

Algoritmos

Pedro Hokama

- [c/rs] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest, Ronald L. Rivest e Clifford Stein.
- [timr] Algorithms Illuminated Series, Tim Roughgarden
Apresentação Baseada:
 - Stanford Algorithms
<https://www.youtube.com/playlist?list=PLXFMmk03Dt7Q0xr1PIAriY5623cKiH7V>
<https://www.youtube.com/playlist?list=PLXFMmk03Dt5EMI2s2WQBLSz17A5HEK6>
 - Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
 - Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420Qualquer erro é de minha responsabilidade.

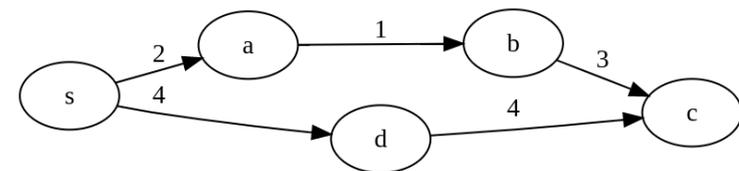
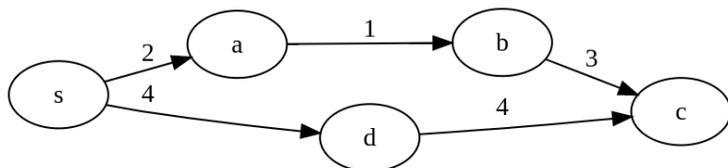
1 / 54

2 / 54

Caminhos mais Curtos de Única Fonte

Problema do Caminho Mínimo de Única fonte

Dado um grafo direcionado $G = (V, A)$ com m arcos e n vértices, em que cada arco $a \in A$ tem um custo c_a , e um vértice fonte $s \in V$. Desejamos calcular para cada vértice $v \in V$ o valor $D(v)$ do $s - v$ caminho mais curto.

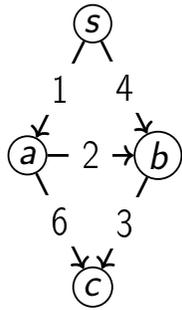


- No exemplo acima, $D(c) = 6$.
- Já vimos um algoritmo para esse problema quando os pesos são não negativos.

3 / 54

4 / 54

Algoritmo de Dijkstra



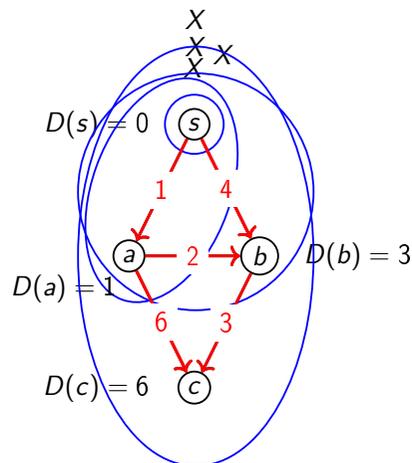
5 / 54

Algoritmo de Dijkstra

- Manter um conjunto X com os vértices já processados.
- Considerar arcos com início em X e final em $V \setminus X$
- Escolher o arco (u, v) que minimiza $D(u) + c_{(u,v)}$, ou seja, o arco que minimiza o caminho para um vértice de $V \setminus X$.
- Note que não existe caminho menor para v .
- Computa $D(v)$ e inclui v em X .
- Repita até que todos os vértices estejam em X , ou não tenha nenhum arco.

6 / 54

Algoritmo de Dijkstra



7 / 54

- O Algoritmo de Dijkstra executa em tempo $O(m \log n)$.
- O Dijkstra só funciona corretamente se os comprimentos forem **positivos**, ou seja, $c_a \geq 0, \forall a \in A$.
- Quando os custos podem ser negativos precisamos de outro algoritmo.
- Os custos podem ser negativos por exemplo, se ao atravessamos um arco eu receba algo por essa viagem.

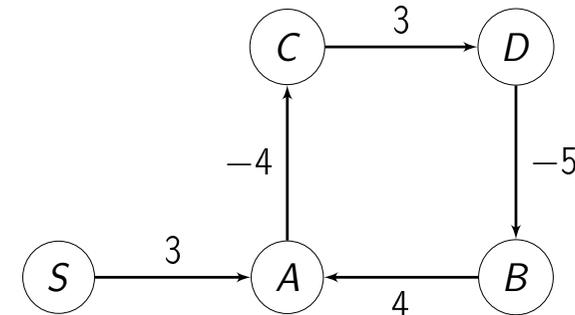
8 / 54

- Ou ainda se o grafo pode tratar de decisões financeiras e um arco significar a venda de uma ação.
- Além disso mesmo com custos positivos o Dijkstra não é bem distribuído (no sentido de paralelizável, o que é relevante para protocolos de internet)
- Opção: o Algoritmo de Bellman-Ford (que apesar de mais antigo que a ARPAnet forma a base para os protocolos de roteamento de internet modernos (possivelmente com muitos detalhes de implementação))

9 / 54

Ciclos Negativos

- Como definir caminhos mais curtos quando G tem um **ciclo** de custo negativo.



10 / 54

- Se **ciclos são permitidos** então o problema simplesmente **não tem solução**. Você pode ficar ciclando indefinidamente obtendo ciclos com custo cada vez mais baixo.
- Se **ciclos são proibidos**, não é conhecido nenhum algoritmo **eficiente** para encontrar o caminho mais curto.
- Por hora vamos considerar que o grafo não tem ciclos negativos, e depois veremos que o algoritmo de Bellman-Ford na verdade é capaz de detectar tais ciclos.

11 / 54

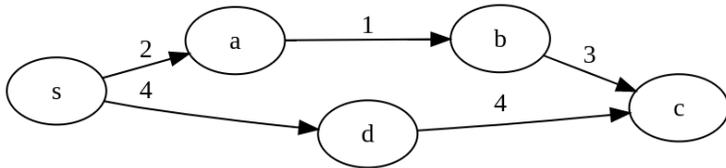
Encontrando uma subestrutura ótima

- Note que como não existem ciclos negativos, qualquer caminho mais curto entre a fonte e qualquer outro vértice passa por no máximo $n - 1$ arcos.
- Ao contrário dos Grafo Caminhos, não é óbvio que podemos remover algum vértice para formar um problema menor.
- e ao contrário do problema da mochila não é óbvio dizer que podemos ordenar os vértices, já que um vértice i pode ser necessário para o caminho até o vértice $i - 1$.

12 / 54

Subestrutura ótima

- A ideia então será restringir o número de arcos que podem ser usados em um caminho.



13 / 54

- Seja $G = (V, A)$ um grafo direcionado com custo c_a nos arcos e um vértice fonte s . [podem conter ciclos negativos]

14 / 54

Subestrutura ótima

- Para todo $v \in V$, $i \in \{1, 2, \dots\}$, seja P o menor $s - v$ caminho com no máximo i arcos. [pode conter ciclos]
- Caso 1: se P tem $\leq (i - 1)$ arcos, então ele já é um caminho mais curto com $\leq (i - 1)$ arcos. (então já é uma solução para um subproblema menor, isso é análogo a não pegar um item na mochila)
- Caso 2: se P tem i arcos, seja o último arco (w, v) , então P' é o $s - w$ caminho mais curto com $\leq (i - 1)$ arcos.

15 / 54

Quantos subproblemas candidatos eu preciso olhar para decidir qual o menor caminho para chegar em v usando i arcos? (suponha que eu já calculei os subproblemas menores)

- a 2
- b $1 + \text{grau de entrada de } v$
- c $n - 1$
- d n

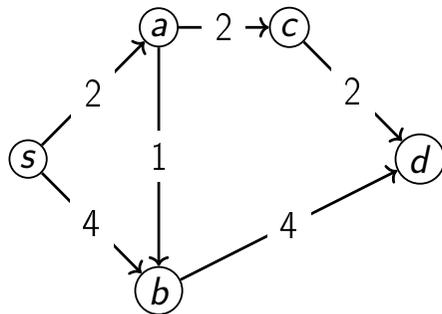
16 / 54

Recorrência

- Seja $L_{i,v}$ o custo do menor caminho de s à v que usa no máximo i arcos. $+\infty$ se não existe.
- Para todo $v \in V$ e $i \in \{1, 2, 3, \dots, n - 1\}$

$$L_{i,v} = \min \begin{cases} L_{(i-1),v} \\ \min_{(w,v) \in A} \{L_{(i-1),w} + c_{wv}\} \end{cases}$$

17 / 54



19 / 54

Algoritmo Bellman-Ford

- Considerando que não temos ciclos negativos, os caminhos tem no máximo $n - 1$ arcos.
- Vamos preencher um vetor bidimensional A , em que $A[i, v]$ vai ser o custo do menor caminho de s até v usando no máximo v arestas.

18 / 54

Algoritmo 1: Bellman-Ford(G, s)

Entrada: Um conjunto Grafo, e um vértice fonte s

Saída: O valor do menor caminho de s para todos os outros vértices

- 1 $A[0, s] = 0$; $A[0, v] = +\infty$ para todo $v \neq s$;
 - 2 **para** $i = 1, 2, \dots, n - 1$ **faça**
 - 3 **para todo** $v \in V$ **faça**
 - 4 $A[i, v] = \min\{A[i - 1, v]; \min_{w:(w,v) \in A} \{A[i - 1, w] + c_{wv}\}\}$;
 - 5 devolva $A[n - 1, *]$;
-

20 / 54

Algoritmo Bellman-Ford

- O tempo de execução do Bellman-Ford é $O(mn)$
- Você pode parar o algoritmo se em uma iteração a distância para nenhum vértice mudar.
- Podemos detectar ciclos negativos rodando uma iteração a mais $i = n$, se encontrarmos algum caminho mais curto significa que existe um ciclo de custo negativo.

21 / 54

Algoritmo Bellman-Ford

- Podemos reconstruir os caminhos (ou identificar o ciclo negativo) analisando a tabela.
- Se só estivermos interessados no valor da solução ótima, podemos economizar espaço de armazenamento.
- Também podemos guardar um vetor $P[n]$ de predecessores de cada vértice.

22 / 54

Caminhos Mínimos

- Vimos dois algoritmos para encontrar o caminho mais curto entre um determinado vértice fonte s e todos os outros vértices.
- Dijkstra e Bellman-Ford.
- Mas com frequência estaremos interessados em encontrar o caminho mínimo entre quaisquer dois vértices.

23 / 54

Caminhos Mínimos

Caminho Mínimo entre Todos os Pares de Vértices

Dado um grafo direcionado $G = (V, A)$, com n vértices e m arcos, em que cada arco $a \in A$ tem um custo c_a (pode ser negativo). Desejamos encontrar o custo mínimo para ir de u a v para todos os pares $u, v \in V$ ou detectar se houver um ciclo de custo negativo.

24 / 54

Caminho Mínimo entre Todos os Pares de Vértices

- Podemos resolver esse problema, chamando o algoritmo de Caminho Mínimo de Única fonte algumas vezes. Quantas execuções seriam necessárias?

- a 1
- b $n - 1$
- c n
- d n^2

25 / 54

Caminho Mínimo entre Todos os Pares de Vértices

- Se o grafo não tem custos negativos, poderíamos usar o Algoritmo de Dijkstra, e obter uma complexidade de

$$O(nm \log n) \begin{cases} O(n^2 \log n) & \text{em grafos esparços} \\ O(n^3 \log n) & \text{em grafos densos} \end{cases}$$

26 / 54

Caminho Mínimo entre Todos os Pares de Vértices

- De fato não sabemos se algum algoritmo pode ser mais rápido que $O(n^3)$, uma vez que precisa ser calculado para todos os pares, e podemos imaginar que um trabalho linear por par seria necessário. (Mas lembre do algoritmo de Strassen para multiplicação de matrizes que era subcúbico).

27 / 54

Caminho Mínimo entre Todos os Pares de Vértices

- Mas se o grafo tem custos negativos, poderíamos usar o Bellman-Ford (executado n vezes):

$$O(n^2 m) \begin{cases} O(n^3) & \text{em grafos esparços} \\ O(n^4) & \text{em grafos densos} \end{cases}$$

28 / 54

- Parece ser uma complexidade muito grande. De fato o algoritmo de Floyd-Warshall que veremos executa em $O(n^3)$ independente do número de arcos, e funciona mesmo com os custos negativos.
- Portanto é pelo menos tão bom quanto n execuções do Bellman-Ford. Muito melhor para grafos densos.
- Para grafos sem custos negativos, executar o Dijkstra n vezes ainda é melhor, mas para grafos densos ambos são competitivos.

29 / 54

Caminho Mínimo entre Todos os Pares de Vértices

- Esse problema pode aparecer por exemplo se queremos saber se uma relação tem um fechamento transitivo (se existe um caminho orientado entre qualquer par de vértices).
- Assim como o Bellman-Ford, adaptações do Floyd-Warshall podem resolver vários problemas práticos.

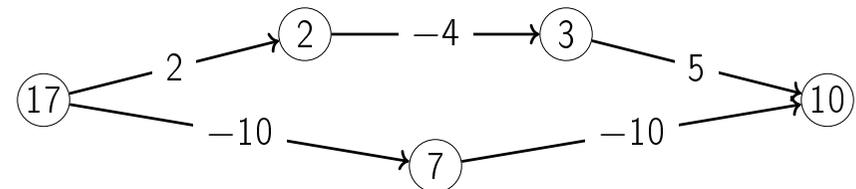
30 / 54

Ideias - Subestrutura Ótima

- Como vimos no Bellman-Ford, encontrar qual subproblema pode ser resolvido não é tão simples. Acabamos por perceber que o subproblema estava no número de arcos que poderíamos utilizar.
- **Idéia:** Vamos limitar o número de vértices que vamos usar. E também limitar quais vértices podemos usar.
- Considere uma ordenação arbitrária de $V = 1, 2, 3, \dots, n$, e seja o prefixo $V^{(k)} = \{1, \dots, k\}$.

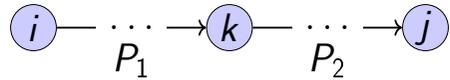
31 / 54

- Suponha que G não tem ciclos negativos. Fixado uma origem $i \in V$, e um destino $j \in V$ e $k \in \{1, 2, \dots, n\}$. Seja P o caminho de custo mínimo (portanto sem ciclos) entre i e j com todos os nós internos em $V^{(k)}$.
- Por exemplo, seja $i = 17$, $j = 10$ e $k = 5$.



32 / 54

- Considere que $k = n$.
- Caso 1: Se k não está em P , então P é um caminho de custo mínimo com todos os vértices internos em $V^{(k-1)}$.
- Caso 2: Se k está em P então podemos dividir P em dois pedaços.

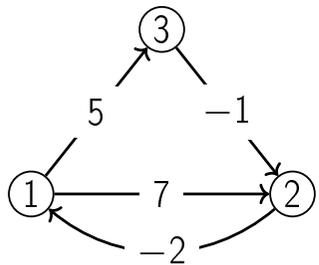


- P_1 é o caminho de custo mínimo entre i e k com todos os nós internos em $V^{(k-1)}$ e P_2 é o caminho de custo mínimo entre k e j com todos os nós internos em $V^{(k-1)}$

33 / 54

- Seja S um vetor tridimensional indexado em i, j e k .
- $S[i, j, k]$ vai representar o custo do caminho mínimo entre i e j em que todos os nós internos estão em $\{1, 2, \dots, k\}$ (ou $+\infty$ se tal caminho não existir)
- Qual deve ser $S[i, j, 0]$ se (I) $i = j$, (II) $(i, j) \in A$, (III) $i \neq j$ e $(i, j) \notin A$.
 - a 0, 0, e $+\infty$
 - b 0, c_{ij} e c_{ij}
 - c 0, c_{ij} e $+\infty$
 - d $+\infty$, c_{ij} e $+\infty$

34 / 54



$k = 0$ destino

	1	2	3
origem 1			
origem 2			
origem 3			

$k = 0$ destino

	1	2	3
origem 1	0	7	5
origem 2	-2	0	∞
origem 3	∞	-1	0

$k = 1$ destino

	1	2	3
origem 1			
origem 2			
origem 3			

35 / 54

36 / 54

$k = 1$ destino

	1	2	3
origem 1	0	7	5
2	-2	0	3
3	∞	-1	0

$k = 2$ destino

	1	2	3
origem 1			
2			
3			

$k = 2$ destino

	1	2	3
origem 1	0	7	5
2	-2	0	3
3	-3	-1	0

$k = 3$ destino

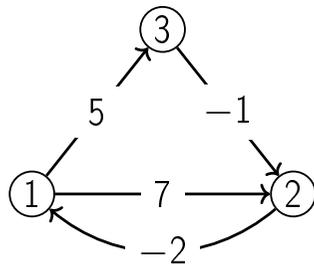
	1	2	3
origem 1			
2			
3			

37 / 54

38 / 54

$k = 3$ destino

	1	2	3
origem 1	0	4	5
2	-2	0	3
3	-3	-1	0



Algoritmo 2: Floyd-Warshall(G)

Entrada: Um grafo caminho G

Saída: Os custos dos caminhos mínimos entre todos os pares de vértices

```

1 para todo  $i \in V$  faça
2   para todo  $j \in V$  faça
3      $S[i, j, 0] = \begin{cases} 0 & \text{se } i = j \\ c_{ij} & \text{se } (i, j) \in A \\ +\infty & \text{se } i \neq j \text{ e } (i, j) \notin A \end{cases}$ 
4 para  $k \in \{1, \dots, n\}$  faça
5   para todo  $i \in V$  faça
6     para todo  $j \in V$  faça
7        $S[i, j, k] = \min \begin{cases} S[i, j, k - 1] \\ S[i, k, k - 1] + S[k, j, k - 1] \end{cases}$ 
  
```

39 / 54

40 / 54

- Corretude: Pela subestrutura ótima e indução
- Complexidade: $O(1)$ por subproblema, total de $O(n^3)$.
- E se o grafo tiver um ciclo de custo negativo?
- Nesse caso, no final do algoritmo você terá um $S[i, i, n] < 0$ para pelo menos um $i \in V$
- E se precisamos reconstruir o caminho entre i e j ?
- Nesse caso podemos guardar um vetor bidimensional auxiliar $T[i, j]$ que vai guardar o vértice interno com o índice mais alto no caminho de i até j .

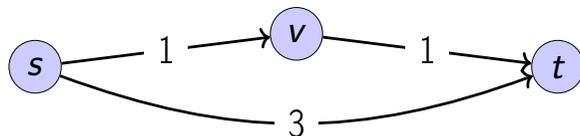
41 / 54

Algoritmo de Johnson

- Relembrando que podemos resolver o problema com n chamadas ao algoritmo de caminhos mínimos de única fonte:
 - ▶ Custos não negativos: $O(mn \log n)$ com o Dijkstra
 - ▶ Caso geral: $O(mn^2)$ com o Bellman-Ford
- Mas eu ainda quero usar o Dijkstra mesmo com pesos negativos. Será possível?
- O algoritmo de Johnson:
 - ▶ Faz 1 chamada ao Bellman-Ford
 - ▶ Faz n chamadas ao Dijkstra
- Portanto $O(mn) + O(nm \log n) = O(nm \log n)$

42 / 54

- Podemos só somar um valor em todas as arestas para tornar todos os custos positivos? Não! Suponha no grafo abaixo que somamos 2 em todas os arcos, perceba que o menor caminho muda. Só funcionaria se todos os caminhos tivessem o mesmo número de arestas.



- Nem tudo está perdido, podemos sim usar um rebalanceamento para se livrar dos pesos negativos.

43 / 54

- Seja $G = (V, A)$ um grafo direcionado com custos c_a nos arcos, podendo ser negativos. Fixe um número real p_v para cada vértice $v \in V$.
- Para cada arco $a = (u, v)$ de A , faça $c'_a = c_a + p_u - p_v$

$$\begin{array}{c}
 \textcircled{u} \text{ --- } 2 \text{ --- } \textcircled{v} \\
 p_u = -4 \qquad p_v = -3
 \end{array}$$

44 / 54

- Se o $s - t$ caminho P tem custo L com os custos originais, qual é o custo L' de P com os custos modificados?

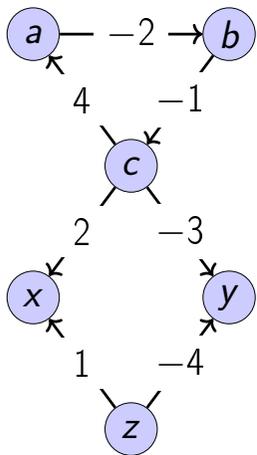
- a L
- b $L + p_s + P_t$
- c $L + p_s - P_t$
- d $L - p_s + P_t$

$$\begin{aligned}
 L' &= \sum_{a \in P} c'_a \\
 &= \sum_{a=(u,v) \in P} c_a + p_u - p_v \\
 &= \left(\sum_{a \in P} c_a \right) + p_s - p_t = L + p_s - p_t
 \end{aligned}$$

45 / 54

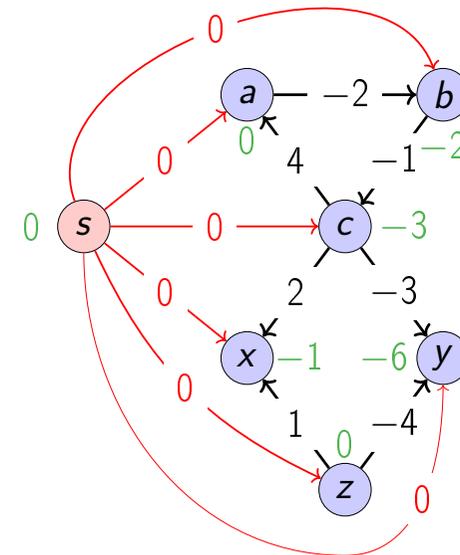
- Note que dessa forma qualquer caminho foi alterado exatamente pelo mesmo custo.
- Dessa forma os caminhos mínimos foram preservados (obviamente não o seu custo, mas é fácil de obtê-lo)
- E se encontramos valores p_v de forma que todos os arcos fiquem com custos não-negativos?
- Como encontrar esses valores? Alias, será que tais valores existem? 🤔
- Se o grafo não tiver ciclos negativos, SIM, esses valores existem!

46 / 54



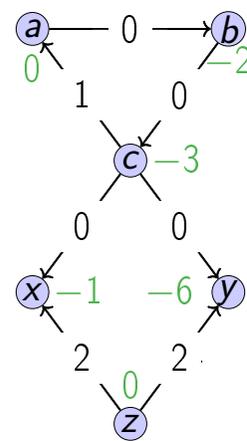
- A ideia consiste em executar um algoritmo para o problema do caminho mínimo de única fonte.
- Um problema: É preciso alcançar todos os vértices, no grafo ao lado qualquer vértice que você escolher para ser sua fonte não vai alcançar todos os outros.
- Truque, inserir um novo vértice fictício que se liga a todos os outros por um arco de custo 0.

47 / 54



48 / 54

- Note que adicionar s e os novos arcos não cria nenhum novo caminho para qualquer $u, v \in V$.
- como existem arcos com custo negativo, nos resta executar o Bellman-Ford (o que também detecta ciclos negativos)
- Esse são exatamente os valores que procuramos. p_v é o custo do $s - t$ caminho mínimo.



- Podemos então para cada arco $a = (u, v)$ calcular o novo custo $c'_a = c_a + p_u - p_v$
- Agora todos os custos ficaram não negativos 😎 (pelo menos para esse exemplo)
- Agora não precisamos mais usar o Bellman-Ford e podemos usar o Dijkstra!

Algoritmo 3: Johnson(G)

Entrada: Um grafo caminho G

Saída: Os custos dos caminhos mínimos entre todos os pares de vértices

- 1 Formar G' , adicionando à G um vértice s extra e um novo arco (s, v) para todo $v \in V$ com custo 0;
- 2 Executar o Bellman-Ford em G' com fonte s (se detectar ciclo negativo, terminar);
- 3 **para** $v \in V$ **faça** $p_v =$ custo do $s - t$ caminho mínimo em G' ;
- 4 **para** $(u, v) \in A$ **faça** $c'_{uv} = c_{uv} + p_u - p_v$;
- 5 **para** $v \in V$ **faça** Executar o Dijkstra em G com fonte em v e os custos $\{c'_a\}$, obtendo os custos $d'(u, v)$;
- 6 **para** cada par $u, v \in V$ **faça** $d(u, v) = d'(u, v) - p_u + p_v$;
- 7 devolva d ;

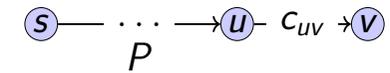
Tempo de execução:

$$O(n) + O(mn) + O(m) + O(nm \log n) + O(n^2) = O(mn \log n)$$

- Corretude: Como os caminhos de custo mínimo são preservados, se não houver custos negativos as execuções do Dijkstra os encontram corretamente, logo o algoritmo de Johnson funciona.
- Resta demonstrar que de fato os custos $\{c'_a\}$ são não negativos.

53 / 54

- Considere um arco (u, v) , por construção p_u é custo do caminho mínimo P de s a u , e p_v do caminho de s a v .



- Como $P + (u, v)$ é um possível caminho para v , certamente o caminho de custo mínimo tem um valor menor ou igual a esse. Ou seja

$$p_v \leq p_u + c_{uv}$$

$$0 \leq p_u + c_{uv} - p_v$$

$$c'_{uv} = p_u + c_{uv} - p_v \geq 0$$

54 / 54