Algoritmos de Menor Caminho

Disciplina: Laboratório de Programação II

Professor: Luciano Brum
Email: lucianobrum18@gmail.com
Site: https://sites.google.com/view/brumluciano

Assunto da aula de hoje:

Algoritmos de Menor Caminho

Tópicos

- Conceito Básico.
- Algoritmo de Bellman-Ford.
- Caminho mais curto de uma única origem em gao’s.
- Algoritmo de Dijkstra.
- Resumo.

Conceitos Básicos

- Problema 1: um motorista deseja a rota mais curta possível do RJ a SP.

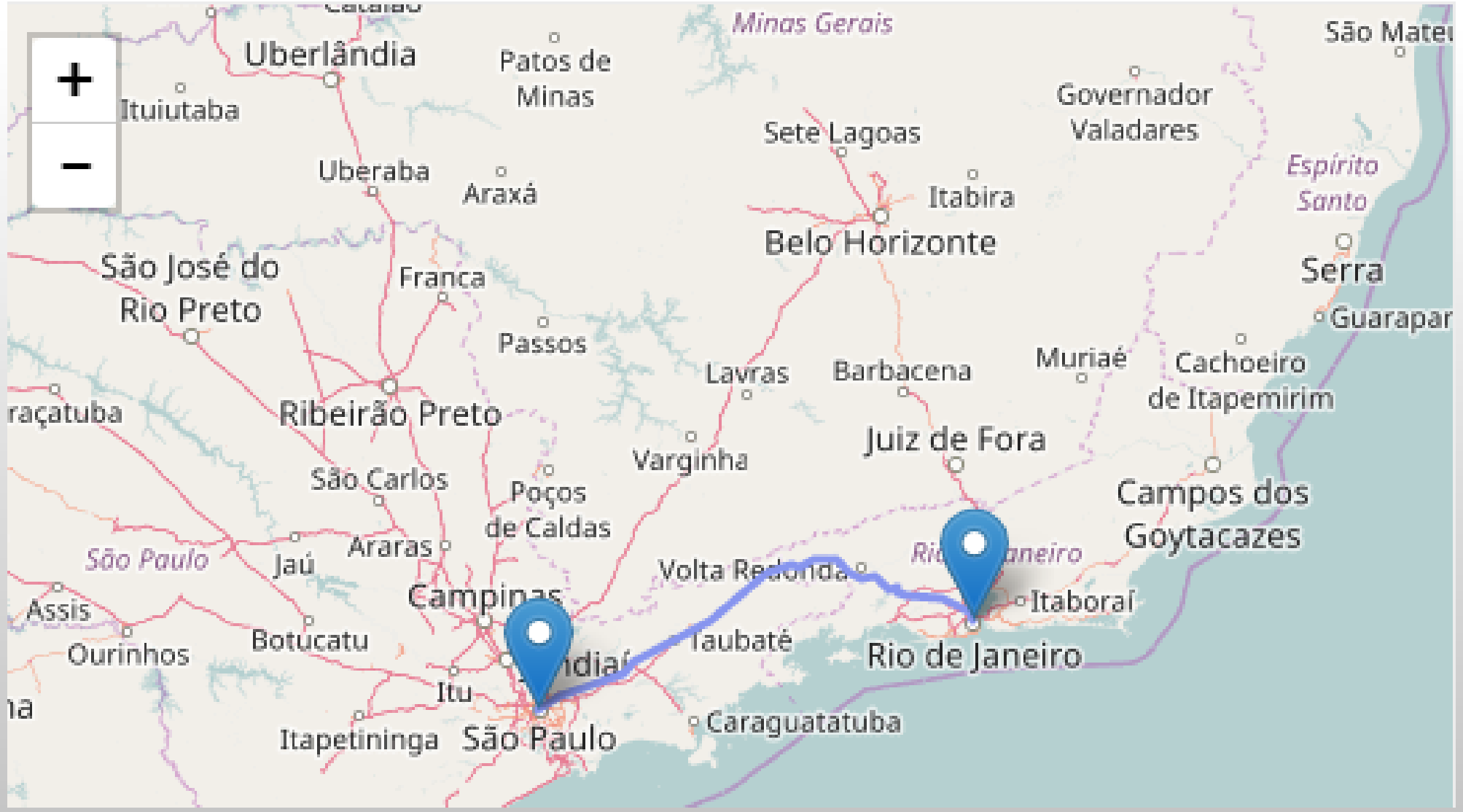
- Dado um mapa rodoviário do Brasil, no qual a distância entre cada par de cidades adjacentes é conhecida, como determina a rota mais curta?

Conceitos Básicos

- Solução: enumerar todas rotas existentes que levam o motorista de RJ a SP, somar todos os caminhos destas rotas, e por fim, selecionar a de caminho mínimo.

- Porque essa não seria uma boa solução?

Conceitos Básicos



Conceitos Básicos

- Problema 2: como determinar a rota mais curta de tráfego de informações entre dois roteadores?

- Mesmo problema !!

Conceitos Básicos

- É possível modelar esses problemas com grafos orientados ponderados G = (V,E), com uma função peso w : E ->R mapeando arestas para valores de pesos reais.

- O peso do caminho p = <*v0,v1,...,vk*> é o somatório dos pesos de suas arestas.

- $W(p) = \sum_{i=1}^{n} w(v_{i-1}, v_i)$

Conceitos Básicos

- Definimos o caminho mais curto de ***u*** até ***v*** por:
  - $\varrho(u,v) = \min (w(p): u \xrightarrow{p} v)$, se existe um caminho de u até v.
  - $\varrho(u,v) = \infty$, caso contrário.
- No problema da menor rota entre SP e RJ, podemos modelar as cidades como vértices, as estradas que conectam cada par de cidades como arestas, e a distância representa o peso dessas arestas.

Conceitos Básicos

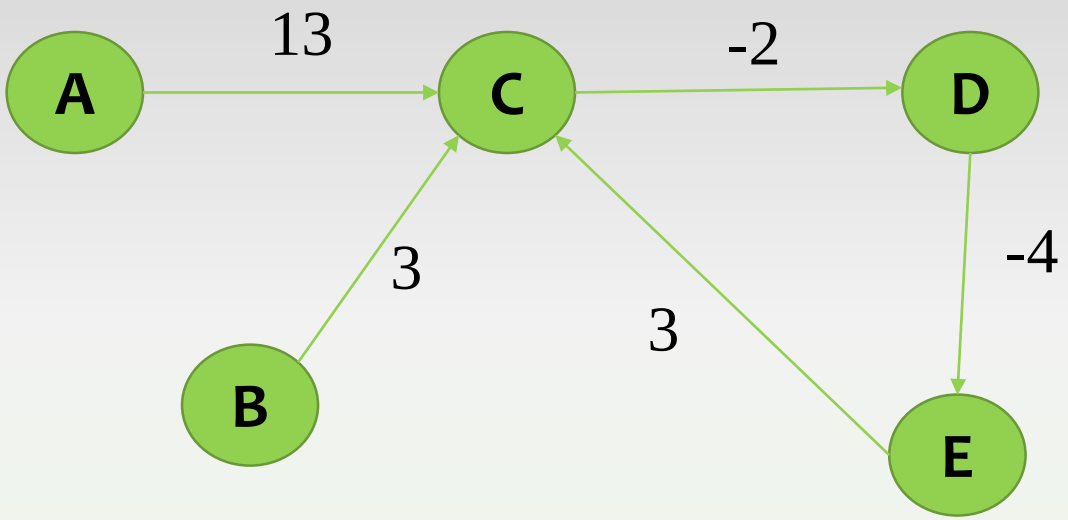
- As arestas podem representar tempo, custos, penalidades, distâncias, perdas ou qualquer outra quantidade que se acumule ao longo de um caminho e desejamos minimizar.

Conceitos Básicos

- Arestas de peso negativo: quando temos ciclos com arestas de peso negativo, o problema do menor caminho não fica bem definido.

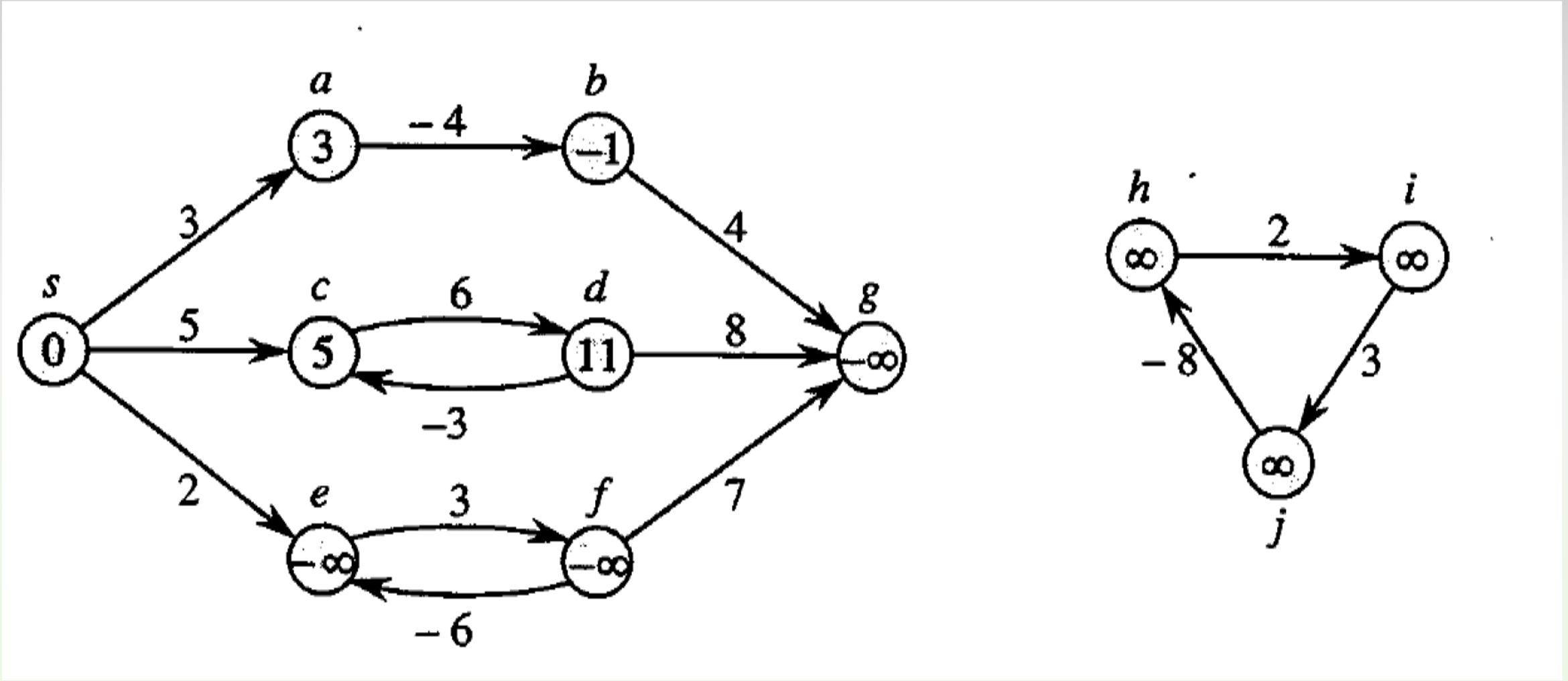
- Porquê?

Conceitos Básicos



- Qual menor caminho de A até D?

Conceitos Básicos



Conceitos Básicos

- Menor caminho de ***s*** até ***a***: ϱ(s,a) = w(s,a) = 3
- Menor caminho de ***s*** até ***b***: ϱ(s,b) = w(s,a) + w(a,b) = -1
- Menor caminho de ***s*** até ***c***: <s,c>, <s,c,d,c>, <s,c,d,c,d,c>, <s,c,d,c,d,c,d,c>.
- O ciclo formado é positivo, portanto o menor caminho é <s,c>.

Conceitos Básicos

- Menor caminho de ***s*** até ***d***: <s,c,d>, <s,c,d,c,d>, <s,c,d,c,d,c,d>...
- O ciclo formado é positivo, portanto o menor caminho é <s,c,d>.

- Menor caminho de ***s*** até ***e***: <s,e>, <s,e,f,e>, <s,e,f,e,f,e>, <s,e,f,e,f,e,f,e>.

- Qual é o menor caminho?

- Mesmo problema os caminhos de ***s*** até ***f*** e ***g***.
- Portanto, o menor caminho de ***s*** até ***e, f*** e ***g*** é $-\infty$ (ciclo negativo).

Conceitos Básicos

- Menor caminho de ***s*** até ***h, i e j***: Não há caminho.

- Mesmo estes sendo um ciclo negativo, o caminho até estes vértices é $-\infty$.

Conceitos Básicos

- Existem diversos algoritmos para o cálculo do caminho mínimo:
  - ***Bellman-Ford***: Resolve o problema para grafos com um vértice-fonte e arestas que podem ter pesos negativos.
  - ***Dijkstra***: Resolve o problema com um vértice-fonte em grafos cujas arestas tenham peso ≥ a zero.
  - ***A****: um algoritmo heurístico que calcula o caminho mínimo com um vértice-fonte.
  - ***Floyd-Marshall***: Determina a distância entre todos os pares de vértices de um grafo.
  - ***Johnson***: Determina a distância entre todos os pares de vértices de um grafo, pode ser mais veloz que o algoritmo de Floyd-Warshall em grafos esparsos

Tópicos

- Conceito Básico.
- Algoritmo de Bellman-Ford.
- Caminho mais curto de uma única origem em gao's.
- Algoritmo de Dijkstra.
- Resumo.

Bellman-Ford

- O algoritmo de Bellman-Ford, criado por Richard Bellman e Lester Ford Jr., computa, para um dado dígrafo (grafo orientado) G com arestas ponderadas, o menor caminho de um nodo de origem s até cada um dos outros nodos de G.

  - Este algoritmo, ao contrário do largamente conhecido algoritmo de Dijkstra, não impõe nenhuma restrição sobre o sinal do peso das arestas.

  - Solução mais genérica.

Bellman-Ford

- A existência e cálculo do caminho mais curto são garantidos caso não haja a presença de ciclos negativos durante o caminho da origem até um nodo v do grafo.
- Ciclo negativo: todo caminho fechado pertencente a G tal que a soma de seus pesos é negativa.
  - De fato, caso seja necessário passar por um ciclo negativo C para atingir um nodo, na i-ésima vez que completarmos C estaremos realizando um caminho menor que na (i-1)-ésima vez, o que caracteriza a inexistência de um menor caminho.
- No entanto, o algoritmo de Bellman-Ford é capaz de indicar a presença de tais ciclos, o que nos mostra uma segunda utilidade do mesmo.

Bellman-Ford

- Algumas das aplicações deste algoritmo são:
  - Protocolos de Roteamento Vetor-Distância (RIP e OSPF).
  - Problema "Triangular Arbitrage“.

Bellman-Ford

- O último problema citado é bastante interessante e útil para o ramo de economia e investimentos.

- A ideia é sabermos se, dadas, no mínimo, três moedas diferentes (e.g. Reais, Dólares e Euros) e as taxas de conversão entre elas, há como realizarmos uma sequência de câmbio da moeda original para as outras tal que, na hora em que voltarmos para moeda original, teremos uma quantia maior que a inicial.

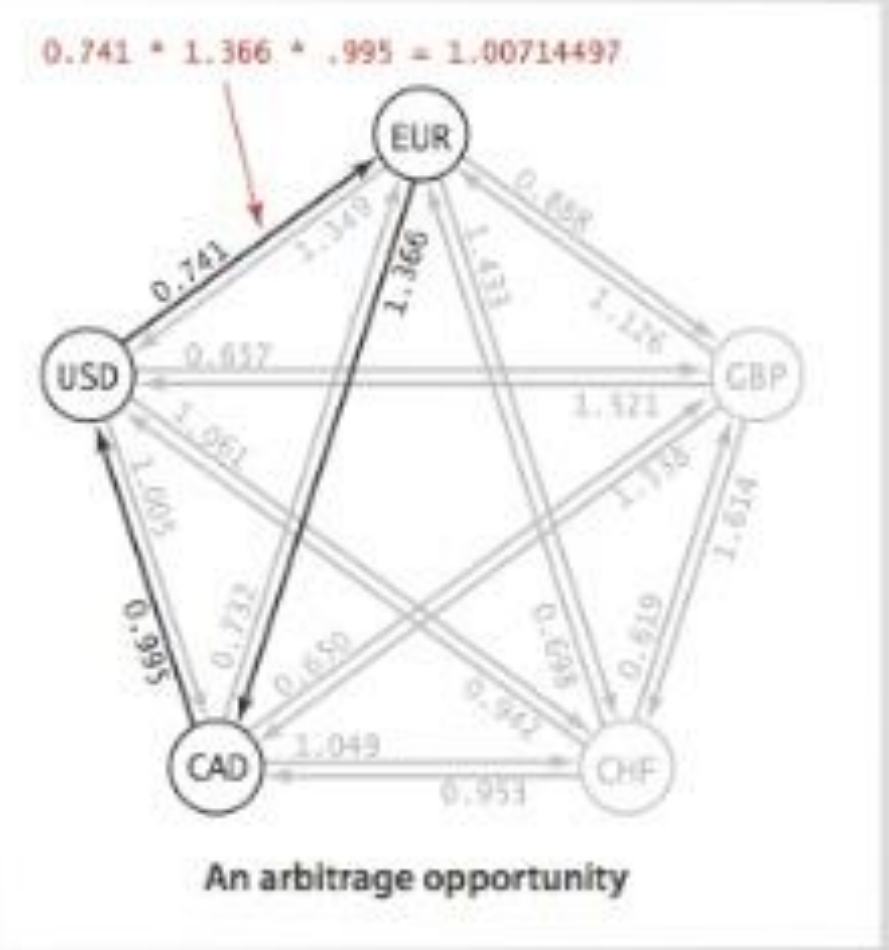
- Esta aplicação vale-se da presença de um ciclo negativo no grafo que representa o problema

1.4 Exemplo (Triangular Arbitrage):

- Com 1.000 dólares americanos podemos comprar 1.000 × 0,741 = 741 euros, então podemos comprar 741 × 1,366 = 1.012,206 dólares canadenses, e finalmente, 1.012,206 × 0,995 = 1.007,14497 dólares americanos com nossos dólares canadenses, obtendo 7,14497 de lucro

- Se trocarmos os pesos das arestas pelo seu respectivo log, verificamos a existência de um ciclo negativo

- Portanto o problema pode ser modelado pela busca de um ciclo negativo no grafo e podemos utilizar o algorimo de Bellman-Ford para isso



An arbitrage opportunity

Relaxamento

Relaxamento

- Os algoritmos de Bellman-Ford e Dijsktra utilizam a técnica de relaxamento.

- Para cada vértice v, mantemos um atributo d[v], que é a **estimativa de caminho mais curto** da origem s até v.

- Utiliza-se o procedimento INITIALIZE-SINGLE-SOURCE de tempo $\Theta(V)$ para inicializar o atributo d(v) e o predecessor de cada vértice.

Relaxamento

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for cada vértice v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

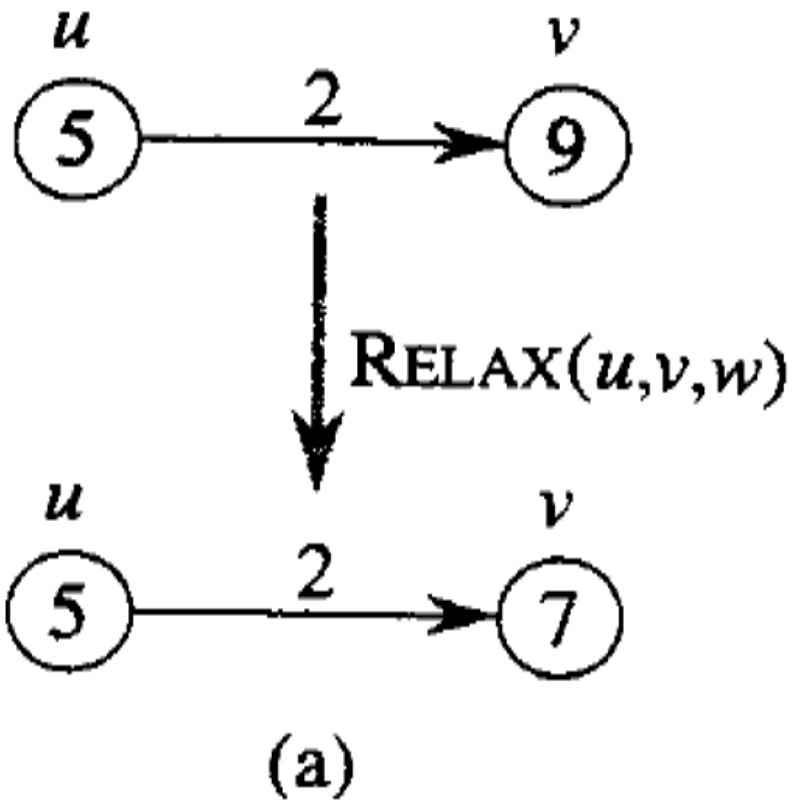
- Após a inicialização: π(v) <- NULL para todo v Є V, d[s] = 0 e d[v] = ∞ v Є V - {S}

Relaxamento

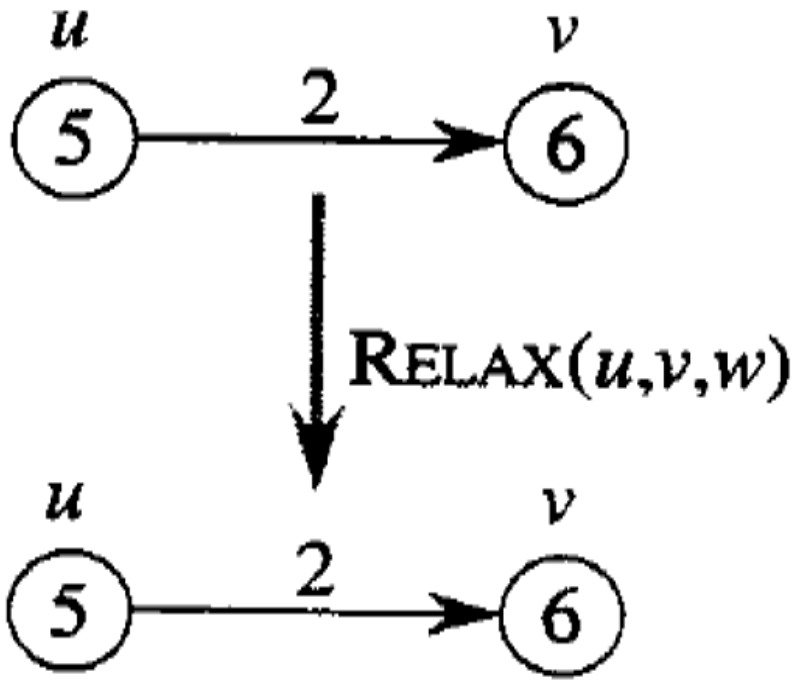
- O processo de relaxar uma aresta (u,v) consiste em testar se podemos melhorar o caminho mais curto para v encontrado até o momento pela passagem através de u e, nesse caso, atualizar d[v] e π[v].

- Uma etapa de relaxamento pode diminuir o valor da estimativa de caminho mais curto d[v] e atualizar o campo predecessor de v, π[v].

Relaxamento



(a)

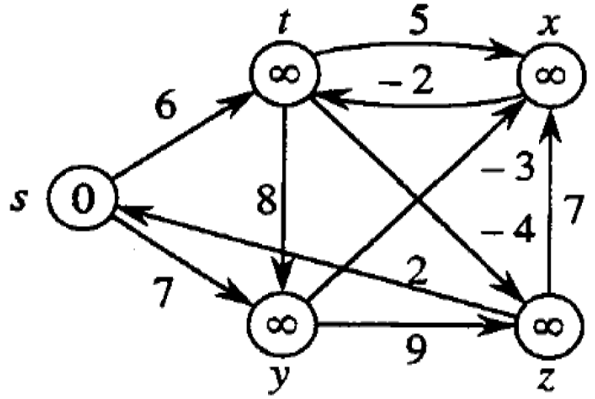


(b)

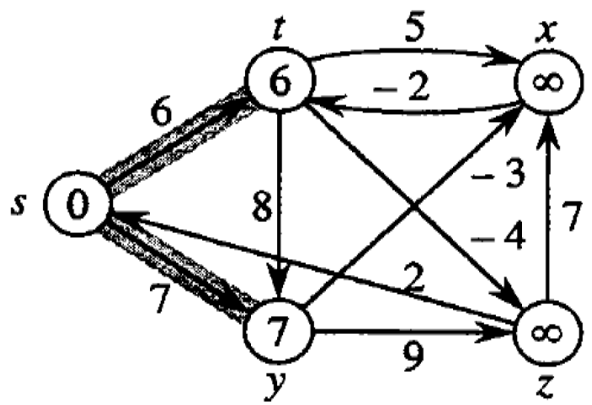
Relaxamento

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2     then d[v] ← d[u] + w(u, v)
3          π[v] ← u
```

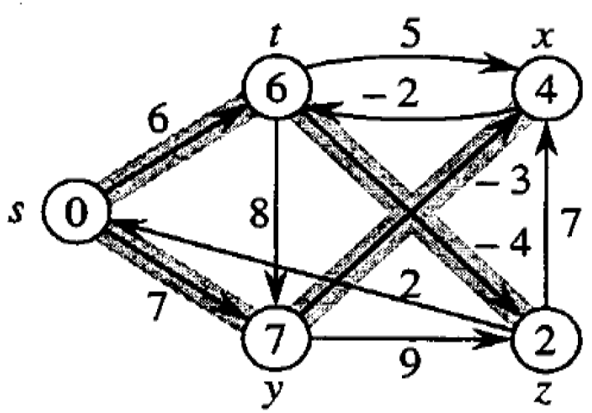
Bellman-Ford



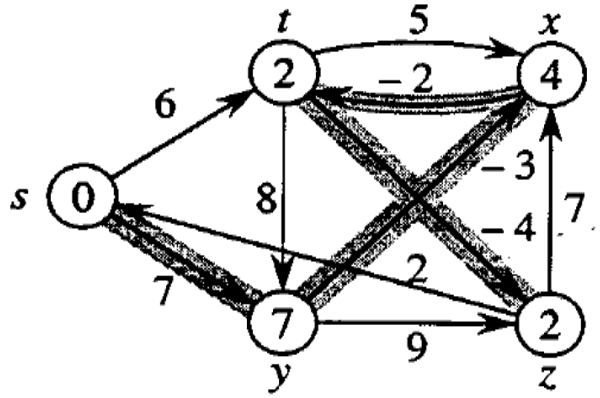
(a)



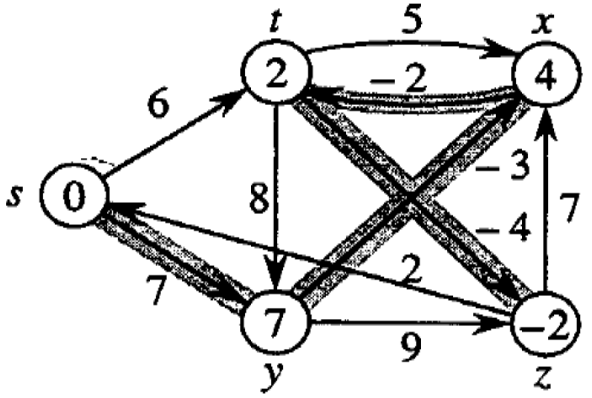
(b)



(c)



(d)



(e)

Bellman-Ford

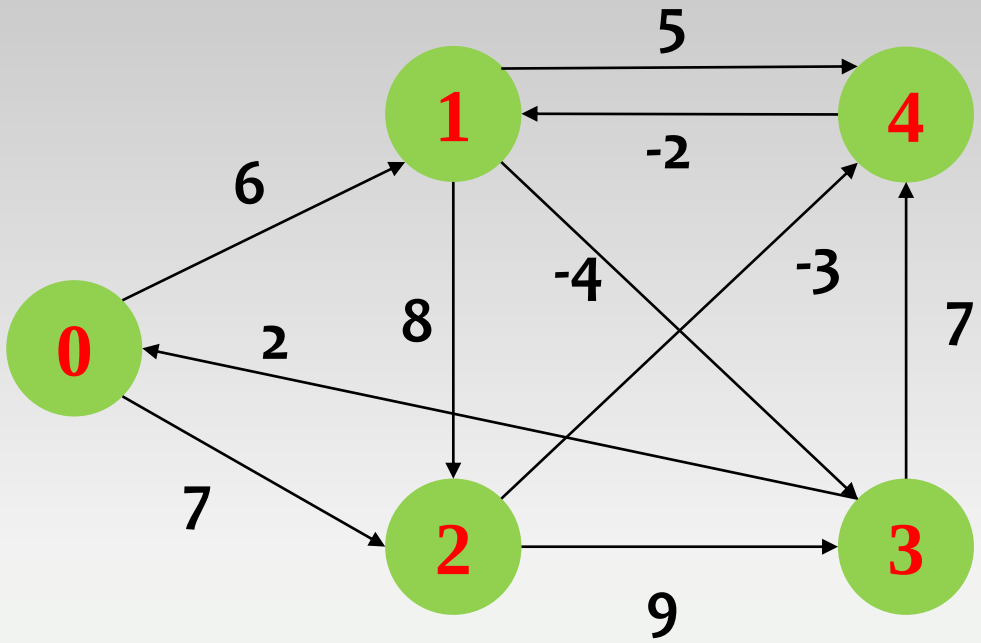
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for cada aresta (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for cada aresta (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

Bellman-Ford

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for cada vértice v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2      then d[v] ← d[u] + w(u, v)
3           π[v] ← u
```

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for cada aresta (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for cada aresta (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```



Algoritmo terminado: menor caminho definido nos vetores distância e predecessor.

	0	1	2	3	4
d[v]	0	2	7	1	4
π[v]	-1	4	0	-2	2

Bellman-Ford

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for cada aresta (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for cada aresta (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

- O algoritmo também detecta se existem ciclos negativos no segundo for.
- Se há, não é possível calcular a menor distância, com a função retornando FALSE.
- Caso contrário, a função encontrou o menor caminho da origem ‘s’ para todos os outros vértices, retornando TRUE.
- O algoritmo de Bellman-Ford é executado no tempo O(V.E).

Tópicos

- ~~Conceito Básico.~~
- ~~Algoritmo de Bellman-Ford.~~
- Caminho mais curto de uma única origem em gao's.
- Algoritmo de Dijkstra.
- Resumo.

Caminho mais curto de uma única origem em gao’s.

- Relaxando as arestas de um gao (grafo acíclico orientado) ponderado G = (V,E), de acordo com a ordenação topológica de seus vértices, pode-se calcular caminhos mais curtos de uma única origem no tempo O (V + E).

Caminho mais curto de uma única origem em gao’s.

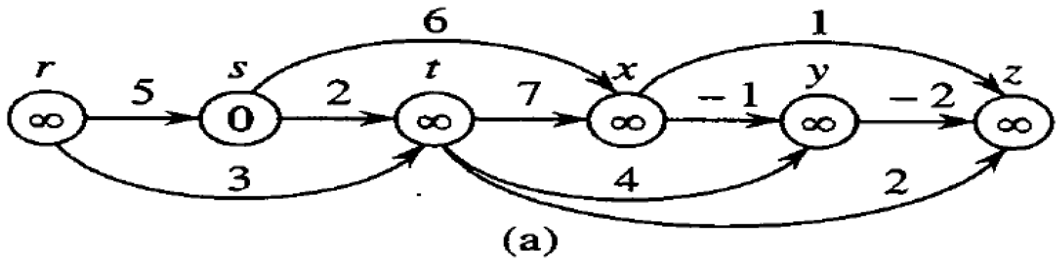
- O problema de caminhos mais curtos sempre é bem definido em um gao.
- Porquê?

Caminho mais curto de uma única origem em gao's.

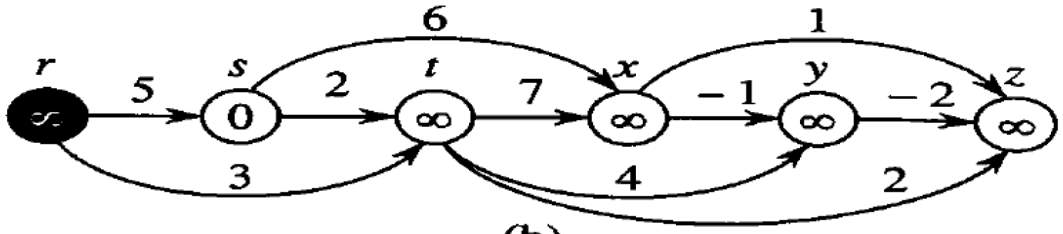
DAG-SHORTEST-PATHS(G, w, s)
1 ordenar topologicamente os vértices de G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** cada vértice u, tomado em seqüência topologicamente ordenada
4 **do for** cada vértice $v \in Adj[u]$
5 **do** RELAX(u, v, w)

Implementação fica como exercício.

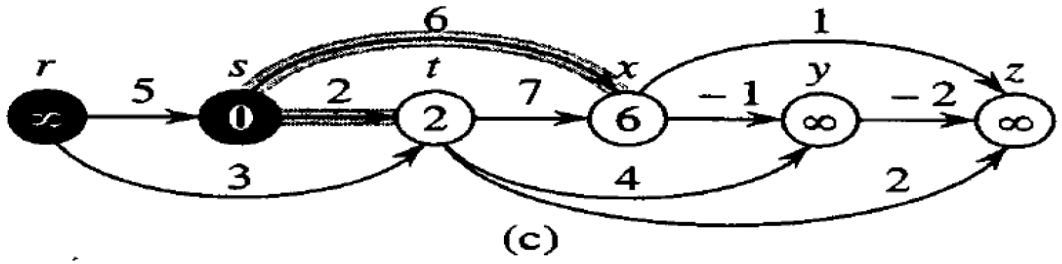
Caminho mais curto de uma única origem em gao's.



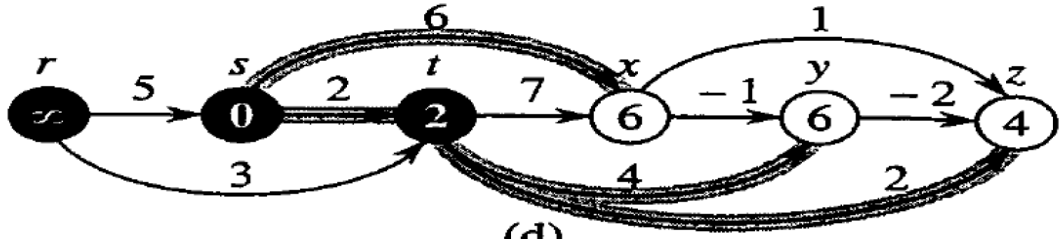
(a)



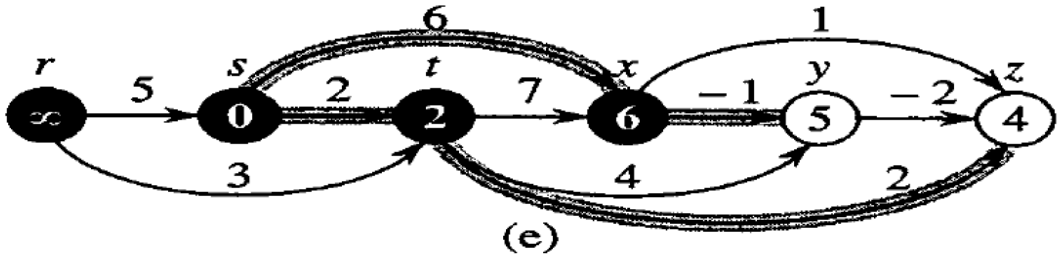
(b)



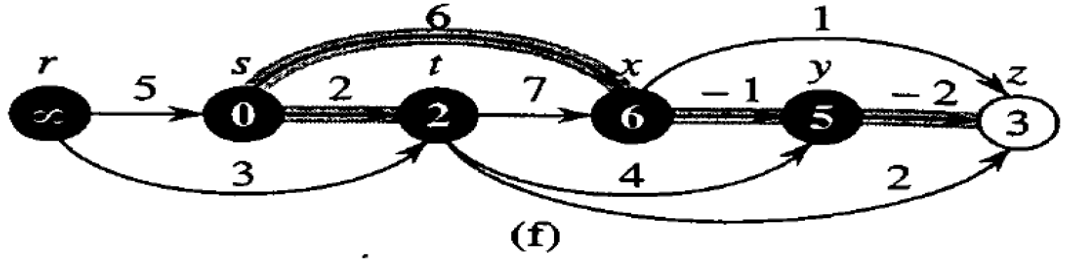
(c)



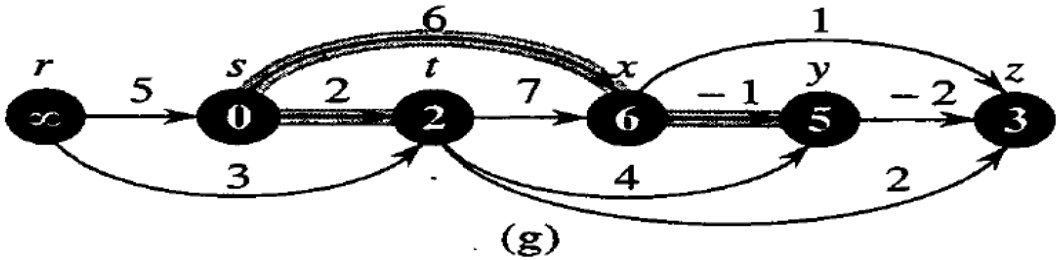
(d)



(e)



(f)



(g)

Caminho mais curto de uma única origem em gao’s.

- Uma aplicação interessante deste algoritmo é na determinação de caminhos críticos na análise de diagramas PERT.

- As arestas representam os serviços a serem executados e os pesos o tempo necessário para concluir cada serviço.

- Se a aresta (u,v) entra no vértice V e a aresta (v,x) sai de V, então o serviço (u,v) deve ser executado antes do serviço (v,x).

Caminho mais curto de uma única origem em gao’s.

- Um caminho crítico é o caminho mais longo pelo gao, correspondendo ao tempo mais longo para execução de determinada sequência ordenada de serviços.

- Duas formas:
  - Tornar os pesos das arestas negativos.
  - Ao usar o INITIALIZE_SINGLE_SOURCE, setar distância mínima dos elementos como -∞ e usar “<” em vez de “>” no RELAX.

Tópicos

- ~~Conceito Básico.~~
- ~~Algoritmo de Bellman-Ford.~~
- ~~Caminho mais curto de uma única origem em gao's.~~
- Algoritmo de Dijkstra.
- Resumo.

Dijsktra

- O algoritmo de Dijsktra, criado por Edsger Dijsktra, em 1956, e publicado em 1959, computa, para um dado dígrafo (grafo orientado) G com arestas ponderadas, o menor caminho de um nodo de origem s até cada um dos outros nodos de G.

  - Este algoritmo resolve o problema do caminho mais curto de uma única origem para grafos orientados ponderados sem arestas de peso negativo.

  - Solução mais específica que a do Bellman-Ford.

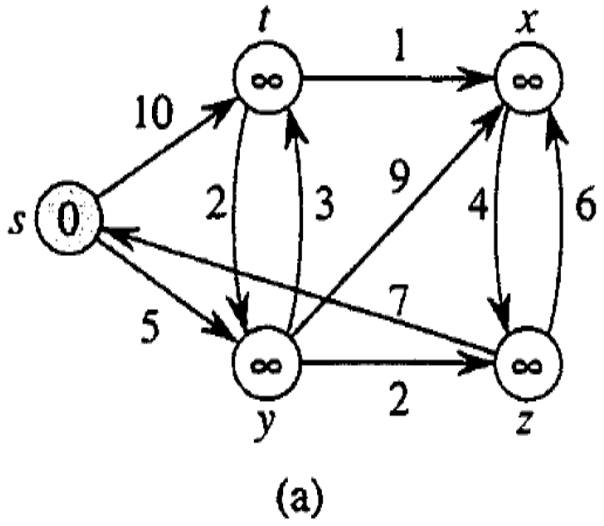
Dijsktra

- O algoritmo mantém um conjunto S de vértices cujos pesos finais de caminhos mais curtos desde a origem ‘s’ já foram determinados.

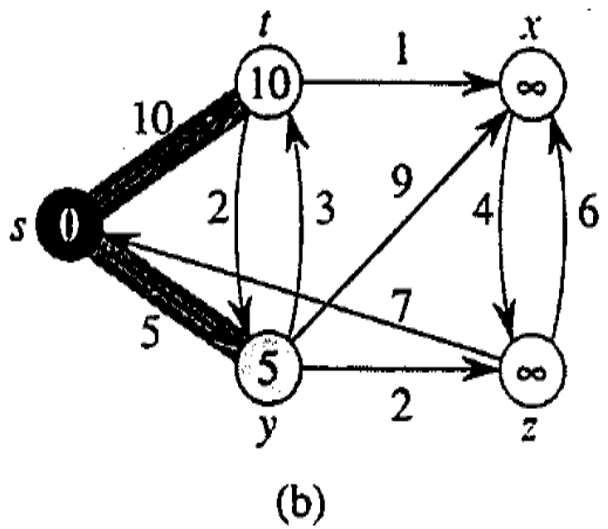
- O vértice u >= V-S é selecionado repetidamente com a estimativa mínima de caminhos mais curtos, adiciona ‘u’ a S e relaxa as arestas que saem de u.

- No exemplo, será mantida uma fila de prioridade mínima Q de vértices, tendo como chaves seus valores de ‘d’.

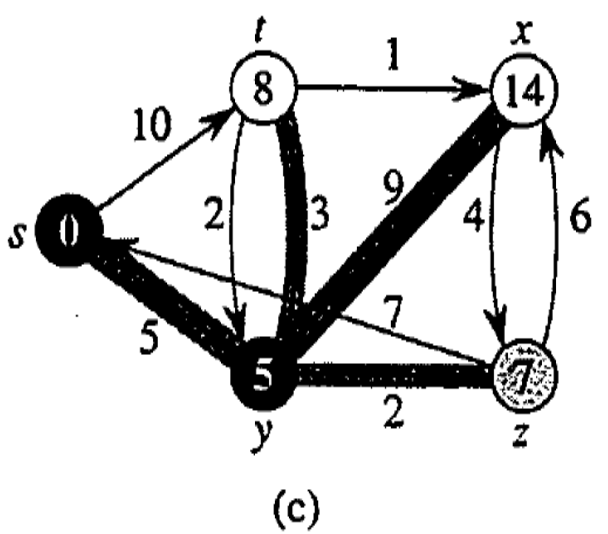
Dijsktra



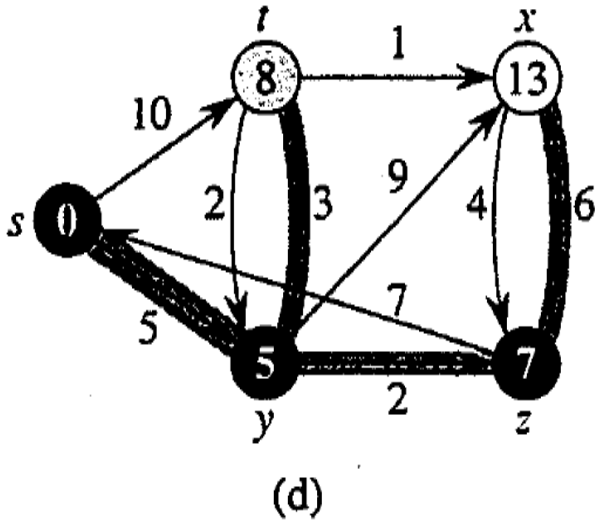
(a)



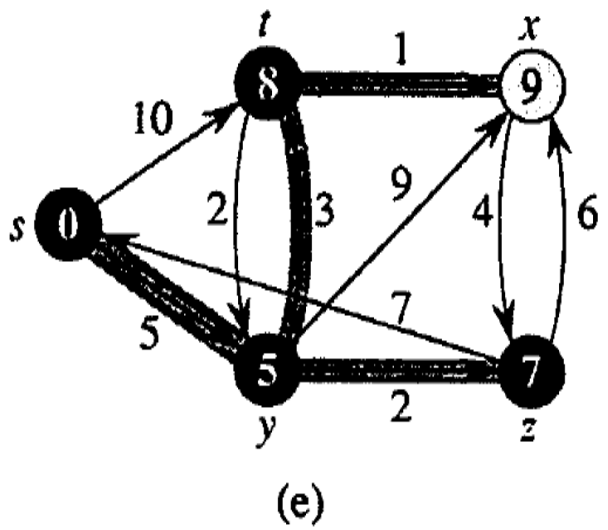
(b)



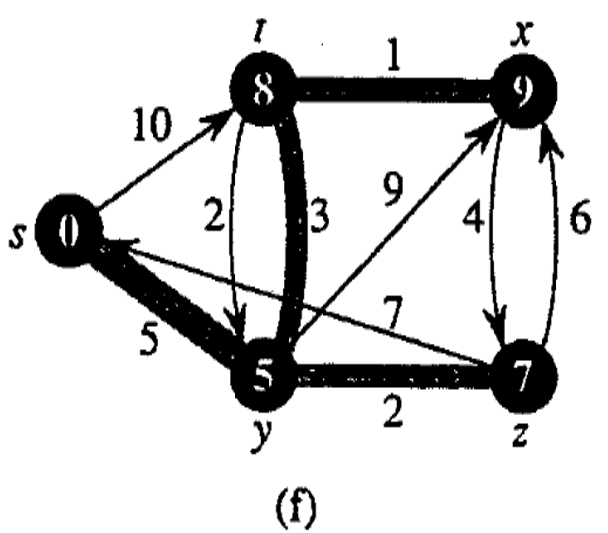
(c)



(d)



(e)



(f)

Dijsktra

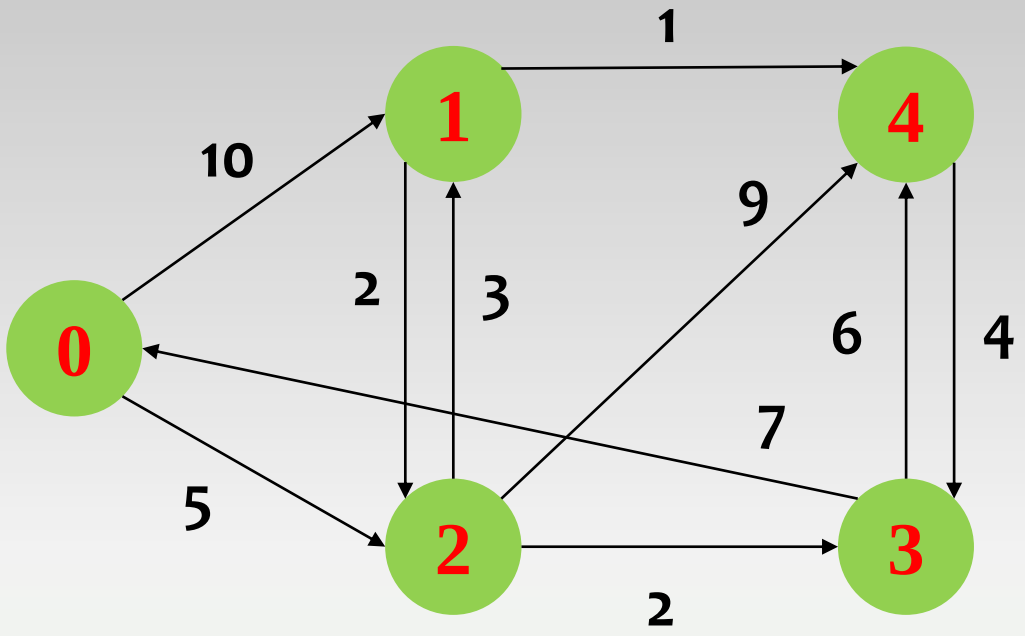
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for cada vértice v ∈ V[G]
2      do d[v] ← ∞
3         π[v] ← NIL
4  d[s] ← 0
```

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2      then d[v] ← d[u] + w(u, v)
3           π[v] ← u
```

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8            do RELAX(u, v, w)
```

Dijsktra

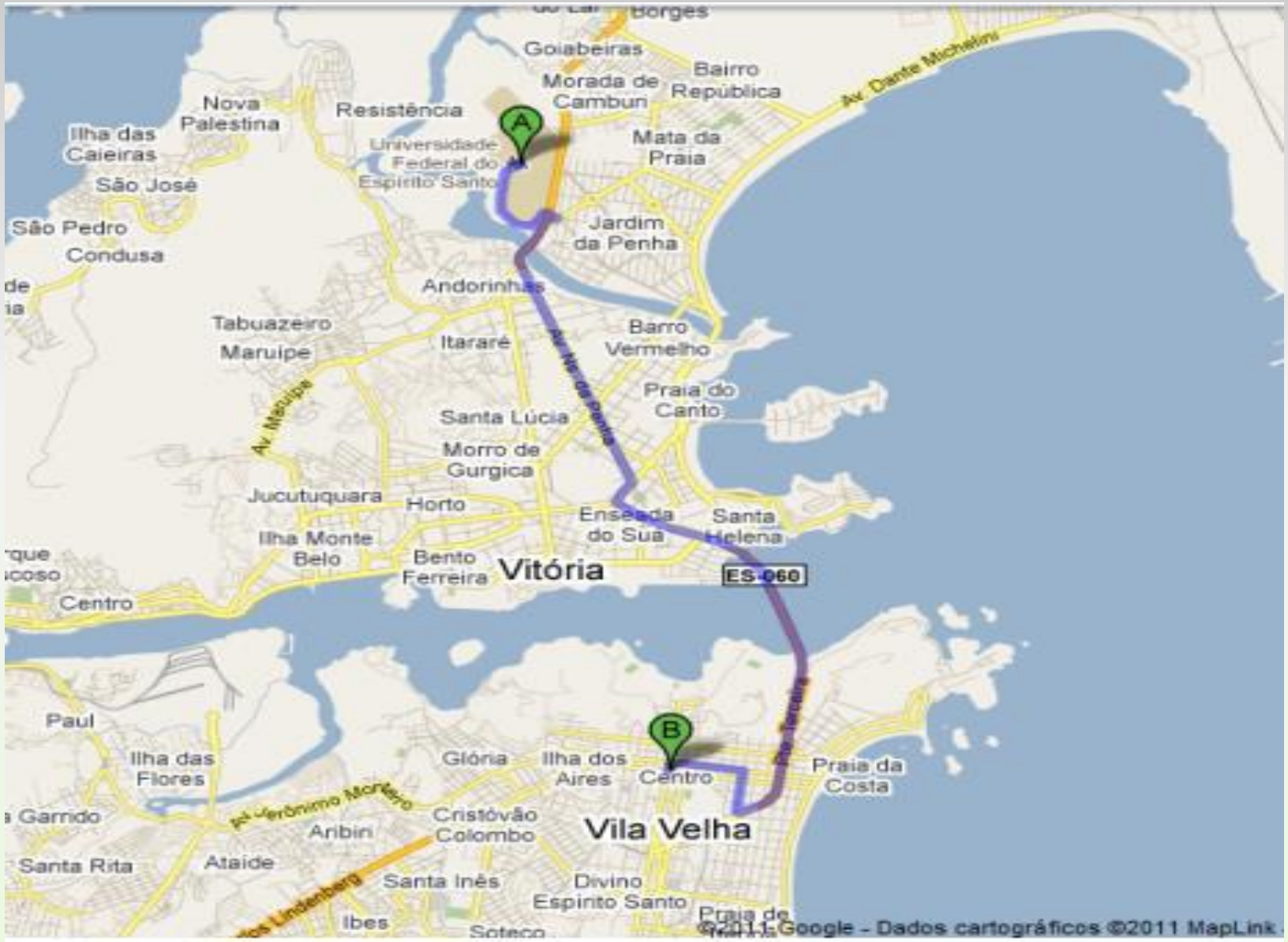


	0	1	2	3	4
π[v]	-1	2	0	2	1
d[v]	0	8	5	7	9
Q	0	1	2	3	4
S	0	2	3	1	4

Algoritmo terminado: menor caminho definido nos vetores distância e predecessor.

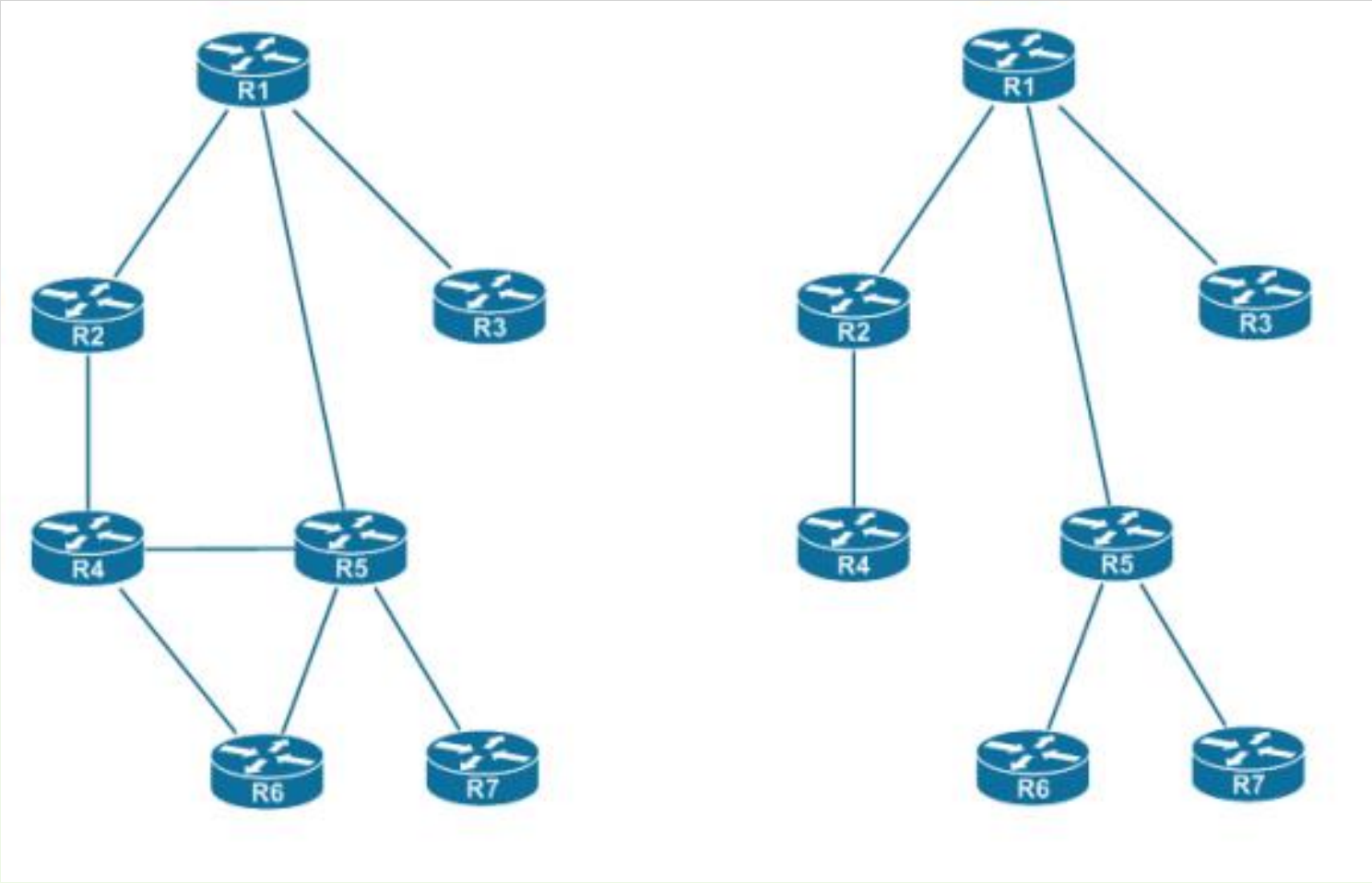
Aplicações

- Cálculo de rotas.



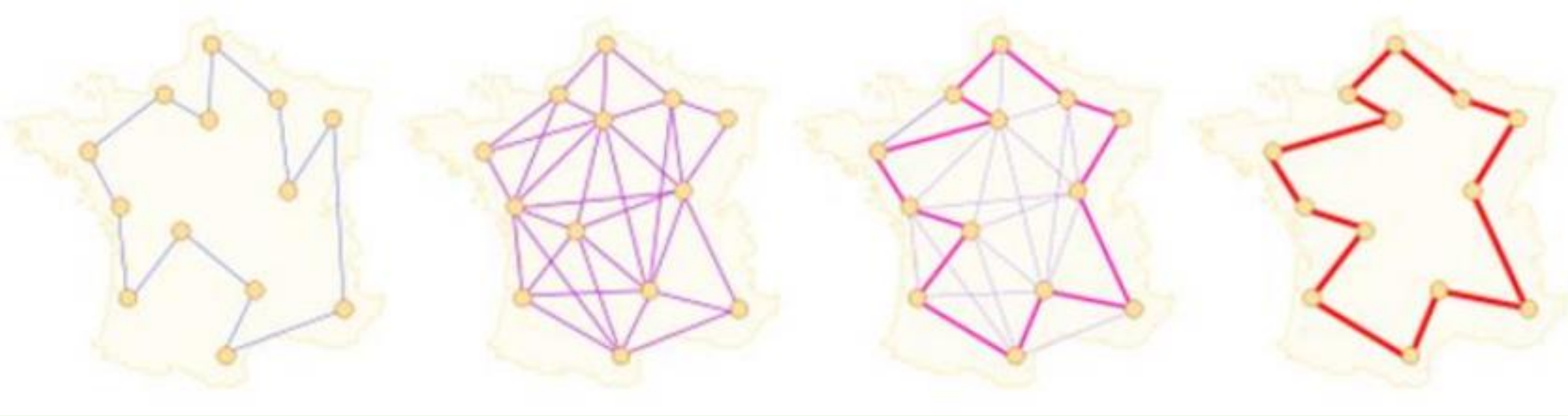
Aplicações

- Cálculo de rotas.
- Algoritmos de Roteamento.



Aplicações

- Cálculo de rotas.
- Algoritmos de Roteamento.
- Heurísticas para o PCV (NP-Difícil).



Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)

O(1)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)

O(1)

O(V)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)
O(V)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)
O(V)
O(1)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)
O(V)
O(1)
O(E) (no total)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra?

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)
O(V)
O(1)
O(E) (no total)
O(1)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra? $O(V^2 + E)$

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)
O(V)
O(1)
O(E) (no total)
O(1)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra? $O(V^2 + E)$

- É possível otimizar?

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra? $O(V^2 + E)$
- É possível otimizar?
- Efetuar a análise do algoritmo e detectar estruturas que determinam a execução e complexidade.

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra? $O(V^2 + E)$

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5     do u ← EXTRACT-MIN(Q)
6        S ← S ∪ {u}
7        for cada vértice v ∈ Adj[u]
8           do RELAX(u, v, w)
```

O(V)
O(1)
O(V)
O(V)
O(V)
O(1)
O(E) (no total)
O(1)

Dijsktra: Análise

- Qual a complexidade do algoritmo de Dijsktra? $O(V^2 + E)$
- É possível otimizar?
- Efetuar a análise do algoritmo e detectar estruturas que determinam a execução e complexidade.
- Depende da implementação da fila de prioridades Q.

Dijsktra: Análise

- Implementação com listas encadeadas ou vetores: $O(V^2 + E)$
- Implementação com heap binário: $O((V+E) \log V) = O(E \log V)$ (grafos esparsos)
- Implementação com heap de fibonacci: $O(V \log V + E)$

Tópicos

- ~~Conceito Básico.~~
- ~~Algoritmo de Bellman-Ford.~~
- ~~Caminho mais curto de uma única origem em gao's.~~
- ~~Algoritmo de Dijkstra.~~
- Resumo.

Resumo

- Foram demonstrados diversos algoritmos para o cálculo do caminho mínimo a partir de uma única origem S.

  - Bellman-Ford.
  - Caminho mínimo em grafos acíclicos.
  - Dijsktra.

Resumo

- Temos também outros algoritmos:
  - Floyd-Marshall e Johnson para o cálculo do caminho mínimo a partir de todos os pares de vértices.

Referências Bibliográficas

- CORMEN, Thomas H. **Algoritmos: teoria e prática**. Parte VI, Capítulo 23, páginas 459-474.

Dúvidas?

Professor Luciano Brum
email: lucianobrum18@gmail.com
https://sites.google.com/view/brumluciano