Projeto e Análise de Algoritmos (PAA)
Caminhos Mínimos de Todos para Todos, Algoritmo de Johnson

O problema dos caminhos mínimos de todos para todos

Neste problema recebemos como entrada:

- um grafo orientado (ou dirigido) G=(V,E),
- com custo c(e) em cada aresta ‘e’ em E.

E queremos encontrar:

- o valor do caminho mínimo de ‘u’ até ‘v’ para todo par u, v em V.
- Também gostaríamos que esses caminhos fossem devolvidos.

No caso de haver um circuito negativo na entrada

- o valor dos caminhos mínimos não está bem definido.
    - Por isso, tal fato deve ser reportado.

Uma estratégia para resolver o problema é

    - executar um algoritmo de caminhos mínimos a partir de cada vértice.
- Este procedimento leva tempo
    - O(n * tempo do algoritmo de caminhos mínimos utilizado).

Se os custos da entrada são não negativos $\Theta(m \log n)$

- podemos usar o algoritmo de Dijkstra, que leva tempo Theta(m log n).

- Portanto, nosso algoritmo para encontrar
  - caminhos mínimos de todos para todos levará tempo
    - n * Theta(m log n) = Theta(n m log n). $\rightarrow n \cdot \Theta(m \log n)$

- Se o grafo é esparso, i.e., m = Theta(n) temos
  - Theta(n m log n) = Theta (n^2 log n). $m = \Theta(n) \Rightarrow \Theta(n^2 \log n)$
- o que é muito bom, já que, apenas para devolver os valores das soluções,
  - precisamos preencher n^2 posições.

- Se o grafo for denso, i.e., m = Theta(n^2) temos
  - Theta(n m log n) = Theta (n^3 log n) $m = \Theta(n^2) \Rightarrow \Theta(n^3 \log n)$
- o que é razoável, já que gastamos pouco mais que tempo linear
  - por caminho mínimo encontrado.

Se os custos da entrada podem ser negativos,

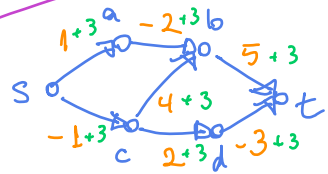
- temos o algoritmo de Floyd-Warshall, que leva tempo Theta(n^3), $\Theta(n^3)$
  - o que é muito bom no caso do grafo denso.
- Agora, se o grafo tem custos negativos e é esparso
  - será que conseguimos um algoritmo com maior eficiência?

Recalculando custos das arestas

→ Somar um valor em cada aresta

Tentando nos livrar dos custos negativos de maneira inocente

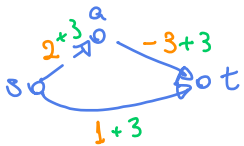
- Exemplo e contraexemplo



$P_1 = s \to a \to b \to t \quad c(P_1) = 4 + 9$

$P_2 = s \to c \to d \to t \quad c(P_2) = -2 + 9$

$P_3 = s \to c \to b \to t \quad c(P_3) = 8 + 9$



$P_1 = s \to a \to t \quad c(P_1) = -1 + 6 = 5$

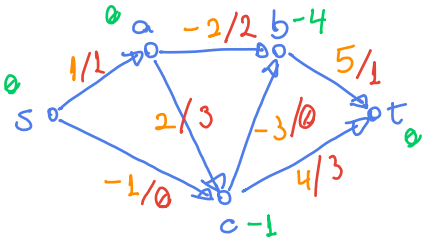
$P_2 = s \to t \quad c(P_2) = 1 + 3 = 4$

Técnica de recálculo de custos do algoritmo de Johnson

- Para e = (u, v) fazemos c'(e) = c(e) + p_u - p_v

$e = (u, v)$

$c'(e) = c(e) + p_u - p_v$



$P_1 = s \to a \to b \to t \quad c(P_1) = 4$

$P_2 = s \to a \to c \to b \to t \quad c(P_2) = 5$

$P_3 = s \to c \to t \quad c(P_3) = 3$

$P_4 = s \to c \to b \to t \quad c(P_4) = 1$

$c'(P_1) = 1 + 2 + 1 = 4$

$c'(P_2) = 1 + 3 + 0 + 1 = 5$

$c'(P_3) = 0 + 3 = 3$

$c'(P_4) = 0 + 0 + 1 = 1$

⊛ Será que os valores não mudam?

Propriedade 1) a mudança de custos não altera a ordem relativa de custo

- entre os caminhos com mesma origem e destino.

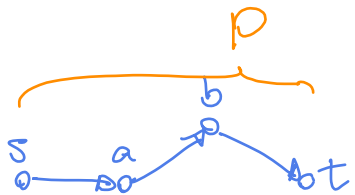
Demonstração: Tome um caminho qualquer P entre ‘s’ e ‘t’.

- Calculando o custo original de P temos
  - c(P) = sum_{e \in P} c(e)

$c(P) = \sum_{e \in P} c(e)$

- Calculando o custo modificado de P temos
  - c'(P) = sum_{e \in P} c'(e)
    = sum_{e=(u,v) \in P} (c(e) + p+u - p_v)
    = p_s - p_t + sum_{e=(u,v) \in P} c(e)
    = c(P) + p_s - p_t

$$c'(P) = \sum_{e \in P} c'(e) = \sum_{e=(u,v) \in P} \left(c(e) + p_u - p_v\right) = c(P) + p_s - p_t$$



$$c'(P) = c'(s,a) + c'(a,b) + c'(b,t)$$
$$= c(s,a) + p_s - \cancel{p_a} + c(a,b) + \cancel{p_a} - \cancel{p_b} + c(b,t) + \cancel{p_b} - p_t$$
$$= \sum_{e \in P} c(e) + p_s - p_t$$

onde a penúltima desigualdade vale pois, para cada vértice ‘v’

- interno do caminho P, o valor p_v aparece duas vezes no somatório,
  - uma vez com sinal positivo e outra vez com sinal negativo.

Note que, independente de qual seja o caminho P entre ‘s’ e ‘t’,

  - a diferença entre c(P) e c’(P) depende apenas de p_s e de p_t.

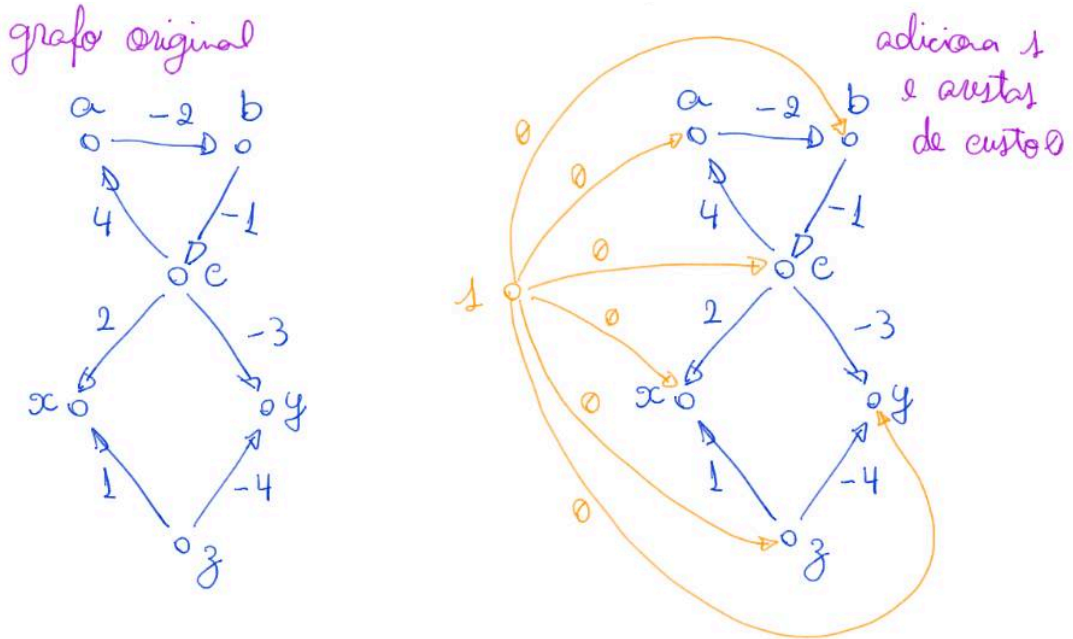
- Portanto, a mudança de custos não altera a ordem relativa de custo
  - entre os caminhos com mesma origem e destino.

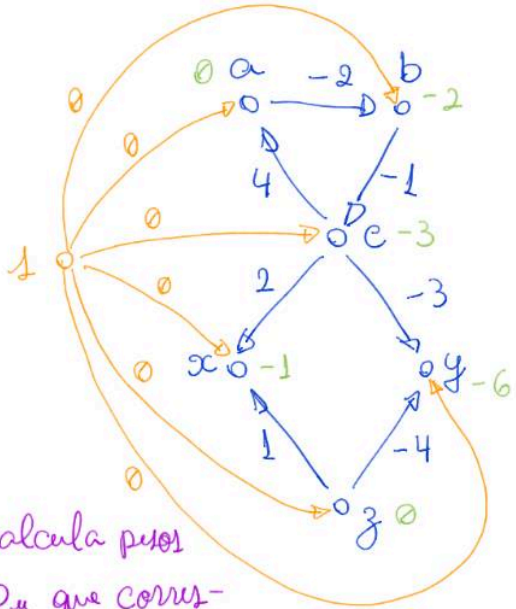
Será que existem esse pesos/potenciais especiais para os vértices

- que eliminam os custos negativos?
    - Se sim, como encontrá-los?
- Solução candidata:
    - o custo do caminho mais negativo que termina em cada vértice.

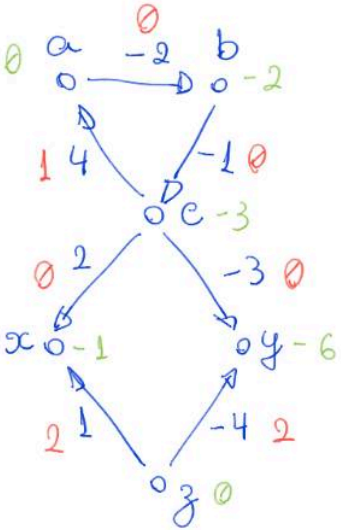
Exemplo:

desenhar





calcula pesos p_u que correspondem ao caminho mínimo de s a u



calcula $c'(u,v) = c(u,v) + p_u - p_v$

⊛ Eliminou os pesos negativos !!!

Será coincidência?

Será que esse procedimento sempre funciona?

Propriedade 2) o custo do caminho mais negativo que termina em cada vértice

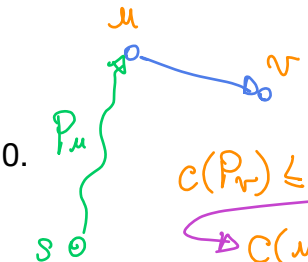
- é o peso/potencial que torna todas as arestas não negativas.

Demonstração: note que o valor p_u para um vértice ‘u’ qualquer

  - corresponde ao caminho mínimo de ‘s’ até ‘u’.
- Como ‘s’ tem uma aresta de custo zero entrando em todos os vértices,
  - o custo dos caminhos mínimos originados em ‘s’ será sempre <= 0. ≤ 0

Considere uma aresta (u, v) qualquer.

- Um caminho possível para ir de ‘s’ até ‘v’ é
  - ir de ‘s’ até ‘u’ e então usar a aresta (u, v).
- Portanto
  - p_v <= p_u + c(u, v) → c(u, v) + p_u - p_v >= 0.
- Mas isso conclui a demonstração, pois
  - c’(u, v) = c(u, v) + p_u - p_v >= 0.



cam. mais custo em G' de s até u

$c(P_v) \leq c(P_u) + c(u,v)$

$c(u,v) \geq p_v - p_u$

Como tomamos uma aresta (u, v) qualquer, mostramos que

- a mudança de custos torna não negativos os custos de todas as arestas.

Portanto,

$c'(u,v) = c(u,v) + p_u - p_v$

$\geq \cancel{p_v} - \cancel{p_u} + \cancel{p_u} - \cancel{p_v} = 0$

Algoritmo de Johnson

algJohnson(G=(V,E), c):

1 - construa G’ adicionando um vértice ‘s’
- e uma aresta com custo ‘0’ indo de ‘s’ até cada ‘v’ em V.

2 - execute o algoritmo de Bellman-Ford em G’ com custos ‘c’ a partir de ‘s’.
- Para cada vértice ‘v’ em V seja
  - Pv = caminho mínimo encontrado pelo algoritmo de Bellman-Ford,
  - e p_v = c(Pv)

3 - para cada aresta (u, v) em E
- calcule custos c’(u, v) = c(u, v) + p_u - p_v

4 - para cada vértice ‘v’ em V execute o algoritmo de Dijkstra em G
- com custos c’ a partir de ‘v’.

Sejam d’(u, v) as distâncias obtidas
- na diversas execuções do algoritmo de Dijkstra.

5 - Para cada par de vértices u, v em V calcule d(u, v) = d'(u, v) - p_u + p_v

Eficiência:

1 - Theta(n)

2 - Theta(n m)

3 - Theta(m)

4 - n * Theta(m log n) = Theta(n m log n)

5 - Theta(n^2)

como n = O(m), o tempo do algoritmo é dominado por Theta (n m log n),

- que supera o algoritmo de Floyd-Warshall em grafos esparsos.

Corretude do algoritmo deriva das propriedades

- 1) a mudança de custos não altera a ordem relativa
    - entre os caminhos com mesma origem e destino,
- e 2) o custo do caminho mais negativo que termina em cada vértice
    - é o peso/potencial que torna todas as arestas não negativas.
- O passo 5 decorre de c'(P) = c(P) + p_s - p_t
    - para qualquer caminho P começado em s e terminado em t.