a matriz $W = (w(i, j))$ com $n = |V|$ linhas e colunas.
A saída é a matriz $D^{(n)}$.

```
FLOYD-WARSHALL(W)
1  D^(0) ← W
2  para k ← 1 até n faça
3      para i ← 1 até n faça
4          para j ← 1 até n faça
5              d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
6  devolva D^(n)
```

Complexidade: $O(V^3)$

Como encontrar os caminhos?

O algoritmo precisa devolver também uma matriz $\Pi = (\pi_{ij})$ tal que $\pi_{ij} = \text{NIL}$ se $i = j$ ou se não existe caminho de i a j, e caso contrário, π_{ij} é o predecessor de de j em algum caminho mínimo a partir de i.

Podemos computar os predecessores ao mesmo tempo que o algoritmo calcula as matrizes $D^{(k)}$. Determinamos uma seqüência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$ e $\pi_{ij}^{(k)}$ é o predecessor de j em um caminho mínimo a partir de i com vértices intermediários em $\{1, 2, \ldots, k\}$.

Quando $k = 0$ temos

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{se } i = j \text{ ou } w(i,j) = \infty, \\ i & \text{se } i \neq j \text{ e } w(i,j) < \infty. \end{cases}$$

Como encontrar os caminhos?

Para $k \geq 1$ procedemos da seguinte forma. Considere um caminho mínimo P de i a j.

Se k não aparece em P então tomamos como predecessor de j o predecessor de j em um caminho mínimo de i a j com vértices intermediários em $\{1, 2, \ldots, k-1\}$.

Caso contrário, tomamos como predecessor de j o predecessor de j em um caminho mínimo de k a j com vértices intermediários em $\{1, 2, \ldots, k-1\}$.
Formalmente,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{se } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{se } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

Exercício. Incorpore esta parte no algoritmo!