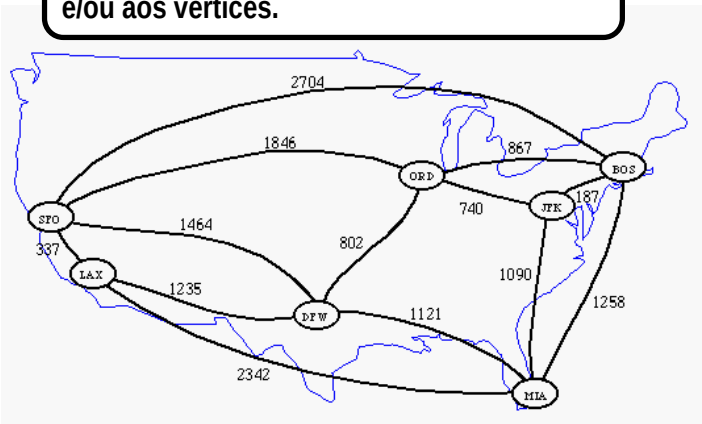
Grafos – aula 3

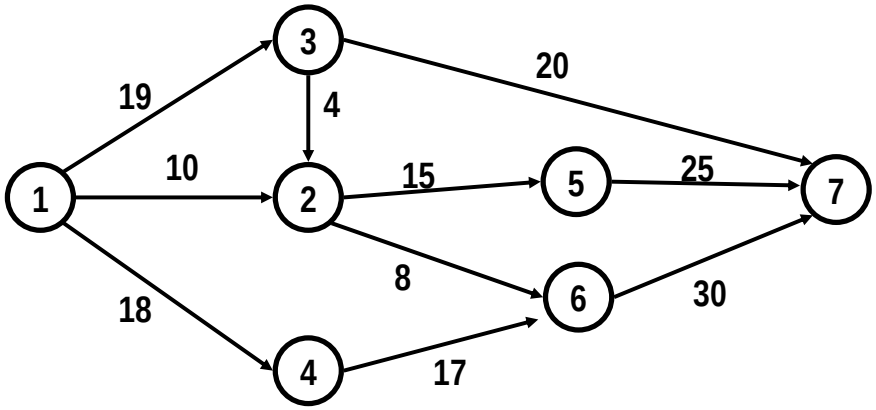
Relembrando ....

Um grafo é *valorado* (ou *ponderado*) se possuir valores associados às linhas e/ou aos vértices.

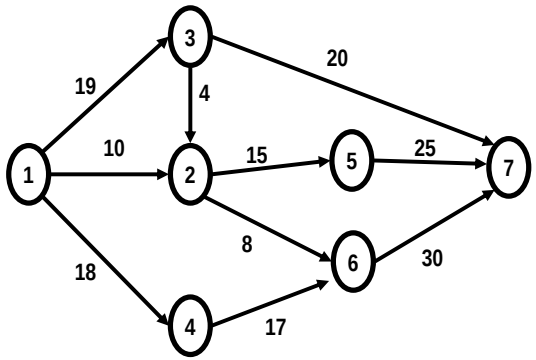


- Rota mais curta entre 2 aeroportos
- Caminho mais curto entre 2 máquinas, para transmissão de dados via internet
- Distâncias mais curtas entre cidades

Rede de eventos e atividades



Rede de eventos e atividades



vértices ⟹ eventos

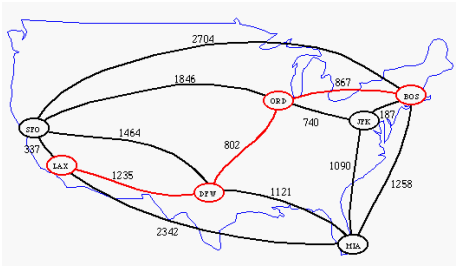
arcos ⟹ atividades

- grafo dirigido
- valorado
- sem ciclos
- fonte e sumidouro

Exs: PERT – técnica de avaliação e revisão de desempenho
CPM – método de caminho crítico

Problema do caminho mínimo

- Dado um grafo ponderado e dois vértices ***v*** e ***u***, queremos encontrar o caminho de custo total mínimo entre ***v*** e ***u***
- O comprimento (ou peso) de um caminho ***P*** é a soma dos pesos das arestas que compõem ***P***
- Exemplo: qual o menor caminho de Boston a Los Angeles?



Problema do caminho mínimo

Propriedade 1:

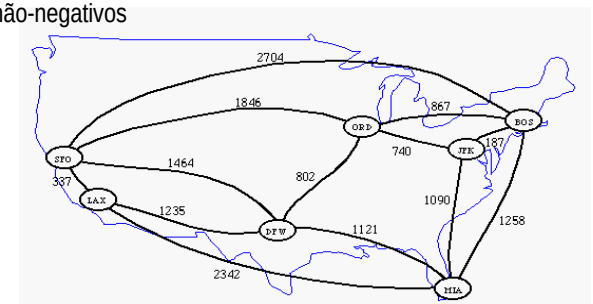
Um sub-caminho de um caminho mínimo é, ele próprio, um caminho mínimo

Propriedade 2:

Existe uma árvore de caminhos mínimos partindo de um vértice inicial até todos os outros vértices

Algoritmo de Dijkstra

- Também conhecido como método **guloso**
- O algoritmo de Dijkstra encontra o caminho mínimo a partir de um vértice de partida ***v*** até todos os outros vértices
- Funciona em:
  - Grafos dirigidos e não-dirigidos
  - Grafos com pesos de arestas não-negativos



Algoritmo de Dijkstra: funcionamento

- Dado um vértice ***v*** (vértice de partida), o algoritmo calcula, para cada vértice ***u***, a distância mínima até ***v***
- O algoritmo mantém o conjunto de vértices (**nuvem C**) cujas distâncias tenham sido computadas
- Cada vértice ***u*** possui um rótulo ***D[u]*** associado.
  - ***D[u]*** armazena uma aproximação da distância entre ***v*** e ***u***.
  - O algoritmo altera ***D[u]*** cada vez que encontra uma distância menor
- Quando um vértice ***u*** é adicionado à nuvem, seu rótulo ***D[u]*** é igual a distância final (até o momento) entre ***v*** e ***u***
- Condição inicial
  - D[v] = 0 (distância de ***v*** até ***v*** é zero)
  - D[u] = ∞ para u ≠ v

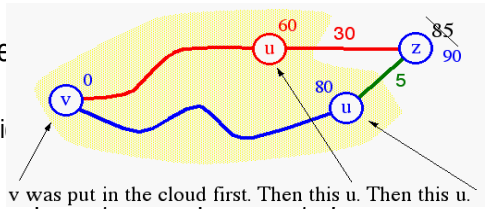
Algoritmo: expandindo a nuvem

- A nuvem ***C*** é inicialmente vazia
- Repita até que todos os vértices tenham sido adicionados à nuvem ***C***:
  - Selecione o vértice ***u*** que não esteja em ***C*** e que tenha o menor rótulo ***D[u]***
    - na primeira iteração, será escolhido o vértice inicial ***v*** – ver condições iniciais!
  - Coloque-o em ***C***
  - Atualize os rótulos dos vértices adjacentes a ***u*** da seguinte forma:

```
para cada vértice z adjacente a u faça
  se z não está na nuvem C, então
       se D[u] + weight(u,z) < D[z] então
             D[z] = D[u] + weight(u,z)
```


Algoritmo: expandindo a nuvem

- A nuvem ***C*** é inicialmente vazia
- Repita até que todos os vértices tenham sido adicionados à nuvem ***C***:
  - Selecione o vértice ***u*** que não e
    D[u]
    - na primeira iteração, será escolhi
  - Coloque-o em ***C***
  - Atualize os rótulos dos vértices adjacentes a ***u*** da seguinte forma:

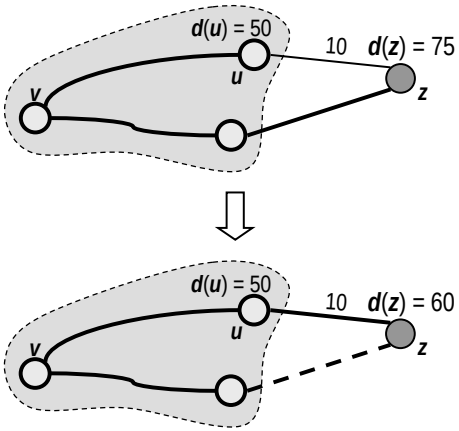


v was put in the cloud first. Then this u. Then this u.

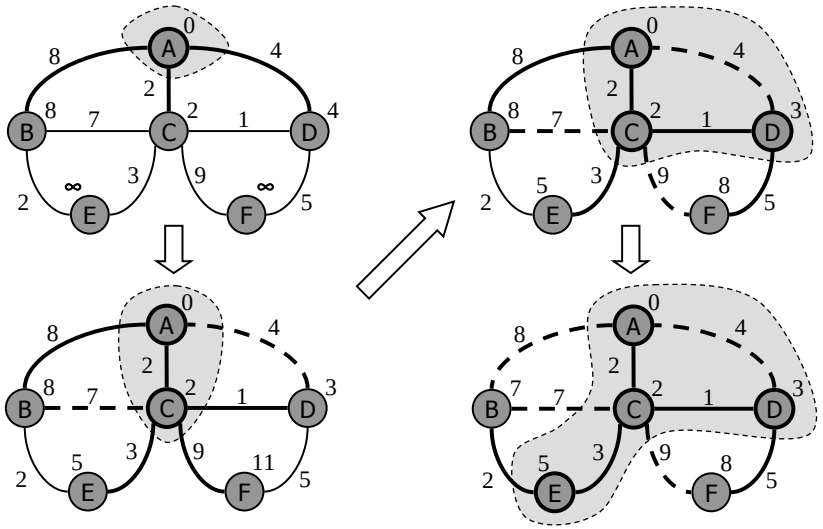
```
para cada vértice z adjacente a u faça
  se z não está na nuvem C, então
       se D[u] + weight(u,z) < D[z] então
             D[z] = D[u] + weight(u,z)
```


Atualizando a distância d(z)

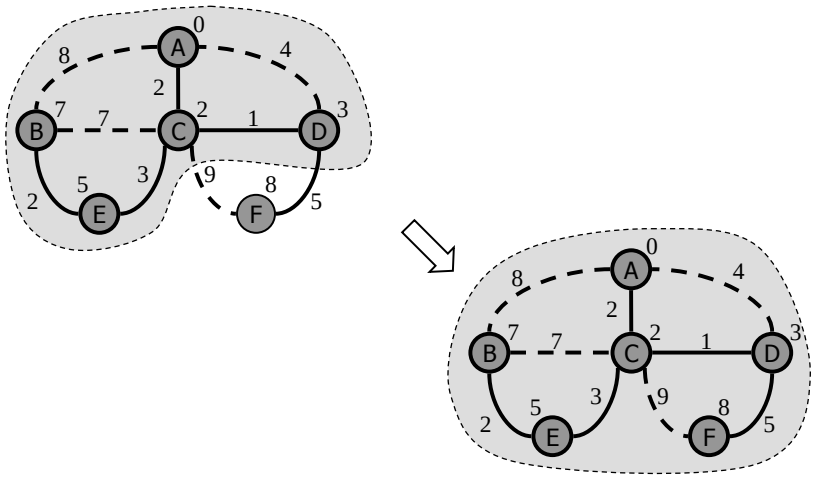
- Considere uma aresta ***e = (u,z)*** tal que
  - ***u*** é o último vértice adicionado à nuvem
  - ***z*** não está na nuvem
- A atualização da distância ***d(z)*** em função da aresta ***e*** é realizada para encontrar a situação ótima
- O **relaxamento da aresta *e* = (*u*,*z*)** atualiza a distância ***d***(***z***)



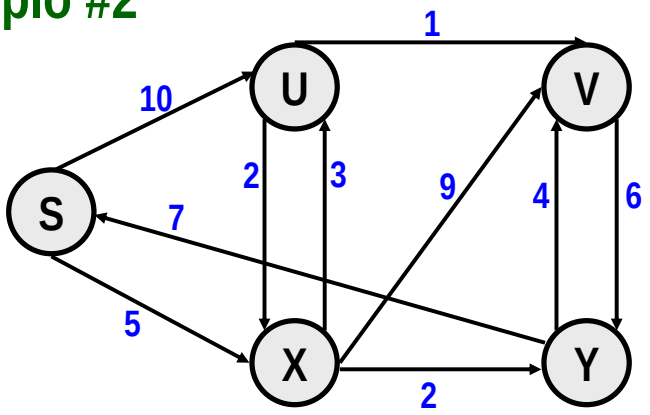
Exemplo #1

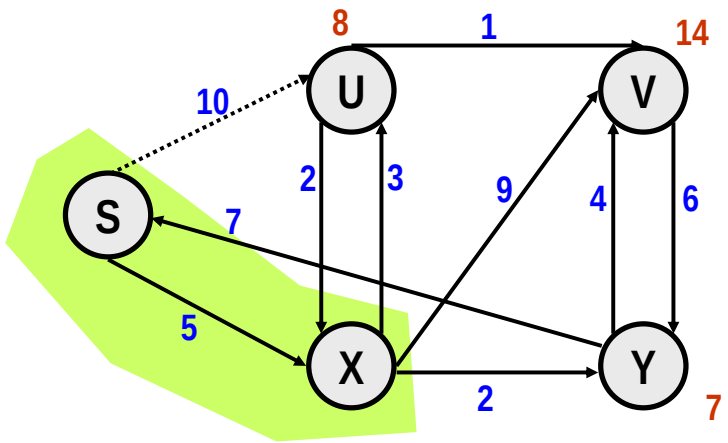


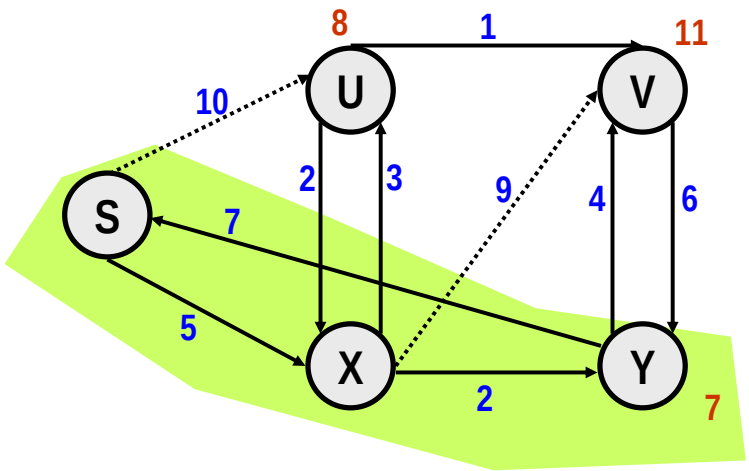
Exemplo #1

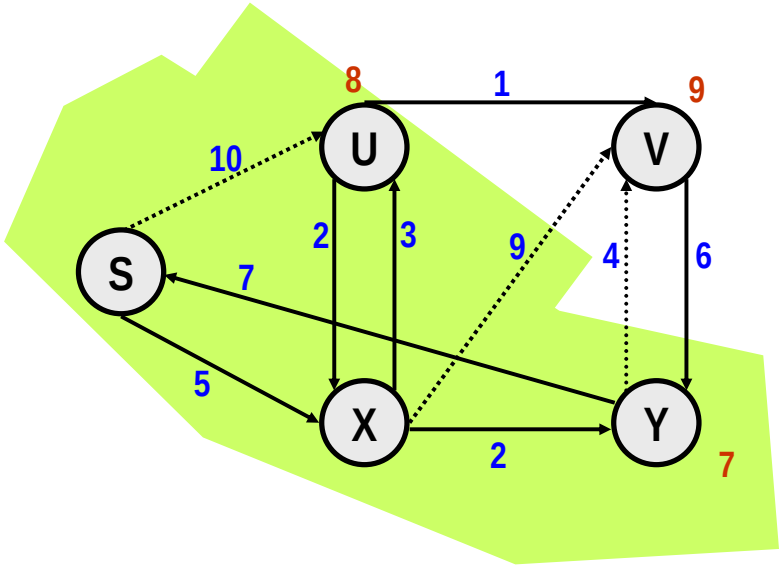


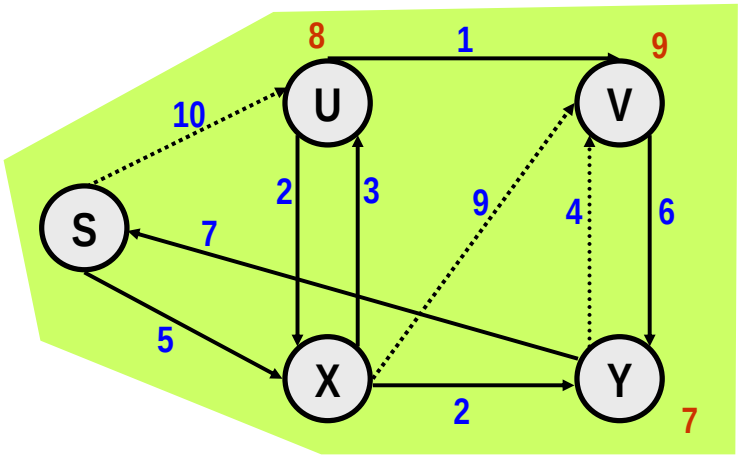
Exemplo #2



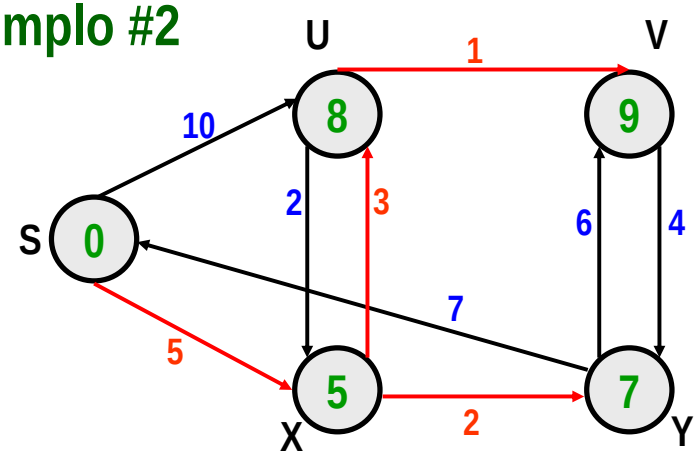








Exemplo #2



Algoritmo de Dijkstra

Algoritmo:

Seja G(V,A) um grafo orientado e s um vértice de G:

1. Atribua valor zero à estimativa do custo mínimo do vértice s (a raiz da busca) e infinito às demais estimativas;

2. Atribua um valor qualquer aos precedentes (o precedente de um vértice t é o vértice que precede t no caminho de custo mínimo de s para t)

3. Enquanto houver vértice aberto:

- seja k um vértice ainda aberto cuja estimativa seja a menor dentre todos os vértices abertos;
- feche o vértice k
- Para todo vértice j ainda aberto que seja sucessor de k faça:
  - some a estimativa do vértice k com o custo do arco que une k a j;
  - caso esta soma seja melhor que a estimativa anterior para o vértice j, substitua-a e anote k como precedente de j.

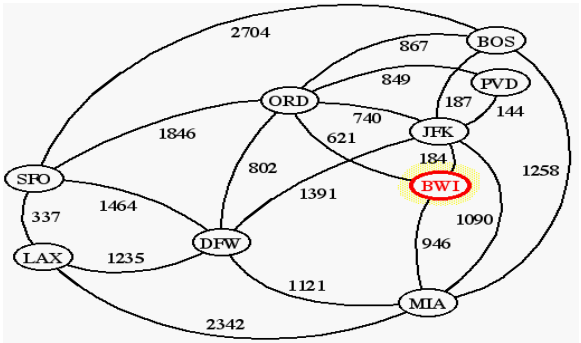
Algoritmo de Dijkstra

- Uma lista ordenada armazena os vértices que não fazem parte da nuvem
  - Chave: distância
  - Elemento: vértice

Algoritmo CaminhoMínimo(G,s)
Entrada: grafo G e vértice inicial s
Saída: rótulo D[u] para cada vértice u de G

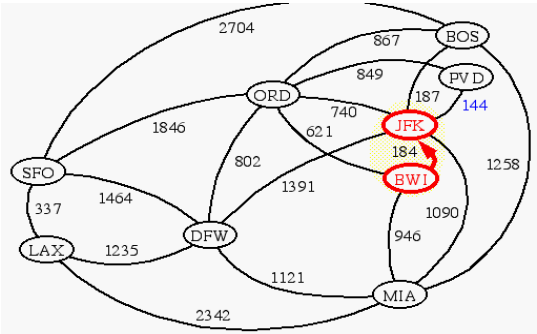
Q ← lista ordenada por distância
para todo *v* ∈ *G*
  se *v = s*
    v.setDistance(0)
  senão
    v.setDistance(∞)
    enquanto *Q não for vazia* **faça**
      u ← *Q.removeMin*()
      para todo vértice *z* **adjacente a** *u* **tal que** *z* **esteja em** *Q* **faça**
        { relaxa aresta }
        se *D[u] + w(u,z) < D[z]* **então**
          z.setDistance(D[u]+w(u,z))
          reordene lista *Q*

Exemplo #3: caminho mais curto começando por Washington (BWI)



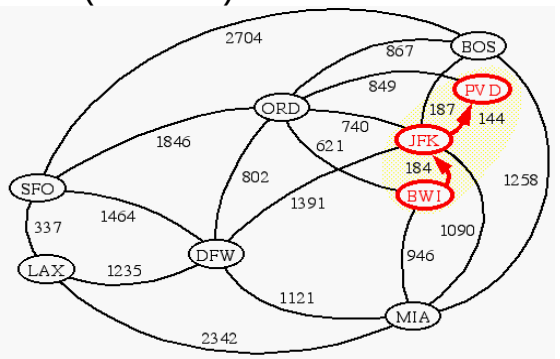
	parent	distance
BOS		∞
BWI		0
DFW		∞
JFK	BWI	184
LAX		∞
MIA	BWI	946
ORD	BWI	621
PVD		∞
SFO		∞

- JFK é o mais próximo...



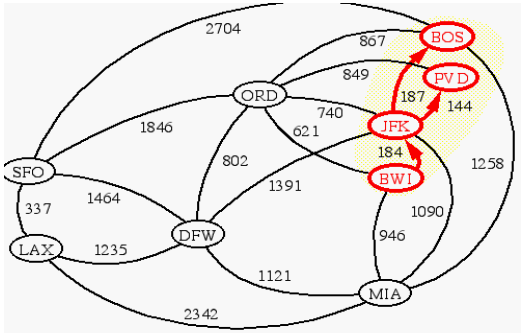
	parent	distance
BOS	JFK	371
BWI		0
DFW	JFK	1575
JFK	BWI	184
LAX		∞
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO		∞

- Seguido por PVD (Warwick)



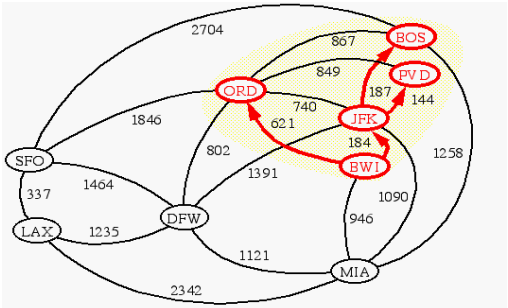
	parent	distance
BOS	JFK	371
BWI		0
DFW	JFK	1575
JFK	BWI	184
LAX		∞
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO		∞

- Boston (BOS) é o próximo



	parent	distance
BOS	JFK	371
BWI		0
DFW	JFK	1575
JFK	BWI	184
LAX		∞
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO	BOS	3075

- Chicago (ORD) segue

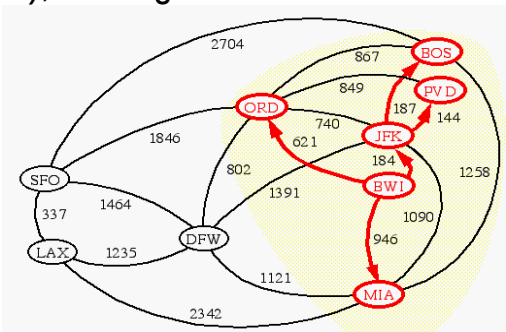


	parent	distance
BOS	JFK	371
BWI		0
DFW	ORD	1423
JFK	BWI	184
LAX		∞
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO	ORD	2467

note that D for DWF was adjusted on this turn

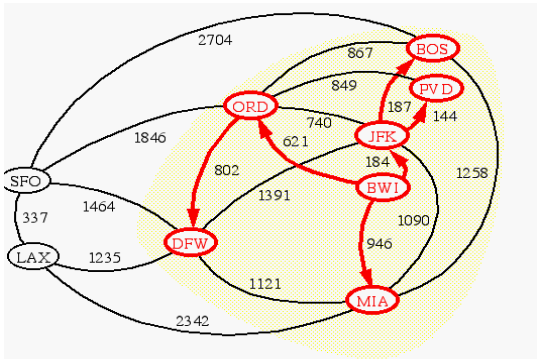
also for SFO

- Miami (MIA), em seguida



	parent	distance
BOS	JFK	371
BWI		0
DFW	JFK	1423
JFK	BWI	184
LAX	MIA	3288
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO	BOS	2467

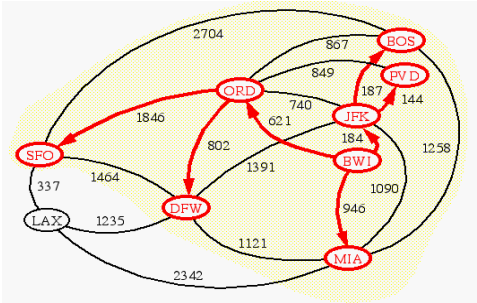
- E depois Dallas (DFW)



	parent	distance
BOS	JFK	371
BWI		0
DFW	JFK	1423
JFK	BWI	184
LAX	DFW	2658
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO	BOS	2467

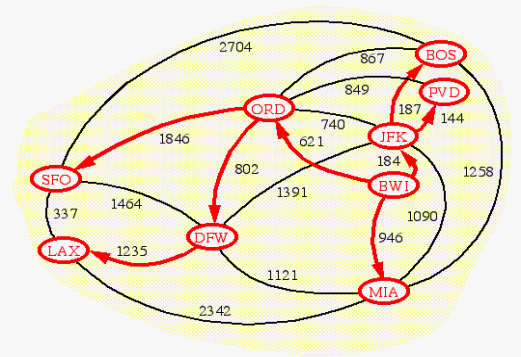
and D for LAX gets updated

- San Francisco (SFO): o próximo



	parent	distance
BOS	JFK	371
BWI		0
DFW	ORD	1423
JFK	BWI	184
LAX	MIA	2658
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO	BOS	2467

- Los Angeles (LAX) é o último



	parent	distance
BOS	JFK	371
BWI		0
DFW	ORD	1423
JFK	BWI	184
LAX	MIA	2658
MIA	BWI	946
ORD	BWI	621
PVD	JFK	328
SFO	BOS	2467

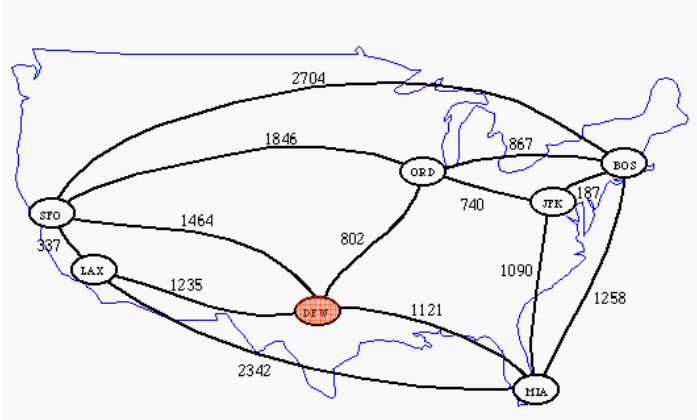
Algoritmo de Dijkstra

- Uma lista ordenada armazena os vértices que não fazem parte da nuvem
  - Chave: distância
  - Elemento: vértice

Algoritmo CaminhoMínimo (G,s)
Entrada: grafo **G** e vértice inicial **s**
Saída: rótulo **D[u]** para cada vértice **u** de **G**

Q ← lista ordenada por distância
para todo *v* ∈ *G*
  se *v* = *s*
    setDistance (v, 0)
  senão
    setDistance (v, ∞)
    enquanto *Q não for vazia* **faça**
      u ← *removeMin(Q)*
      para todo vértice *z* **adjacente a** *u* **tal que** *z*
      esteja em *Q* **faça**
        { relaxa aresta }
        se *D[u] + w(u,z) < D[z]* **então**
          setDistance (*z, D[u]+w(u,z)*)
          reordene lista *Q*

Exercício



Exercício

- Com base no que foi apresentado em aula a respeito de grafos, escreva a estrutura de dados grafo para grafos não-dirigidos, usando uma matriz de adjacência
  - Demonstre as estruturas de dados necessárias para armazenar um grafo
  - Liste as operações que podem ser realizadas
- Implemente o algoritmo de Dijkstra para essa estrutura.