Análise e Síntese de Algoritmos

Dijkstra. Bellman-Ford.
Caminhos mais curtos em DAGs.

CLRS Cap. 24

Prof. Pedro T. Monteiro

IST - Universidade de Lisboa

2024/2025

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica [CLRS, Cap.15]
  - Algoritmos greedy [CLRS, Cap.16]
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares [CLRS, Cap.22]
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Árvores abrangentes [CLRS, Cap.23]
  - Fluxos máximos [CLRS, Cap.26]
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais
  - Complexidade Computacional [CLRS, Cap.34]

Resumo

Caminhos mais curtos em DAGs

Intuição

- Um DAG não tem ciclos
- Mesmo com pesos negativos, o caminho mais curto é bem definido
- Ordenar os vértices por ordem topológica
- Passagem única pelos vértices ordenados

DAG-shortest-path(G, w, s)

```
ordenação topológica dos vértices de G
Initialize-Single-Source(G, s)
for each u ∈ V.G (ordem topológica) do
  for each v ∈ Adj[u] do
    Relax(u, v, w)
  end for
end for
```

Complexidade

- $\Theta(V + E)$

Correcção do algoritmo

Dado $G = (V, E)$, dirigido, acíclico, como resultado do algoritmo, temos que $d[v] = \delta(s, v)$ para todo o $v \in V$

- Seja v atingível a partir de s, e seja $p =< v_0, v_1, \ldots, v_k >$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- Ordenação topológica implica que analisados por ordem (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)

Prova por indução

$d[v_i] = \delta(s, v_i)$ sempre que cada vértice v_i é terminado

- Base: Estimativa de s não alterada após inicialização; $d[s] = d[v_0] = \delta(s, v_0) = 0$
- Indução: $d[v_{i-1}] = \delta(s, v_{i-1})$ após terminar análise de v_{i-1}
- Relaxação do arco (v_{i-1}, v_i) causa $d[v_i] = \delta(s, v_i)$, pelo que $d[v_i] = \delta(s, v_i)$ após terminar análise de v_i

Resumo

Algoritmo Bellman-Ford

Intuição

- Permite pesos negativos
- Identifica existência de ciclos negativos
- Baseado em sequência de passos de relaxação
- Requer manutenção da estimativa associada a cada vértice

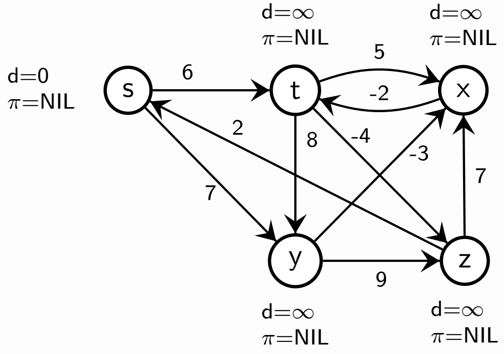
Algoritmo Bellman-Ford

Bellman-Ford(G,w,s)

```
Initialize-Single-Source(G,s)
for i ← 1 to |G.V|-1 do
  for each (u,v) ∈ G.E do
    Relax(u,v,w)
  end for
end for
for each (u,v) ∈ G.E do
  if d[v] > d[u] + w(u,v) then
    return False
  end if
end for
return True
```

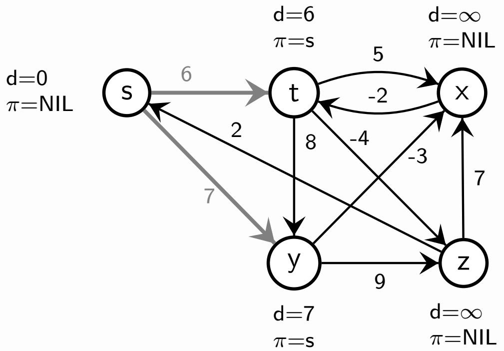

Algoritmo Bellman-Ford

Exemplo
Iteração: 0
Ordem arcos:
(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



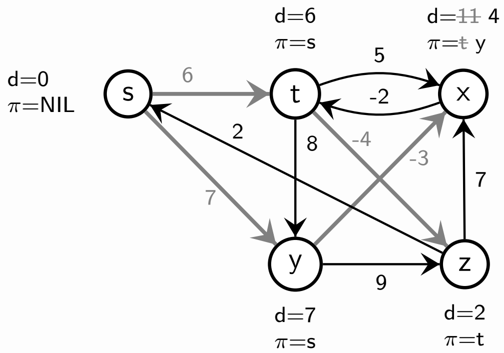
Algoritmo Bellman-Ford

Exemplo
Iteração: 1
Ordem arcos:
(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



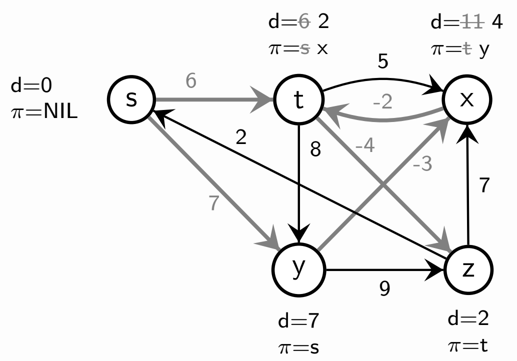
Algoritmo Bellman-Ford

Exemplo
Iteração: 2
Ordem arcos:
(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



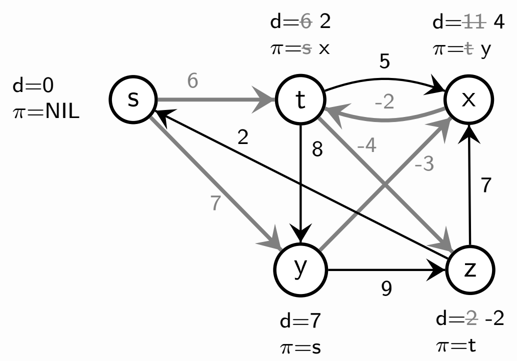
Algoritmo Bellman-Ford

Exemplo
Iteração: 3
Ordem arcos:
`(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)`

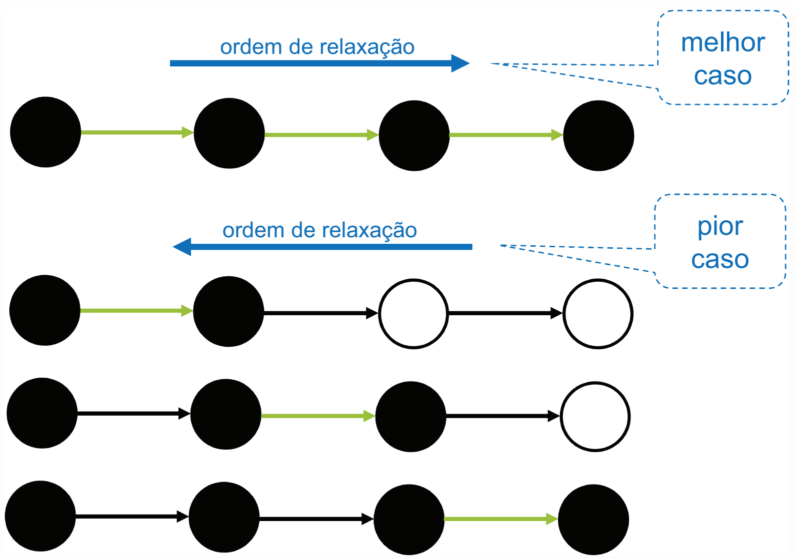


Algoritmo Bellman-Ford

Exemplo
Iteração: 4
Ordem arcos:
`(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)`



Algoritmo Bellman-Ford



Algoritmo Bellman-Ford

Complexidade

- Inicialização: $\Theta(V)$
- A complexidade dos ciclos é $O(VE)$ (dois ciclos, em V e em E)
  - Em cada iteração todos os arcos são relaxados
- Complexidade algoritmo Bellman-Ford: $O(VE)$

Algoritmo Bellman-Ford

Correcção do algoritmo

Se $G = (V, E)$ não contém ciclos negativos, então após a aplicação do algoritmo de Bellman-Ford, $d[v] = \delta(s, v)$ para todos os vértices atingíveis a partir de s

- Seja v atingível a partir de s, e seja $p =< v_0, v_1, \ldots, v_k >$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- p é simples, pelo que $k \leq |V| - 1$

Prova por indução

$d[v_i] = \delta(s, v_i)$ para $i = 0, 1, \ldots, k$, após iteração i sobre os arcos de G, e que valor não é alterado posteriormente

- Base: $d[v_0] = \delta(s, v_0) = 0$ após inicialização (e não se altera)
- Passo indutivo: assumir $d[v_{i-1}] = \delta(s, v_{i-1})$ após iteração (i-1)
- Arco (v_{i-1}, v_i) relaxado na iteração i, pelo que $d[v_i] = \delta(s, v_i)$ após iteração i (e não se altera)

Algoritmo Bellman-Ford

Correcção do algoritmo (cont.)

Se $G = (V, E)$ não contém ciclos negativos (atingíveis a partir de s), o algoritmo de Bellman-Ford retorna TRUE, caso contrário FALSE

- Se não existem ciclos negativos, resultado anterior assegura que para qualquer arco $(u, v) \in E$, $d[v] \leq d[u] + w(u, v)$, pelo que teste do algoritmo falha para todo o (u, v) e o valor retornado é TRUE
- Caso contrário, na presença de pelo menos um ciclo negativo atingível a partir de s, $c =< v_0, v_1, \ldots, v_k >$, onde $v_0 = v_k$, temos que $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

Algoritmo Bellman-Ford

Correcção do algoritmo (cont.)

Prova por contradição

Admitir que algoritmo retorna TRUE na presença de ciclo negativo

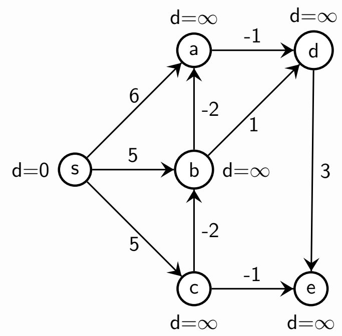
- Para que devolva TRUE é necessário que $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$, para $i = 1, \ldots, k$
- Somando as desigualdades ao longo do ciclo temos que:

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

- Note-se que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$ por ser um ciclo
- Temos então que $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$, o que contradiz a existência de um ciclo negativo. Logo, o algoritmo retorna FALSE

Algoritmo Bellman-Ford

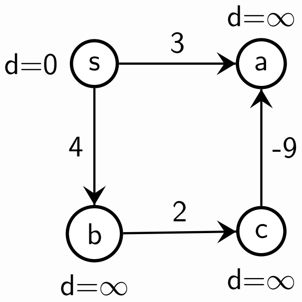
Exercício a iniciar em s



Ordem arcos:
`(c,e),(c,b),(d,e),(b,d),(b,a),(a,d),(s,c),(s,b),(s,a)`

Algoritmo Bellman-Ford

Exercício a iniciar em *s*



Ordem arcos:
`(c,a),(b,c),(s,a),(s,b)`

Resumo

Algoritmo Dijkstra

- Apenas permite pesos não negativos
- Complexidade: $O((V + E)\ log\ V)$

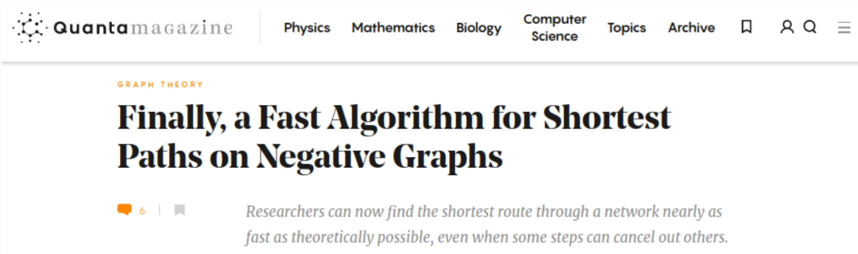
Algoritmo Bellman-Ford

- Permite pesos negativos e identifica ciclos negativos
- Complexidade: $O(VE)$

Caminhos mais curtos em DAGs

- Grafos acíclicos (ordenação topológica dos vértices)
- Complexidade: $O(V + E)$

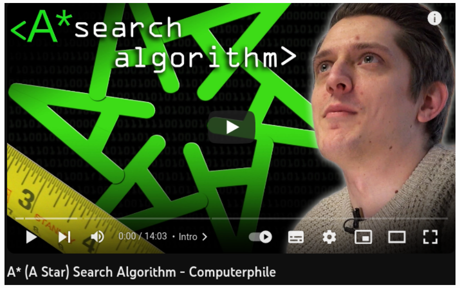
Apesar de estudarmos algoritmos da 2a metade do século XX, não quer dizer que não haja investigação nesta área. Aqui fica o link para quem tiver curiosidade (com links para os papers originais): https://www.quantamagazine.org/finally-a-fast-algorithm-for-shortest-paths-on-negative-graphs-20230118/


Quantamagazine Physics Mathematics Biology Computer Science Topics Archive

GRAPH THEORY

Finally, a Fast Algorithm for Shortest Paths on Negative Graphs

Researchers can now find the shortest route through a network nearly as fast as theoretically possible, even when some steps can cancel out others.



Não estudamos o A* (não faz parte do programa), mas fica aqui uma explicação como extensão do Dijkstra, para quem tiver curiosidade: https://www.youtube.com/watch?v=ySN5Wnu88nE

Questões?

Dúvidas?