SIN110 Algoritmos e Grafos

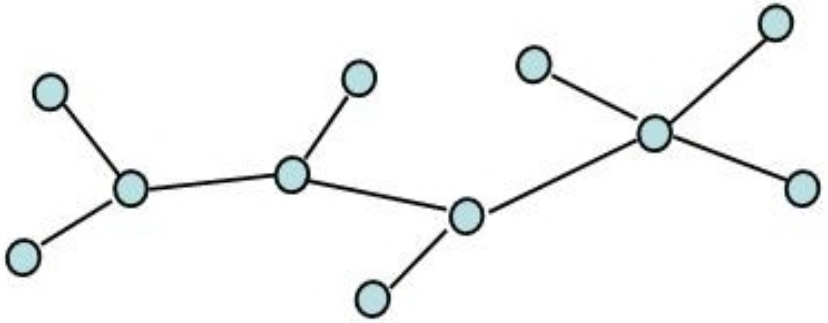
aula 14

Grafos

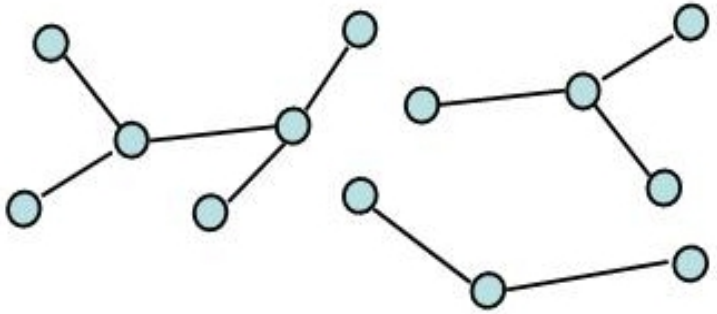
- *Árvores Geradoras Mínimas* (E10)
- ***Caminhos Mínimos***

Árvores Geradoras Mínimas

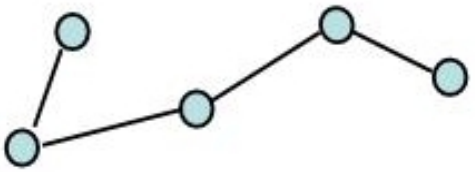
- **Árvore:** grafo conexo sem circuitos



- **Floresta:** grafo cujas componentes conexas são árvores



Um caminho é uma árvore?



Sim!

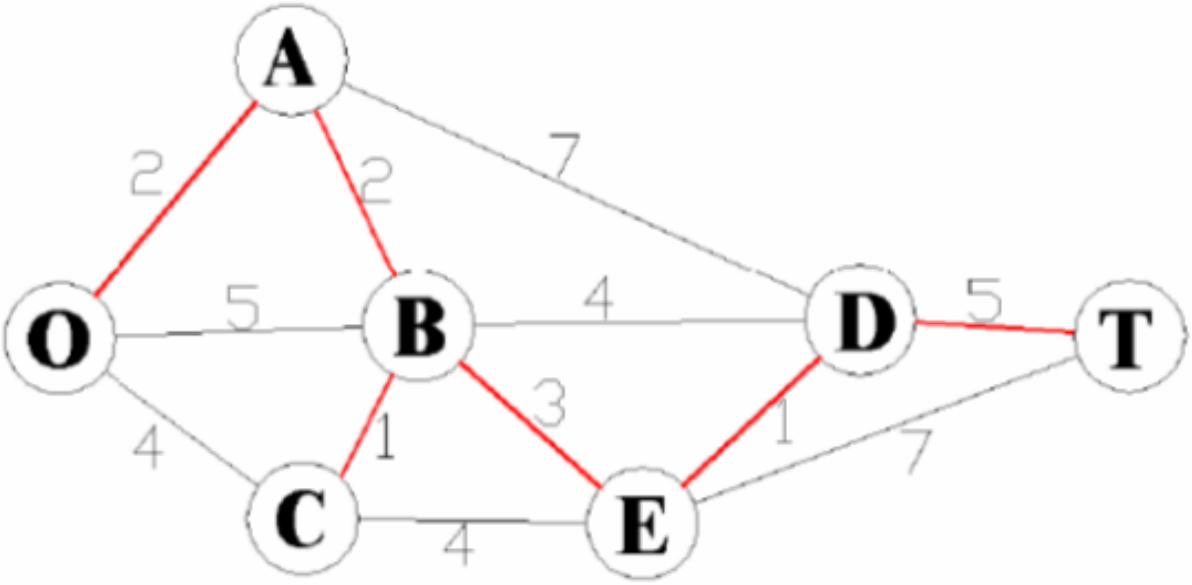
Árvore Geradora Mínima

- **Termo em inglês:**
  - **Minimum Spanning Tree (MST)**

- **Alguns sinônimos encontrados:**
  - **Árvores de Expansão Mínima**
  - **Árvores Geradora Mínima**
  - **Árvores de Cobertura Mínima**
  - **Árvores Espalhadas Mínimas**

conceito

- **Árvore que interliga todos os vértices do grafo utilizando arestas com um custo total mínimo**

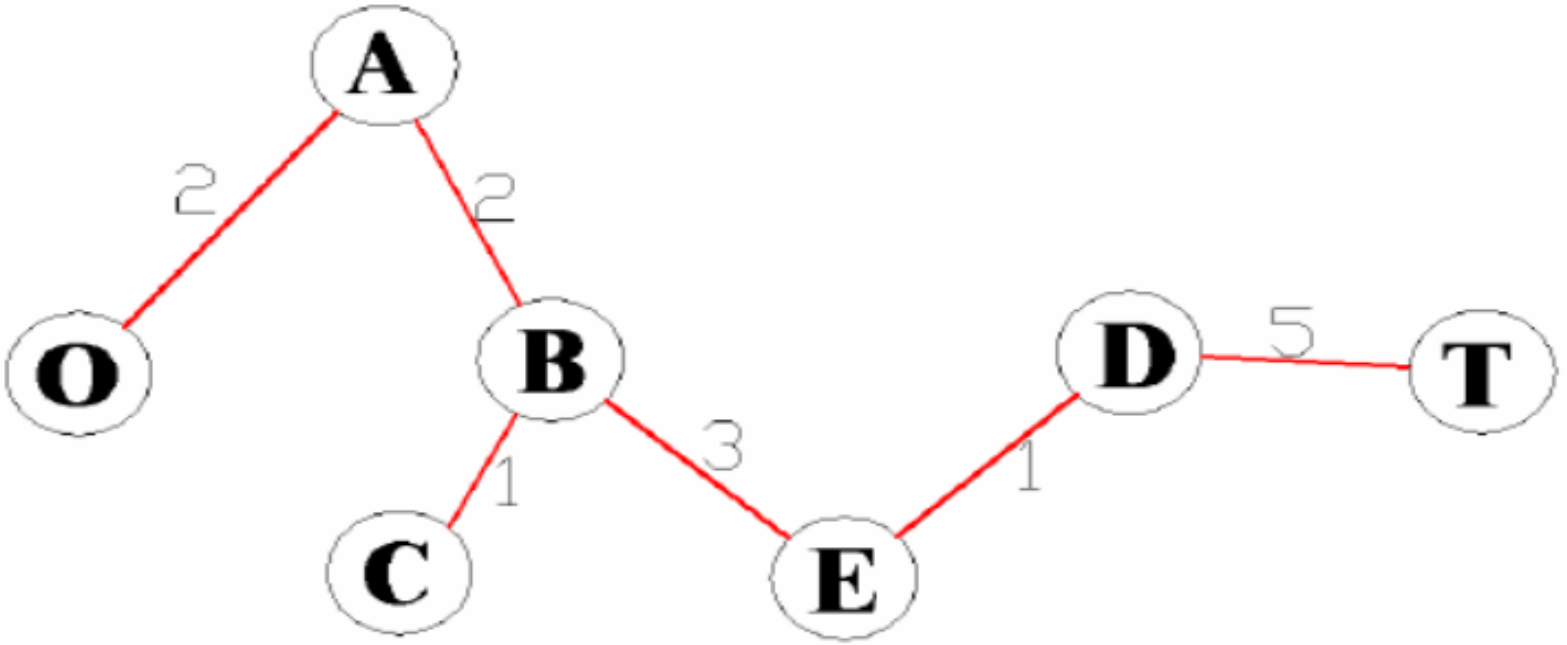


conceito

- **Dado um grafo não dirigido ponderado G, desejamos encontrar uma árvore T que contenha todos os vértices de G e minimize sua soma.**

A solução final possui uma distância total de 14 milhas. Através desta rede, é possível ir de um posto a qualquer outro posto.

Independente do nó inicial, a solução será a mesma.



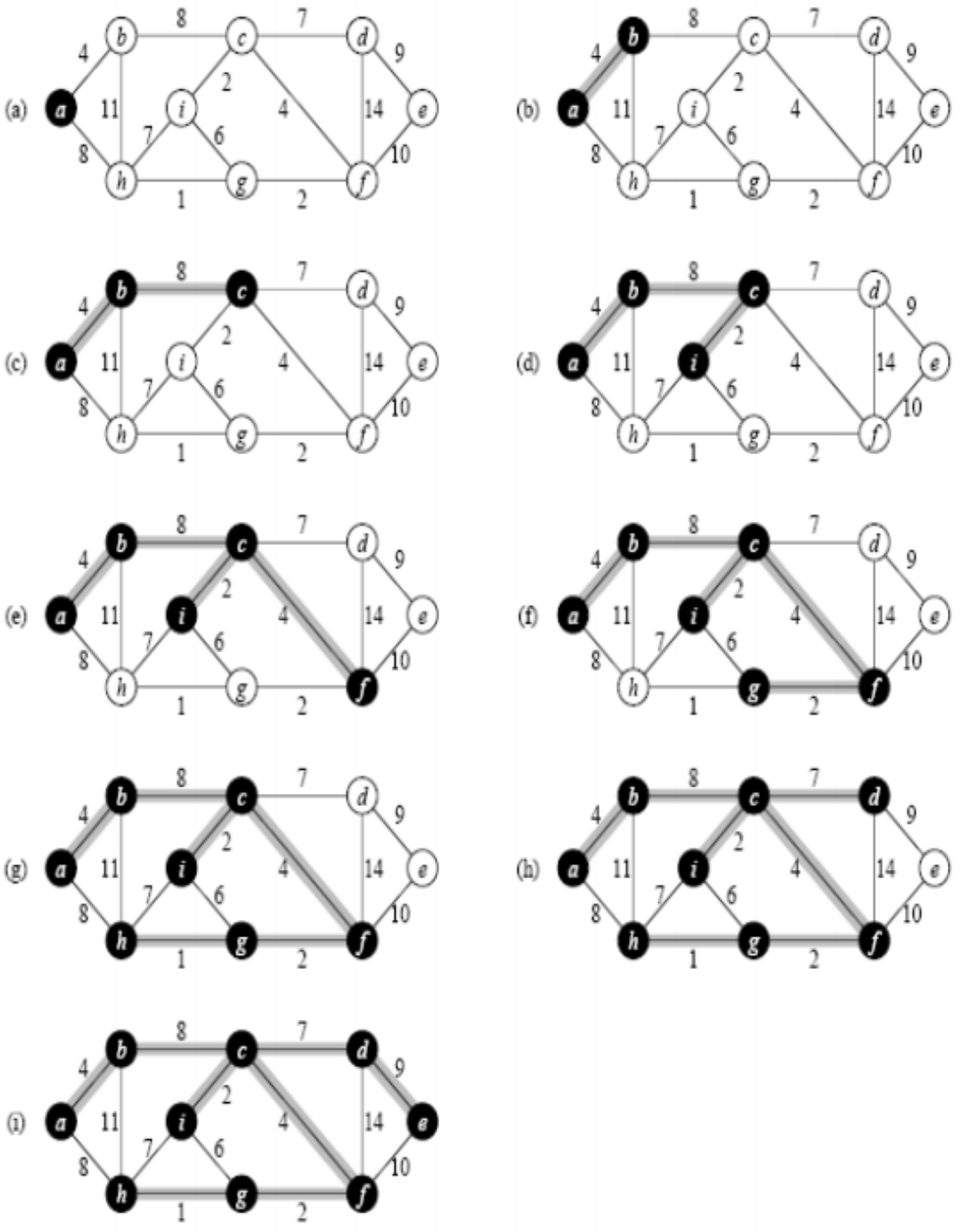
Algoritmo de Prim

O algoritmo de Prim usa uma "fila" *de vértices*, com prioridade ditada por uma chave

```
MST_Prim(G, x)
 1.  para u ← 1 até n faça
 2.        cor[u] ← BRANCO
 3.        pred[u] ← NIL
 4.        chave[u] ← ∞
 5.  Q ← Crie-Fila-Vazia()
 6.  para u ← 1 até n faça
 7.         Insira-na-Fila(u,Q)
 8.  chave[x] ← 0
 9.  pred[x] ← x
10.  enquanto Q ≠ ∅ faça
11.        u ← Retire-Mínimo(Q)
12.        para cada v em Adj[u]faça
13.              se cor[v] = BRANCO e w(u,v) < chave[v]
14.                  então chave[v] ← w(u,v)
15.                        Refaça-Fila(v,Q)
16.                        pred[v]←u
17.         cor[u] ← PRETO
18.  devolve pred
```

Algoritmo de Prim

- Idéia básica:
  - Tomando como vértice inicial A, crie uma fila de prioridades classificada pelos pesos das arestas conectando A.
  - Repita o processo até que todos os vértices tenham sido visitados.



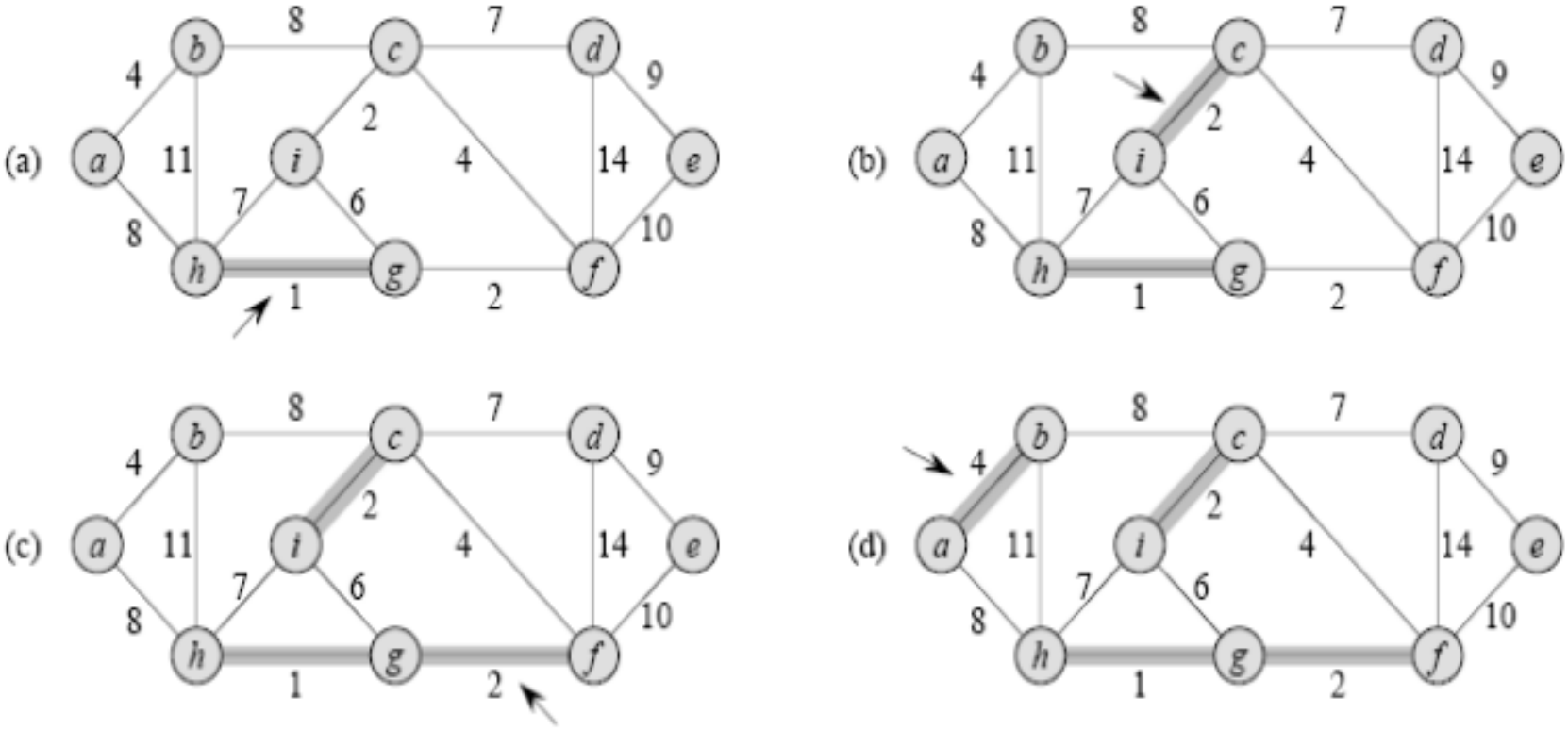
Algoritmo de Kruskal

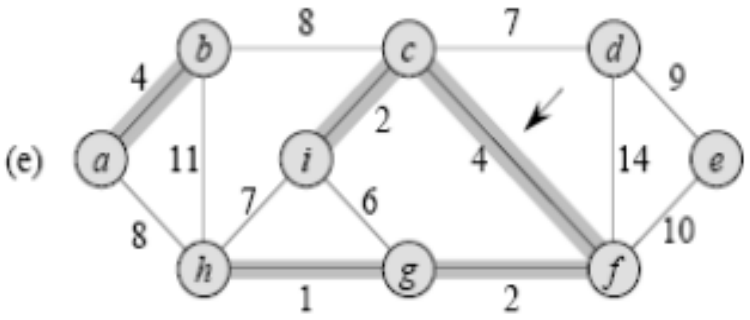
O algoritmo de Kruskal começa com uma floresta de árvores formada por apenas um vértice e procura, a cada passo, uma aresta que, além de conectar duas árvores distintas da floresta, possua peso mínimo.

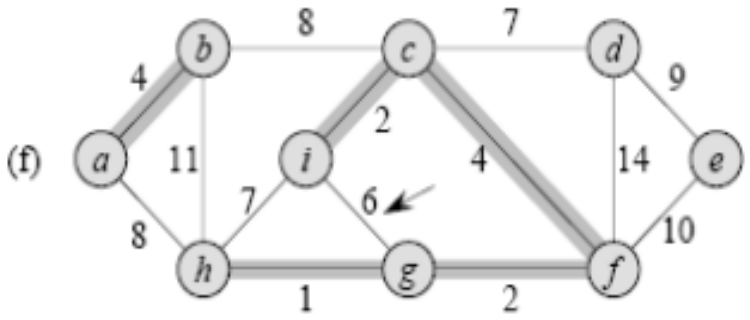
```
MST_Kruskal(G,w)
1.  A ← ∅
2.  para u ← 1 até n faça Construa_Conjunto(u)
3.  Ordene(E)                          {arestas de E em ordem crescente de peso (w)}
4.  para cada aresta (u,v) faça  {tomadas em ordem crescente de peso (w)}
5.        se Conjunto(u) ≠ Conjunto(v)
6.              então A ← A ∪ {(u,v)}
7.                    Une_Árvores(u,v)
8.  devolve A
```

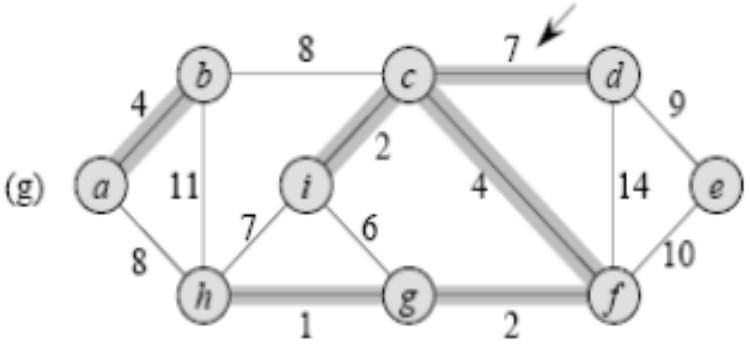
Algoritmo de Kruskal

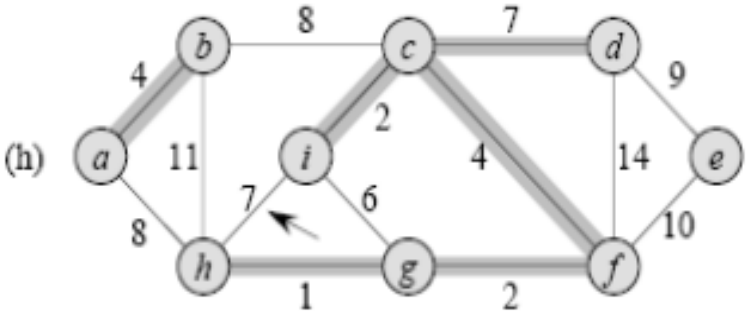
- Idéia básica:
  - Seleciona a aresta de menor peso que conecta duas árvores de uma floresta.
  - Repita o processo até que todos os vértices estejam conectados sempre preservando a invariante de se ter uma árvore.

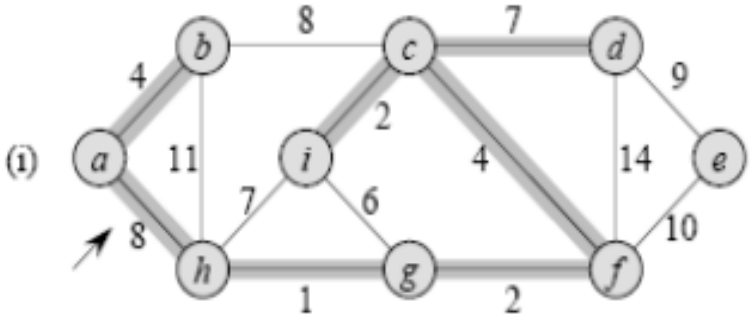


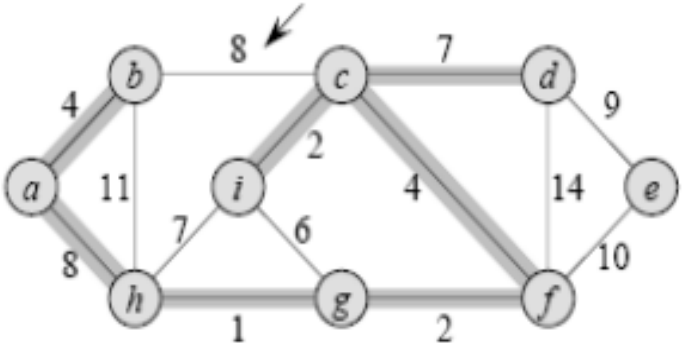

(e)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2


(f)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2

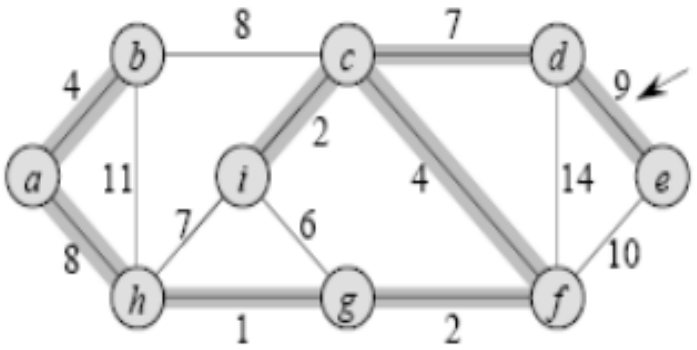

(g)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2


(h)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2

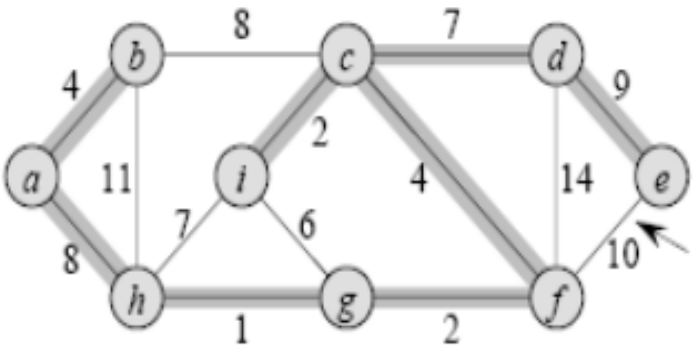

(i)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2


(j)
a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
14
9
7
6
10
1
2

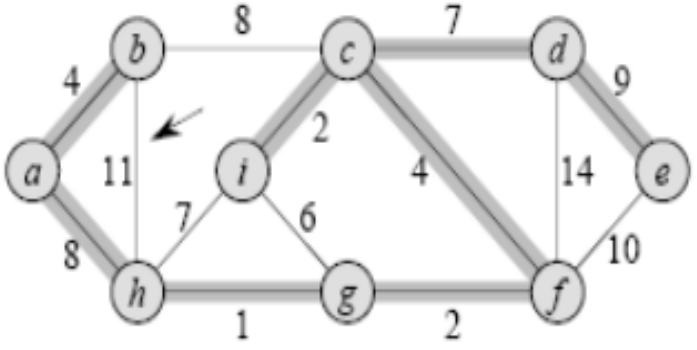
(k)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
4
14
9
6
10
1
2

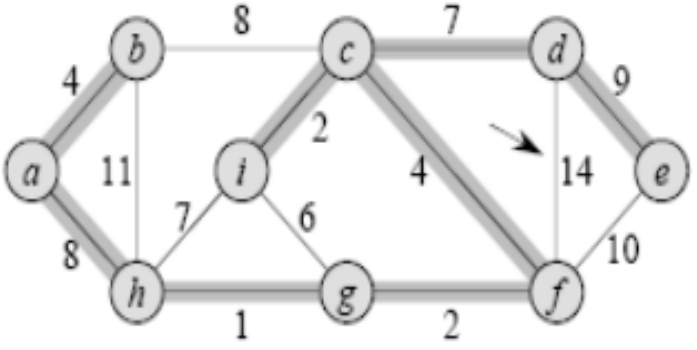
(l)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
4
14
9
6
10
1
2

(m)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
4
14
9
6
10
1
2

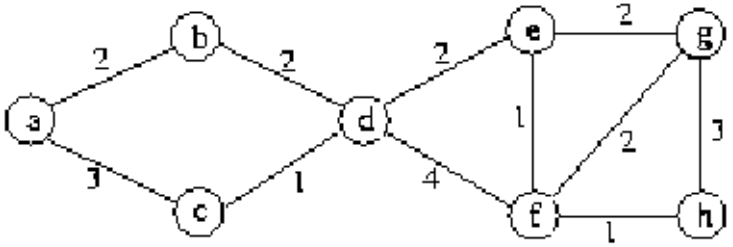
(n)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
7
4
14
9
6
10
1
2

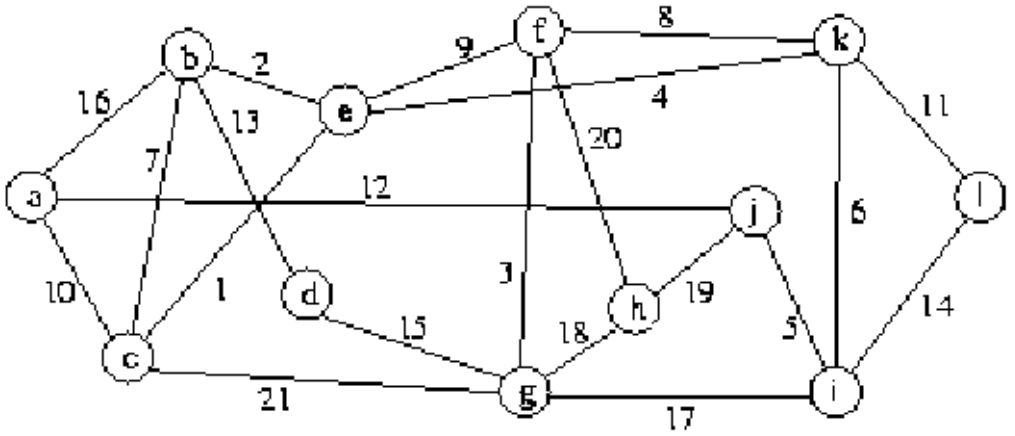
Árvores Geradoras Mínimas

Solução E10

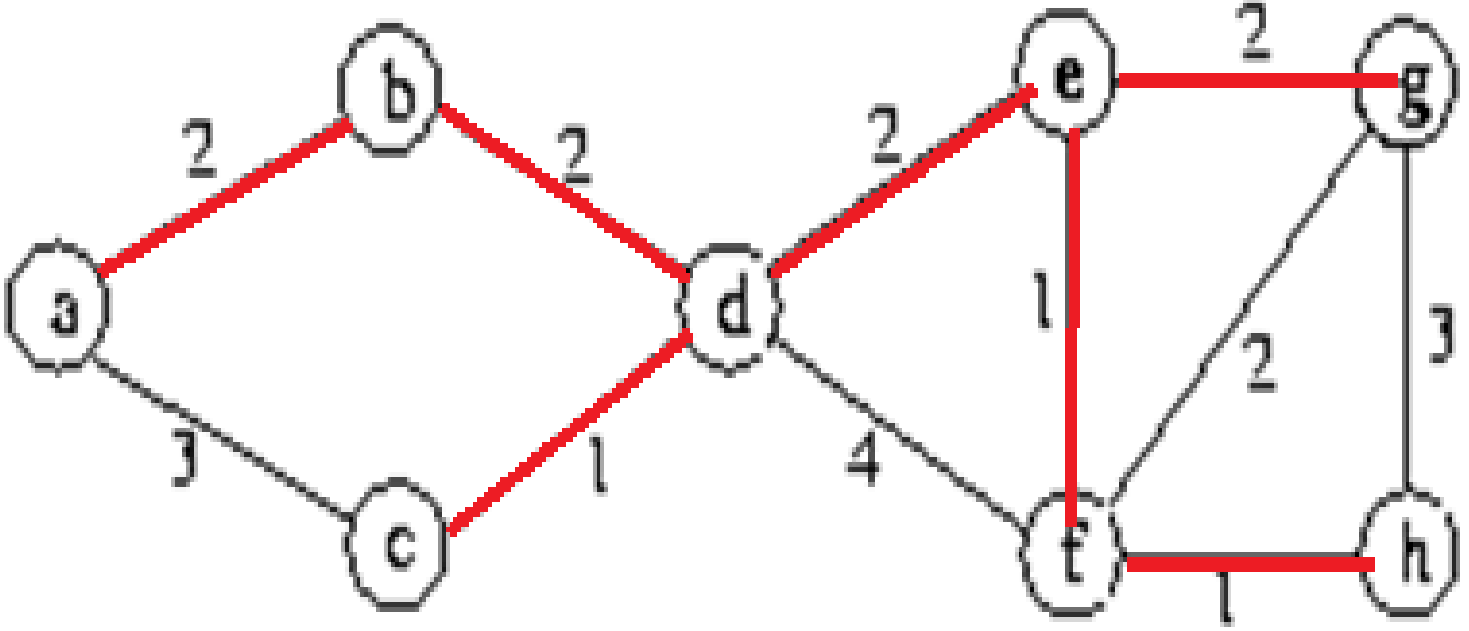
2) Utilize os algoritmos de Kruskal e de Prim para identificar uma árvore geradora mínima em cada um dos grafos ilustrados nas figuras abaixo. Qual é mais adequado em cada pesquisa? Justifique.



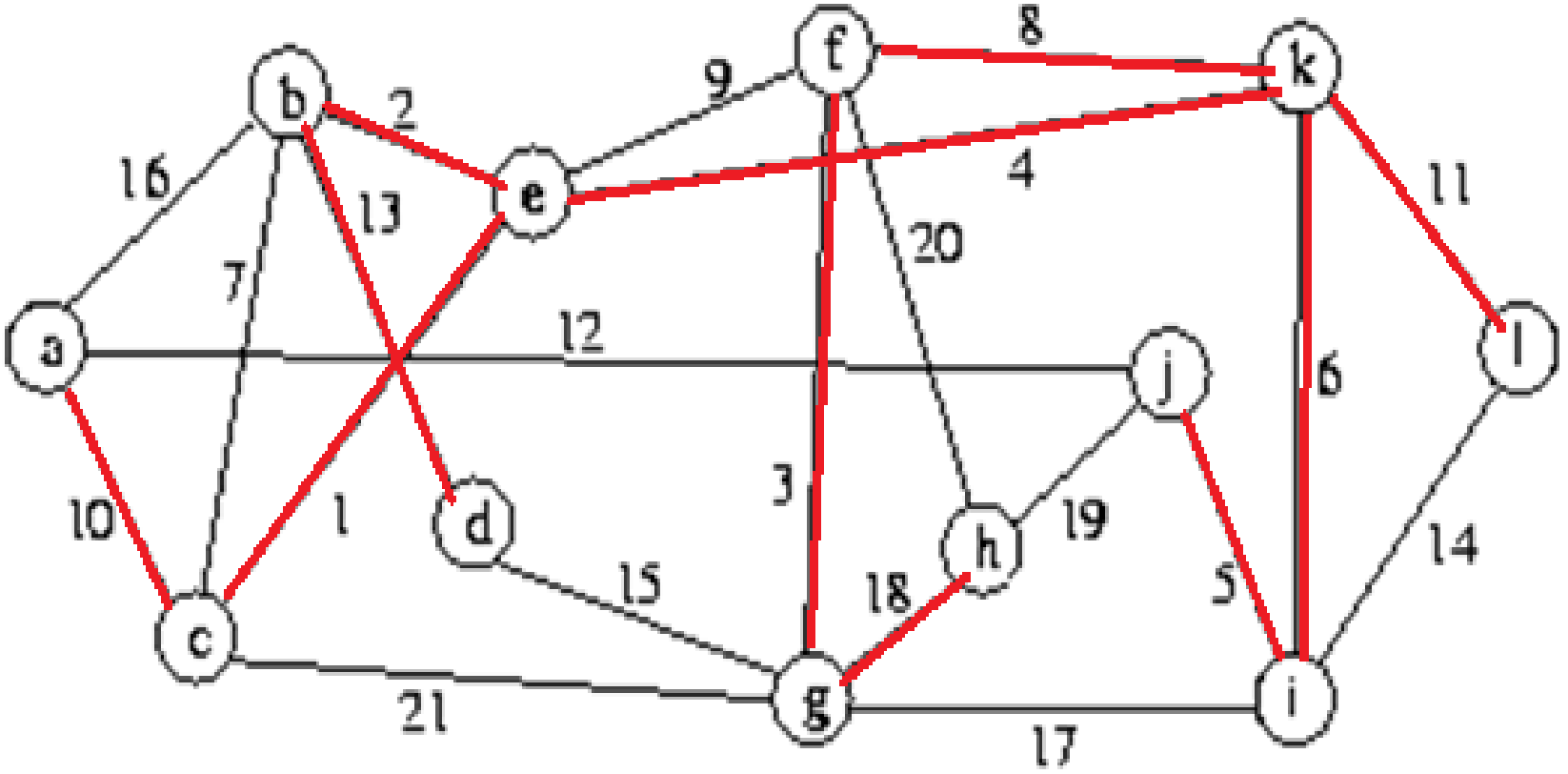
Grafo 1



Grafo 2

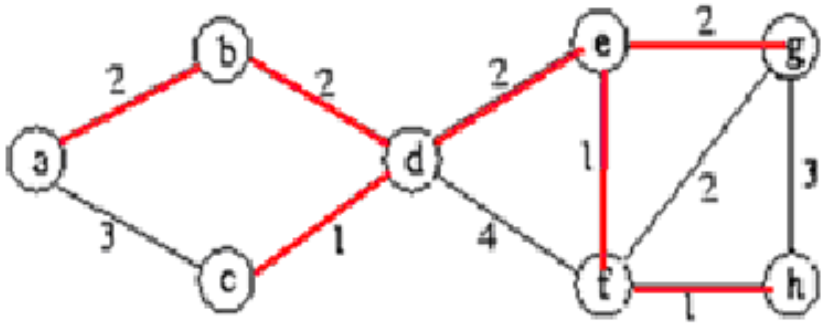


árvore mínima Grafo 1



árvore mínima Grafo 2

3) Seja um grafo com pesos nas arestas e, suponha que um vetor de predecessores pred representa uma árvore geradora de peso P. Escreva um algoritmo que receba pred e um vértice s e devolva o vetor de predecessores de uma árvore geradora com raiz s e peso P.



Algorit – Converte

1. define nova raiz em pred-saída, na posição da raiz e onde pred-entr apontava a raiz.

2. percorre vetor pred-entr a partir desse pred-entr alterando em cada vértice *u*, novo pred será ou quem já o precedia ou quem *antecede*.

Exemplo:

raiz	V:	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
d	pred -entr	b	d	d	**d**	d	e	e	f
h	pred-saída	b	d	d	e	f	h	e	**h**
c	pred-saída'	b	d	**c**	c	d	e	e	f

4) A tabela abaixo mostra as possibilidades de instalação de seções de rede elétrica em um loteamento de sítios (o custo é dado em unidades de R$ 1000). A rede principal, a partir da qual o loteamento será abastecido, passa em frente ao sítio *a*.

um sítio	a	a	a	b	b	c	c	d	d	d	e	e	f	g	g	h	i
outro sítio	b	c	d	d	e	d	f	f	g	h	g	i	h	h	i	j	j
custo	7	5	5,5	7,5	8	6	6,5	5,5	3,5	6,5	4	5	5	5	5	7,5	5,5

Agora, examine a seguinte situação: a empresa responsável pelo loteamento não se obriga, por contrato, a instalar a rede elétrica enquanto pelo menos 80% dos sítios não forem vendidos; até o momento, apenas *a* e *j* encontraram comprador e o proprietário deste último, pessoa influente, pressiona a empresa para que esta leve, imediatamente, energia até o seu sítio. Esta, por seu lado, não julga conveniente aproveitar a ocasião para estender a instalação aos demais sítios.

a) Você é consultado pela empresa, que deseja saber como atender ao proprietário influente da forma mais econômica, ou seja, determinar que seções da rede devem ser construídas de modo a levar energia ao sítio *j*, sem se preocupar com os sítios pelos quais o itinerários escolhidos vão passar;

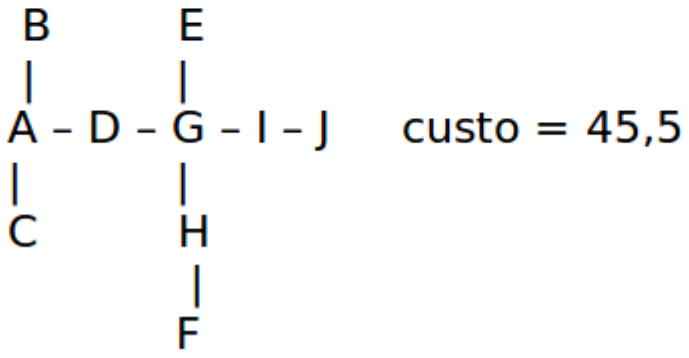
um sítio	a	a	a	b	b	c	c	d	d	d	e	e	f	g	g	h	i
outro sítio	b	c	d	d	e	d	f	f	g	h	g	i	h	h	i	j	j
custo	7	5	5,5	7,5	8	6	6,5	5,5	3,5	6,5	4	5	5	5	5	7,5	5,5

a) atende ligando: A – D – G – I – J custo = 19,5

b) A empresa lhe pede, ainda, que planeje uma rede geral que abasteça todos os sítios, a qual tenha o custo mais baixo possível, levando-se em conta que a linha de *a* para *j* já foi instalada;

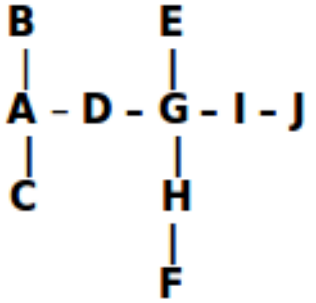
um sítio	a	a	a	b	b	c	c	d	d	d	e	e	f	g	g	h	i
outro sítio	b	c	d	d	e	d	f	f	g	h	g	i	h	h	i	j	j
custo	7	5	5,5	7,5	8	6	6,5	5,5	3,5	6,5	4	5	5	5	5	7,5	5,5

b) MST loteamento:



c) Enfim, a empresa deseja saber se teve prejuízo ao atender à exigência, de modo a poder, eventualmente, compensá-lo ao vender os sítios restantes.

b) MST loteamento:



custo = 45,5

c) Considerando que após 80% vendidos (8 dos 10 seriam ligados) , precisou ligar 50% e portanto a venda desses já resgata o gasto além de viabilizar o investimento pois toda a rede será instalada com menor custo.

Caminhos de custo mínimo

6. Caminhos Mínimos

Introdução

Encontrar o caminho, mais curto possível, entre as cidades de Medina/MG e Bom Jesus/SC.

Solução

→ Um mapa das estradas de rodagem do Brasil, determinar uma rota mais curta?

→ Uma maneira, enumerar todas as rotas possíveis ... haverá milhões de possibilidades !

→ Modelar o problema em um dígrafo ponderado!

Estudando o problema dos *caminhos de custo mínimo* obtemos um resultado eficiente e com menor esforço!

Caminhos mínimos

("Shortest Paths" – SP)

Seja $G = (V, E)$ um grafo pesado orientado ou não-orientado, e $P = v_0, v_1, \ldots v_k$ um caminho nesse grafo. O *peso* do caminho P define-se como

$$w(P) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

Um caminho P do vértice v para o vértice w diz-se um caminho mais curto entre v e w se não existe nenhum caminho de v para w com peso inferior a $w(P)$. P não é necessariamente único.

O nome vem da analogia geográfica, quando os vértices representam *locais* e os pesos *distâncias* entre eles. Outras aplicações: pesos podem representar *custos*, por exemplo de viagens entre locais.

Caminhos mínimos

O problema: dado um grafo G e dois vértices v e w nesse grafo, determinar um caminho mais curto entre eles.

Questão prévia: se construírmos uma *árvore geradora mínima* com origem em v, será o caminho (único) de v para w contido nessa árvore um caminho mais curto? A resposta pode ser encontrada no exemplo anterior . . .

Uma estratégia imediata: força bruta – construír *todos* os caminhos de v para w ($\Rightarrow$ como?); seleccionar o mais curto.

Veremos em seguida um algoritmo muito mais eficiente.

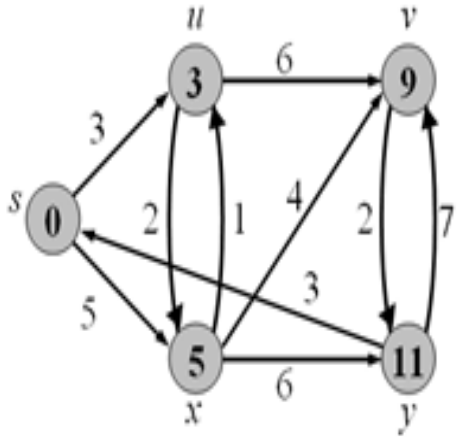
Uma definição necessária: a *distância* $d(x, y)$ do vértice x ao vértice y é o peso de um caminho mais curto de x para y.

Existem algumas variantes deste problema, são elas:

- Menor caminho com destino único: encontrar um caminho mais curto para um vértice destino *v*
- Menor caminho para um par: encontrar um caminho mais curto para um determinado par de vértices *u* e *v*
- Menor caminho para todos os pares: encontrar um caminho mais curto de *u* para *v*, para todos e quaisquer pares *u* e *v*

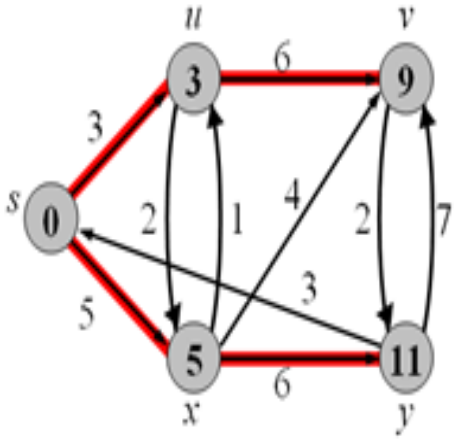
Em algumas instâncias do *problema de menor caminho com uma única origem*, podem existir arestas cujos pesos possuem ***valor negativo***.

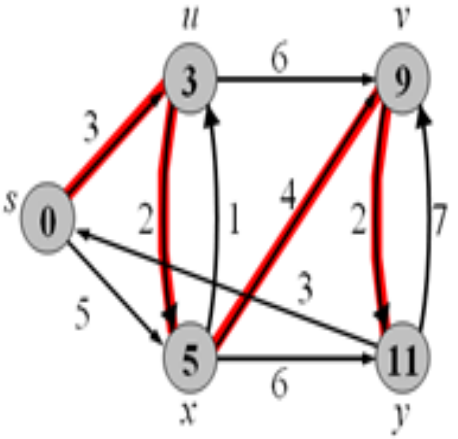
Exemplo de como achar o *caminho mais curto* em um dígrafo, que só tem pesos positivos nos arcos:



Como poderíamos obter a árvore de caminhamentos mínimos partindo da origem s?

Dois possíveis resultados:





Algoritmo de Dijkstra

- Muito semelhante ao algoritmo de PRIM para MSTs.
- Seleciona em cada passo um vértice da orla para acrescentar à árvore que vai construindo.
- O algoritmo vai construindo caminhos cada vez mais longos (no sentido do peso maior) a partir de v, dispostos numa árvore; pára quando encontrar w.
- A grande diferença em relação ao algorito de Prim é o *critério de seleção do arco candidato*.
- A *análise do tempo de execução* é análoga.

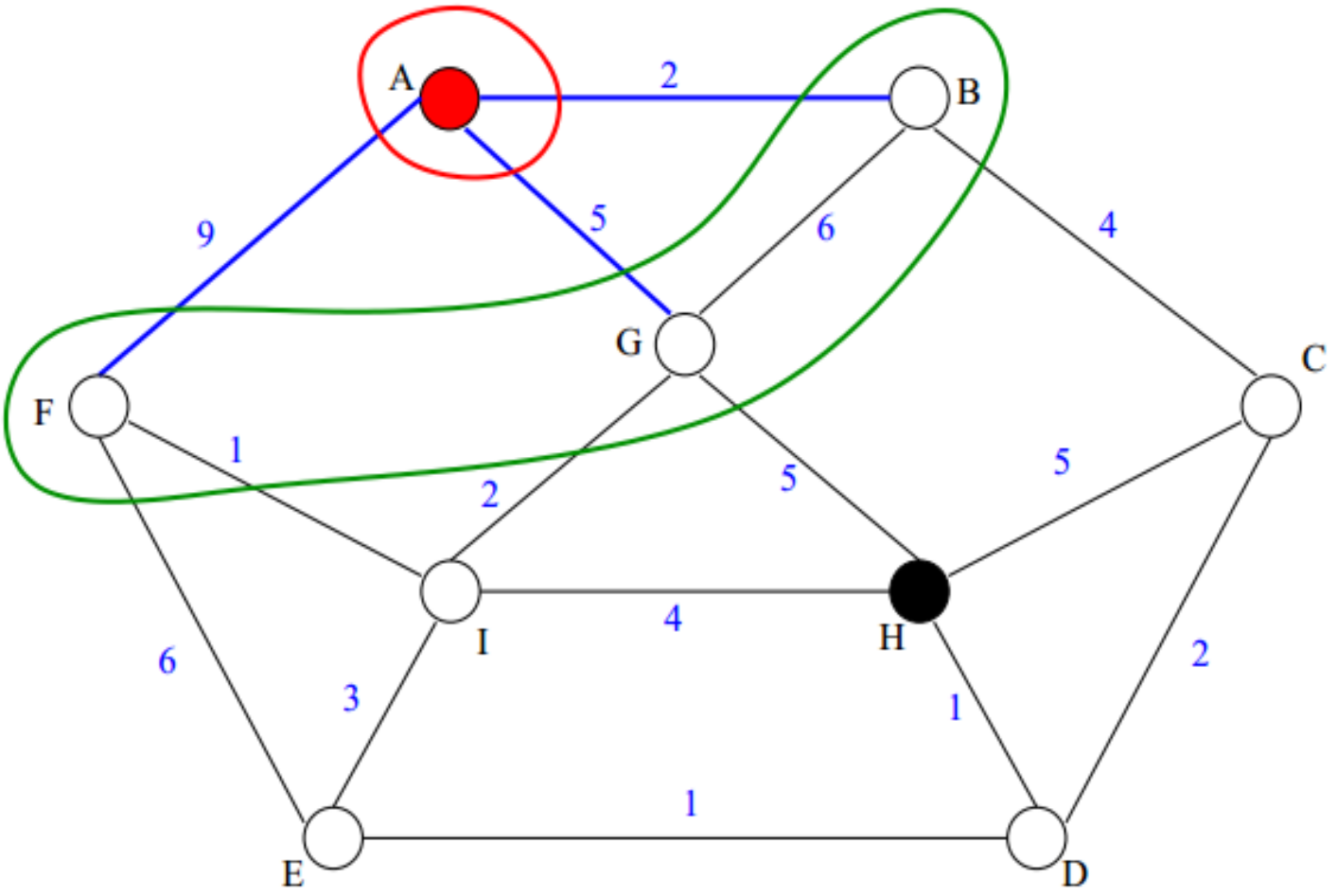
Algoritmo de Dijkstra – Arcos Candidatos

- Para cada nó z na orla, existe um caminho mais curto $v, v_1, \ldots v_k$ na árvore construída, tal que $(v_k, z) \in E$.

- Se existirem vários caminhos $v, v_1, \ldots v_k$ e arco (v_k, z) nas condições anteriores, o arco candidato (único) de z será aquele para o qual $d(v, v_k) + w(v_k, z)$ fôr mínimo.

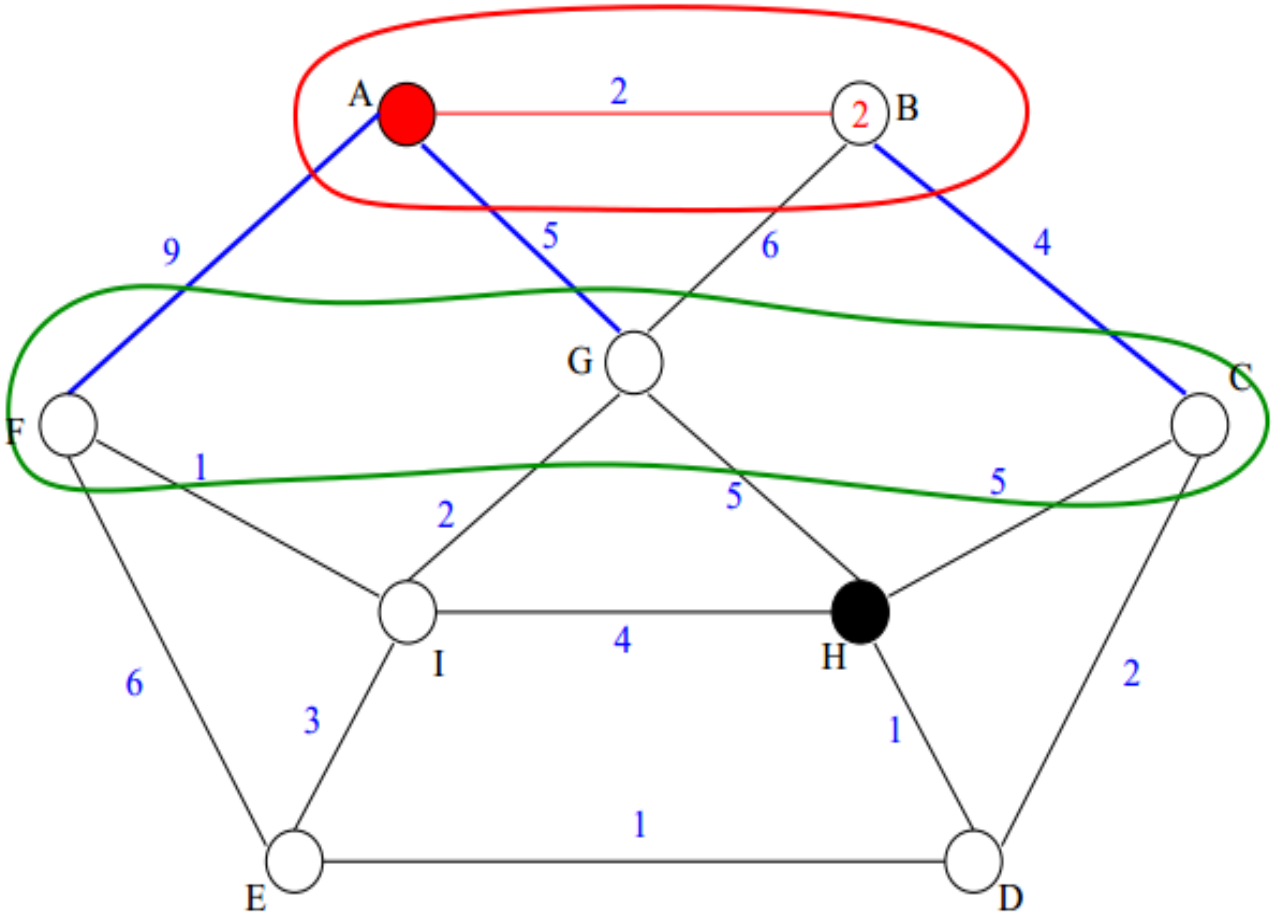
- Em cada passo, o algoritmo selecciona um vértice da orla para acrescentar à árvore. Este será o vértice z com valor $d(v, v_k) + w(v_k, z)$ mais baixo.

Algoritmo de Dijkstra – Exemplo



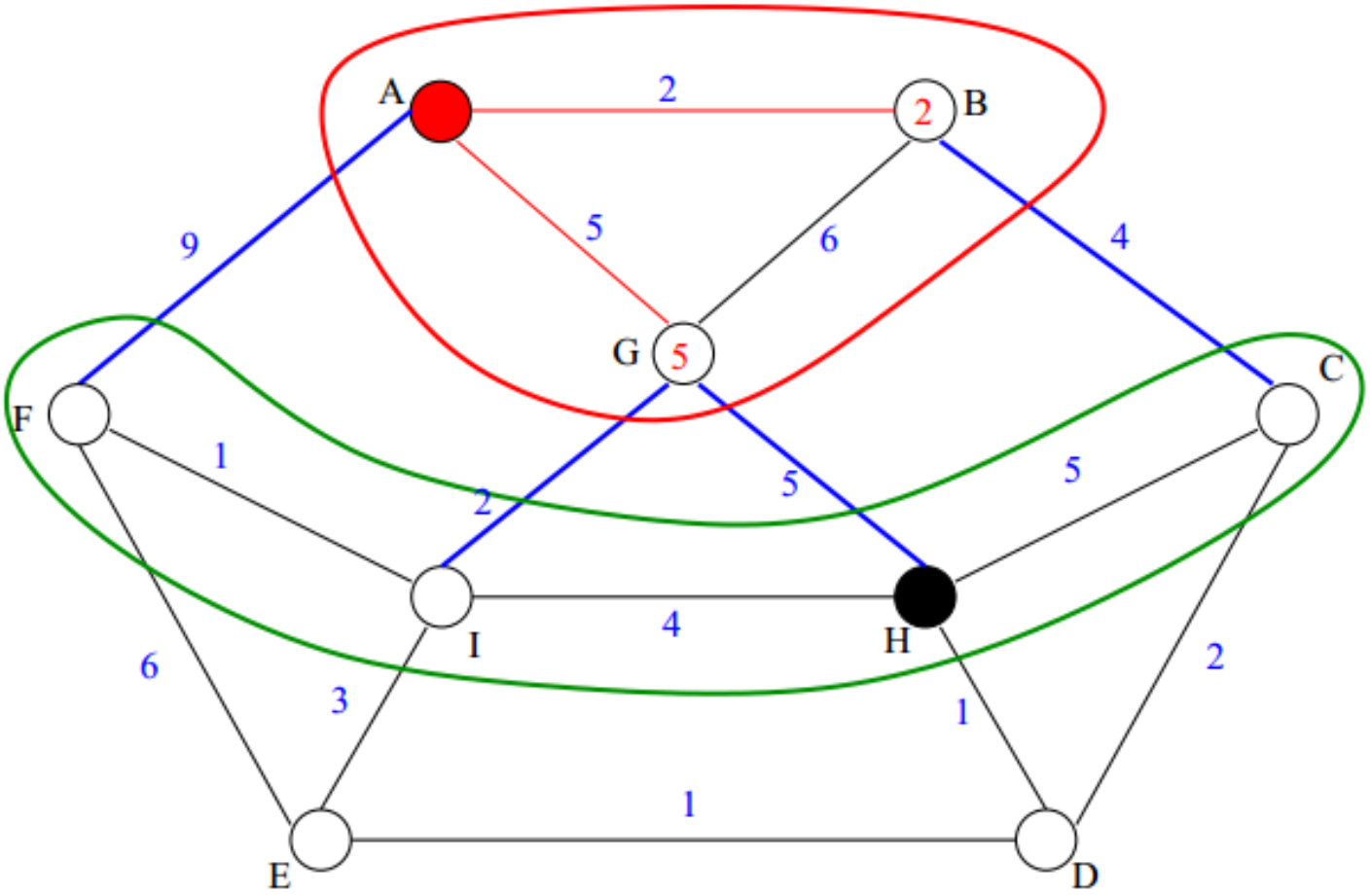
$$d(A,A)+w(A,B)=2; \quad d(A,A)+w(A,F)=9; \quad d(A,A)+w(A,G)=5.$$

Algoritmo de Dijkstra – Exemplo



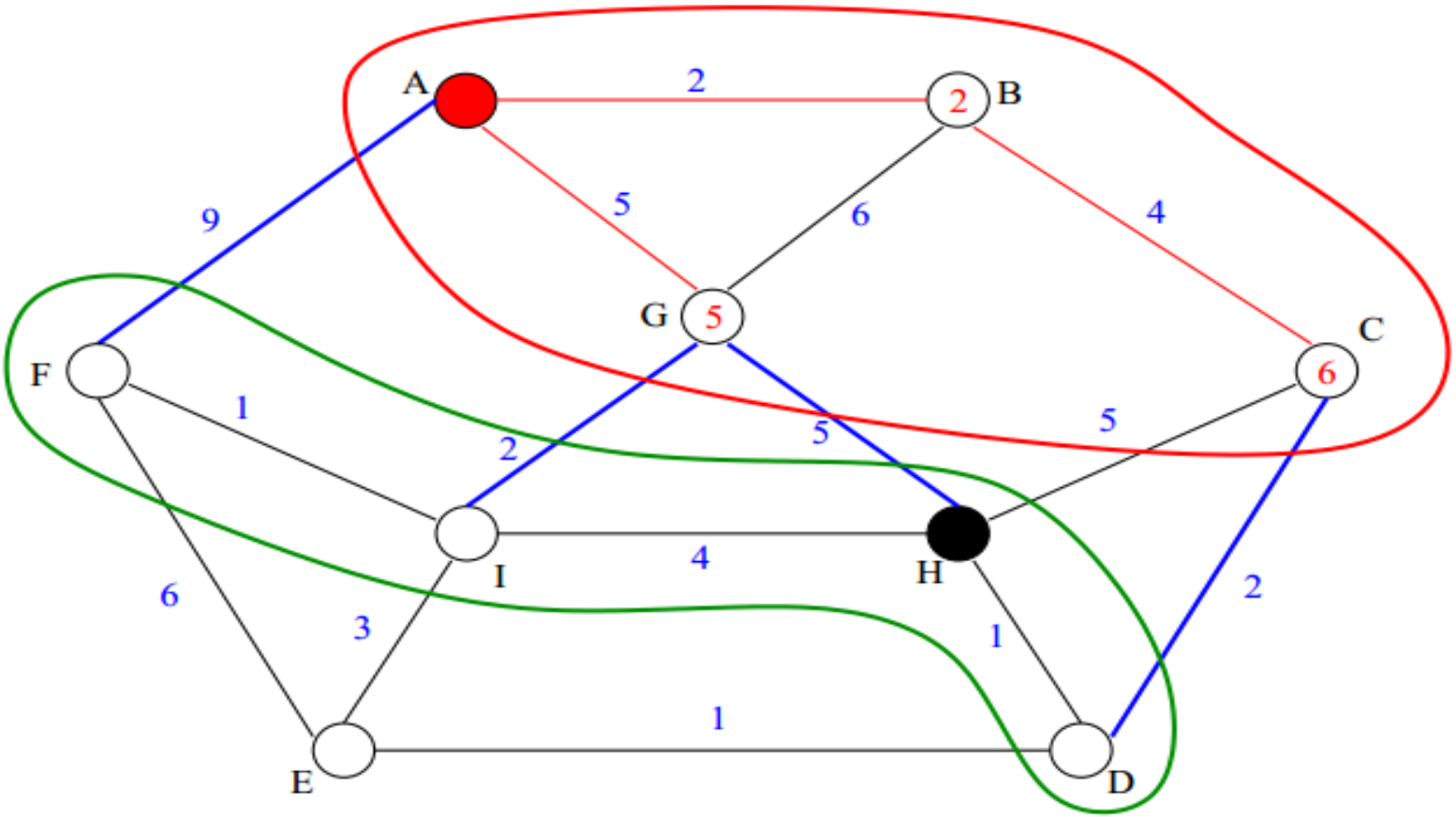
$$d(A,B) + w(B,C) = 6; \quad d(A,A) + w(A,F) = 9; \quad d(A,A) + w(A,G) = 5.$$

Algoritmo de Dijkstra – Exemplo



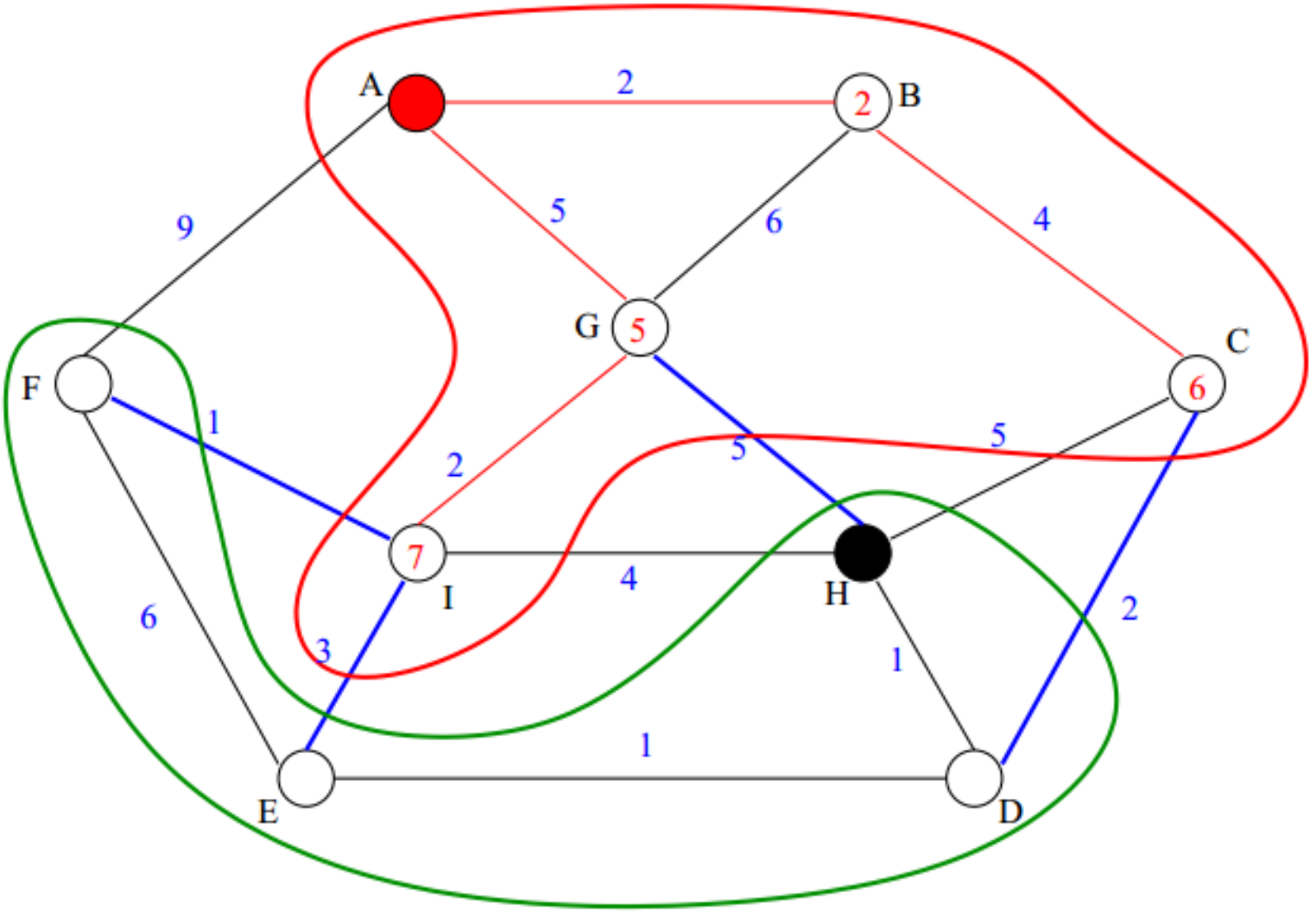
$$d(A,B) + w(B,C) = 6; \quad d(A,A) + w(A,F) = 9;$$
$$d(A,G) + w(G,H) = 10; \quad d(A,G) + w(G,I) = 7.$$

Algoritmo de Dijkstra – Exemplo

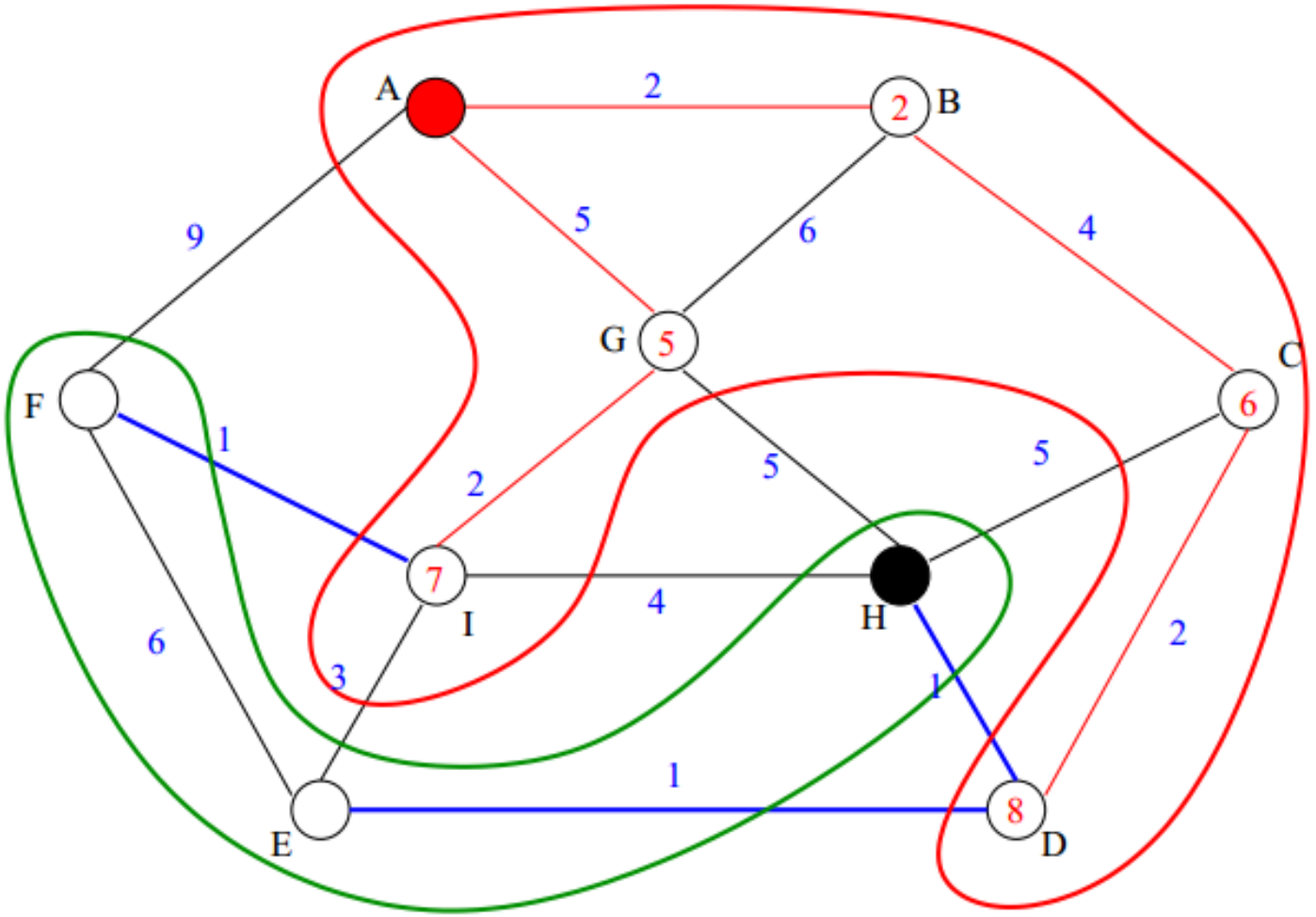


$$d(A,C) + w(C,D) = 8; \quad d(A,A) + w(A,F) = 9;$$
$$d(A,G) + w(G,H) = 10; \quad d(A,G) + w(G,I) = 7.$$

Algoritmo de Dijkstra – Exemplo

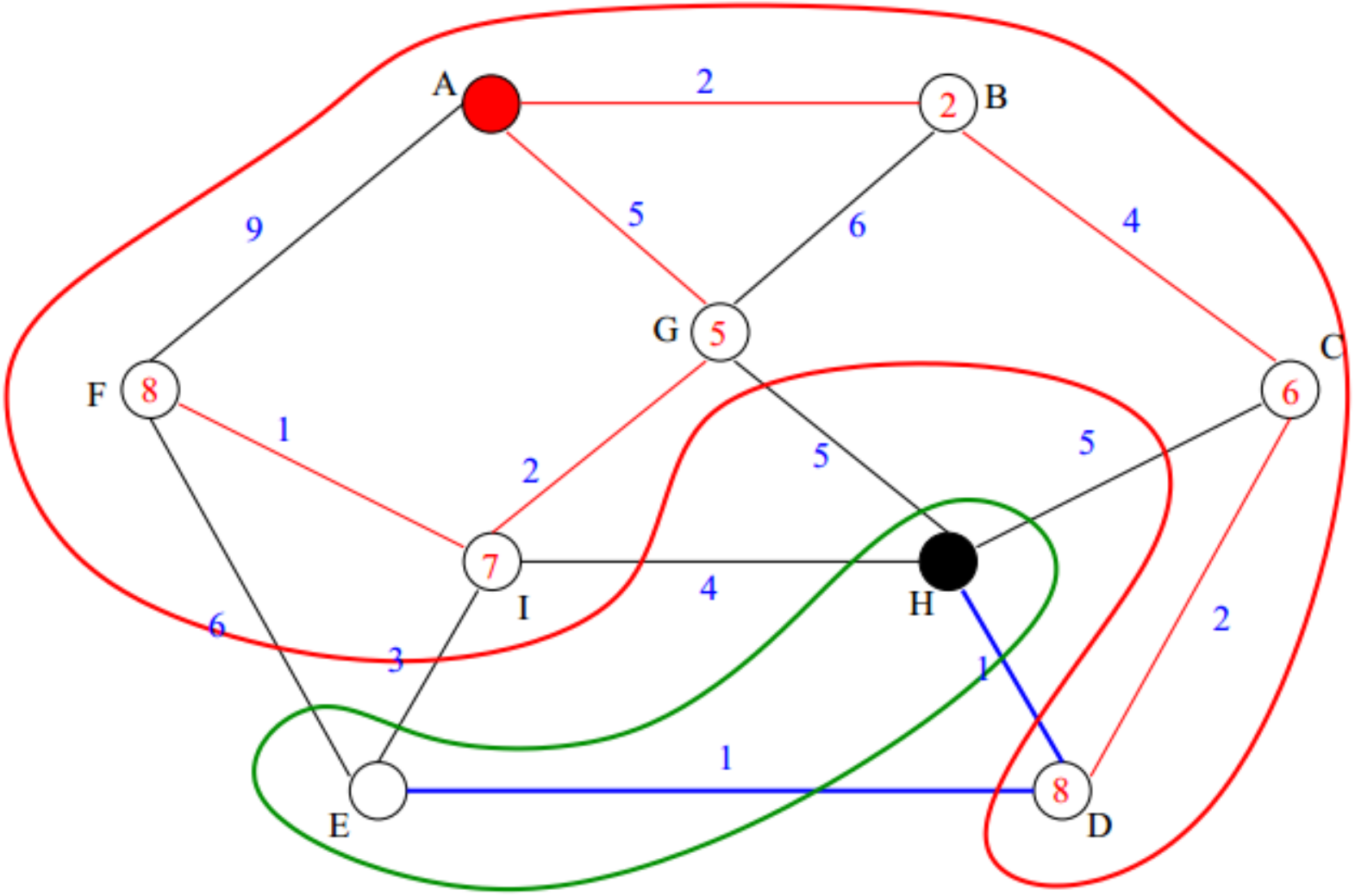


Algoritmo de Dijkstra – Exemplo

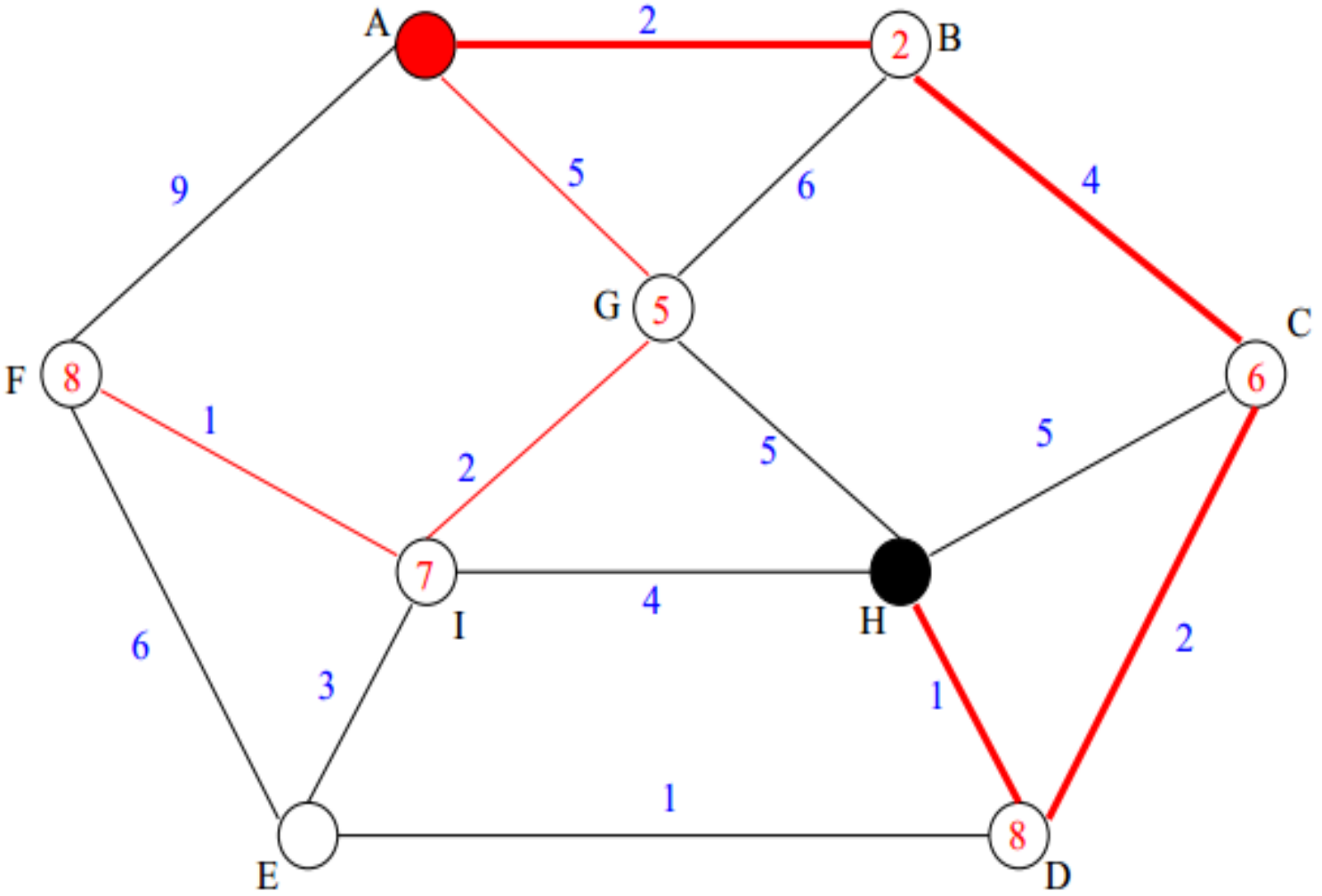


Houve uma alteração do arco candidato de H!

Algoritmo de Dijkstra – Exemplo

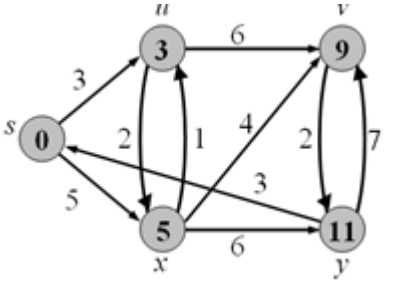


Algoritmo de Dijkstra – Exemplo



Algumas definições:

- Em cada vértice *v*, mantemos um atributo *d[v]*, que é o limite superior do peso de um menor caminho da origem *s* até o vértice *v*, por exemplo, d[s] = 0, d[y] = 11.
- Chamamos *d[v]* de estimativa de menor caminho
- Iniciamos as estimativas de menor caminho e predecessores através do seguinte procedimento:



IniciaOrigemÚnica(*G*,*s*)

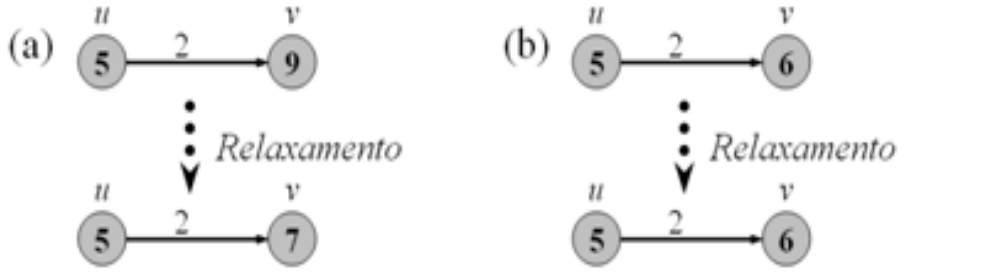
```
1  para cada vértice v em G faça
2        d[v] ← ∞
3        pred[v] ← NIL
4  d[s] ← 0
```

O algoritmo consiste em testar se é possível identificar um menor caminho para um vértice *v* passando pelo vértice *u* (processo de relaxamento de uma aresta (u,v)), e caso afirmativo atualizar *d[v]* e *pred[v]*.

O passo de relaxamento pode reduzir a estimativa de menor caminho, e atualizar o predecessor de um determinado vértice, o procedimento a seguir realiza o relaxamento:

```
Relaxa(u,v,w)
1  se d[v] > d[u] + w(u,v)
2      então  d[v] ← d[u] + w(u,v)
3             pred[v] ← u
```

Exemplos:



ocorre relaxamento

não ocorre

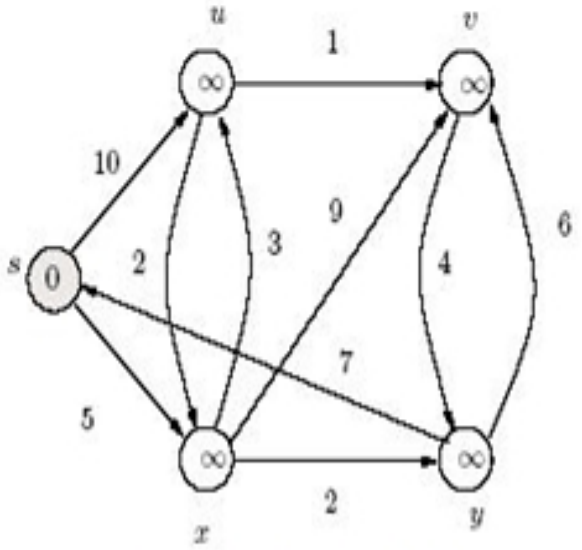
Algoritmo de Dijkstra

Vemos a seguir o algoritmo de Dijkstra que resolve o *problema de caminhos mínimos com uma única fonte* para um dígrafo $G = (V,E)$, quando não há arcos de peso negativo, ou seja, $w(u, v) \geq 0$ para qualquer arco $(u, v) \in E$.

Este algoritmo mantém um conjunto S de vértices, através de uma fila Q, onde

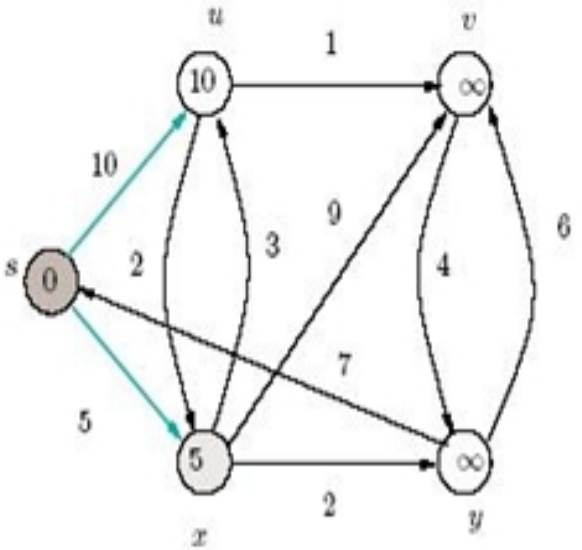
```
Dijsktra(G, w, s)
 1. IniciaOrigemÚnica(G,s)
 2. Q ← Cria-Fila( )
 3. enquanto Q ≠ ∅ faça
 4.       u ← Retire-Mínimo(Q)
 5.       para cada v em Adj[u]faça
 6.             Relaxa(u,v,w)
 7. devolve pred
```

O comando **Crie-Fila()** cria uma "fila" com todos os vértices. A prioridade dos vértices na fila é dada pelo vetor *d*. O comando **Retire-Mínimo**(Q) retira de *Q* um vértice *u* para o qual *d[u]* é mínimo. A alteração do valor de d[v] no procedimento **Relaxa( )** pode afetar a estrutura da fila; isso deve ser levado em conta quando a fila for implementada.



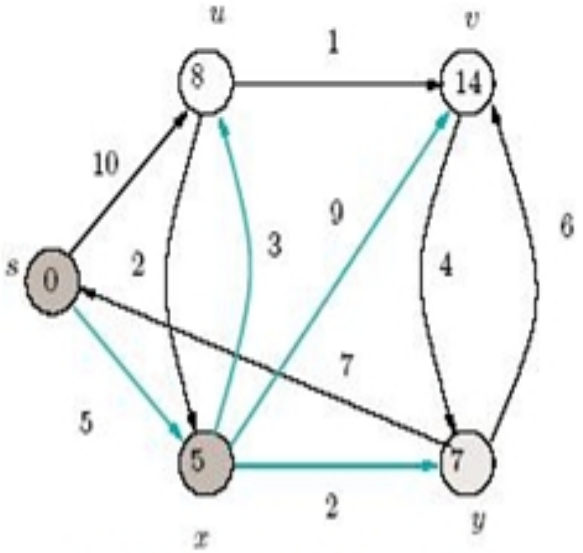
i) execução linhas 1-2

Q: s, u, v, x, y



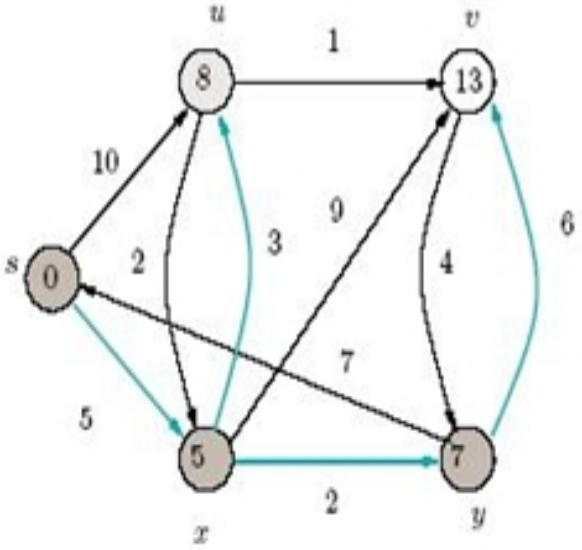
ii) 1ª iteração (linhas 4-6)

Q: x, u, v, y



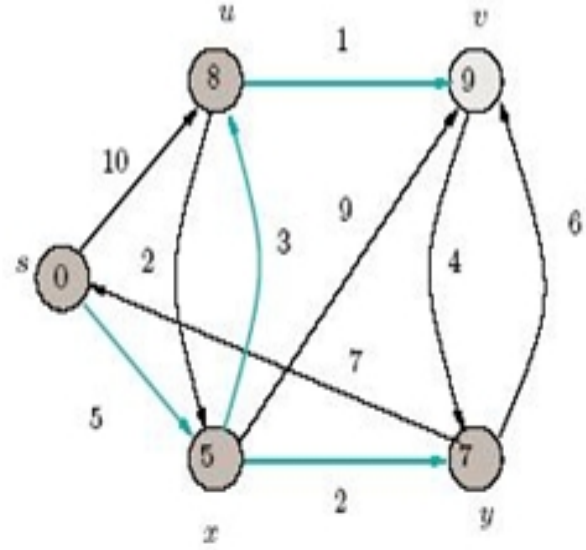
iii) 2ª iteração (linhas 4-6)

Q: y, u, v



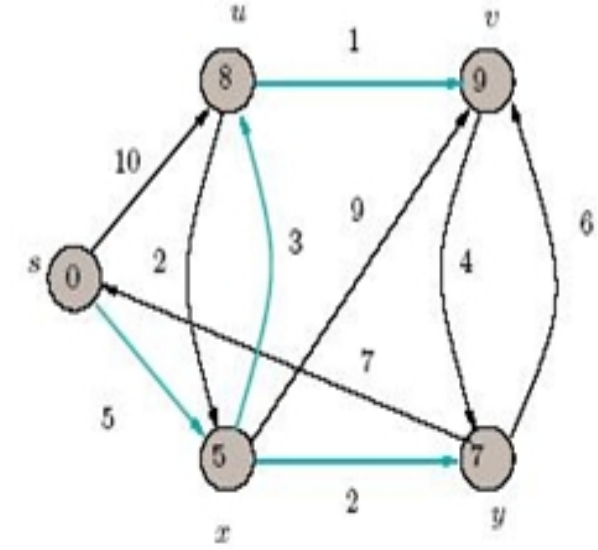
iv) 3ª iteração (linhas 4-6)

Q: u, v



v) 4ª iteração (linhas 4-6)

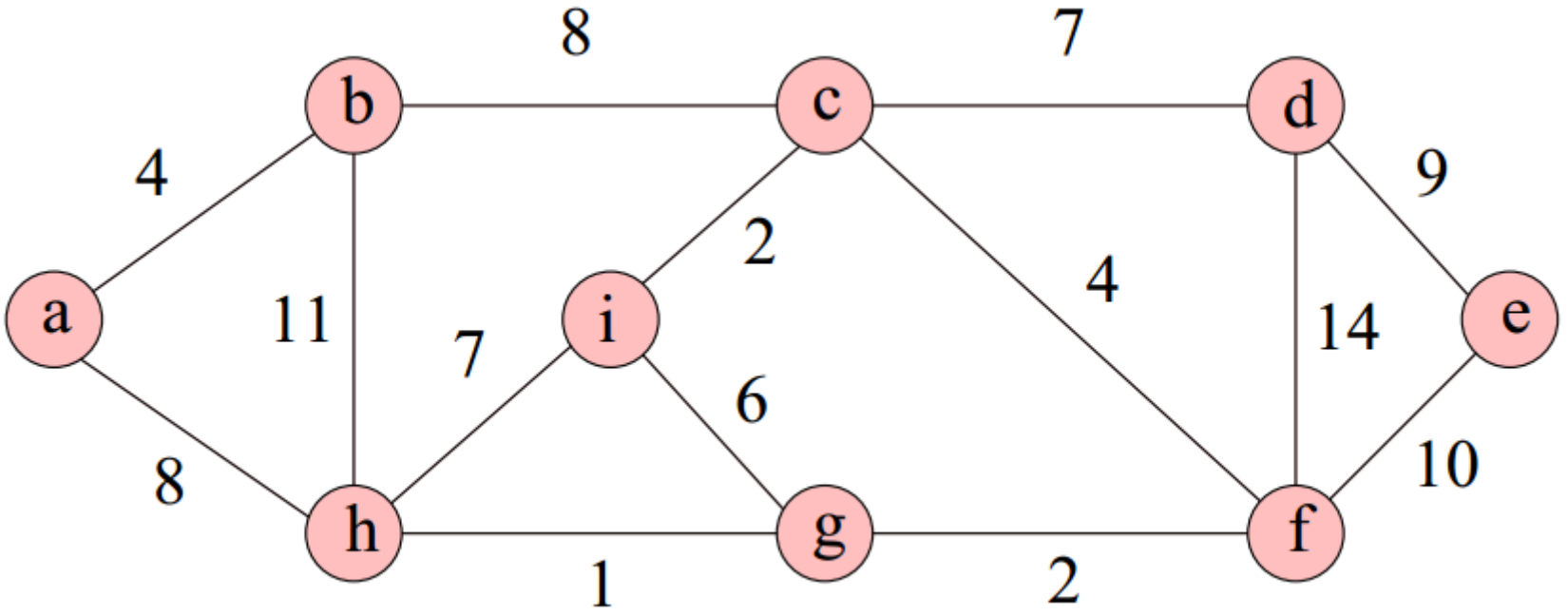
Q: v



vi) 5ª iteração (linhas 4-6)

Q: ∅

+ exemplo:



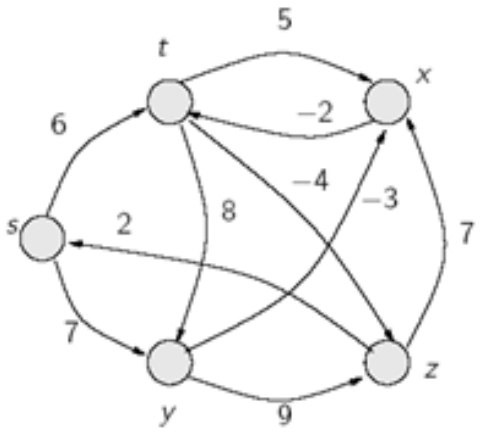
Algoritmo de Bellman - Ford

O algoritmo *Bellman-Ford* resolve o *problema do menor caminho com uma única origem* de uma forma mais genérica, incluindo arestas com peso negativo.

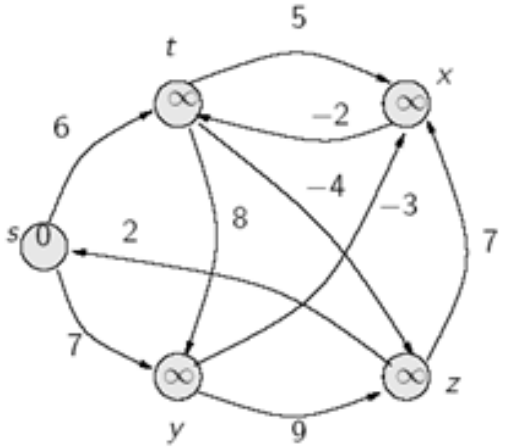
O algoritmo indica se um ciclo de comprimento negativo alcançável a partir de *s* foi ou não encontrado. Em caso afirmativo não há solução e, em caso contrário - onde não há ciclo negativo alcançável a partir de *s* - o algoritmo produz a árvore de caminhos mínimos com raiz em s e os seus respectivos comprimentos (pesos).

```
Bellman-Ford (G, w)
 1. IniciaOrigemÚnica(G,s)
 2.  para i ← 1 até |V| -1 faça          (|V| = n vértices)
 3.        para cada (u,v) ∈ E faça      (E = conjunto de arcos)
 4.              Relaxa(u,v,w)
 5.  para cada (u,v) ∈ E faça            (verifica se existe ciclo negativo)
 6.       se d[v] > d[u] + w(u,v)
 7.             então devolve 0          (tem ciclo negativo alcançavel a partir de s)
 8.  devolve 1                           (não tem ciclo negativo)
```

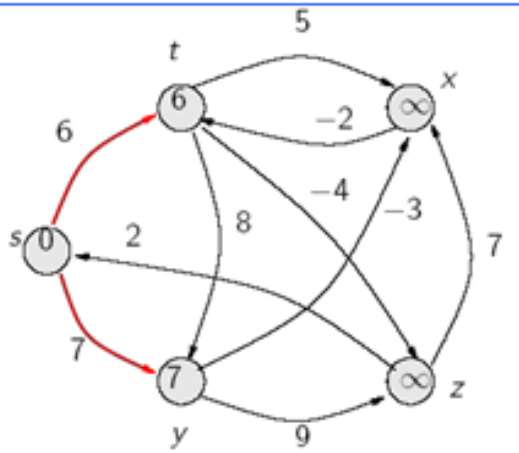
As estimativas de caminhos mais curtos são mostradas dentro dos vértices, e a origem dos arcos "vermelhos" indica os predecessores.



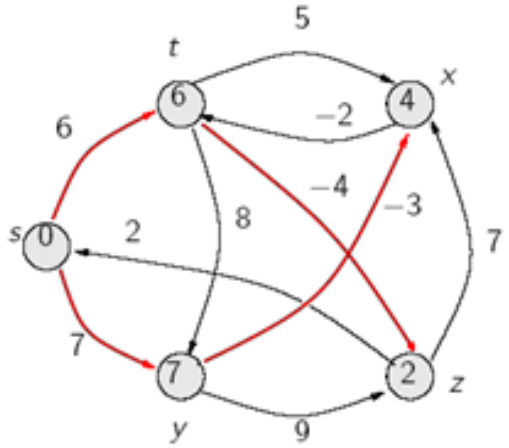
dígrafo



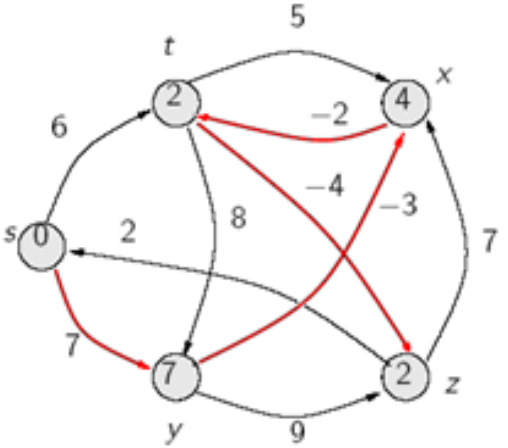
i) inicialização (linha 1)



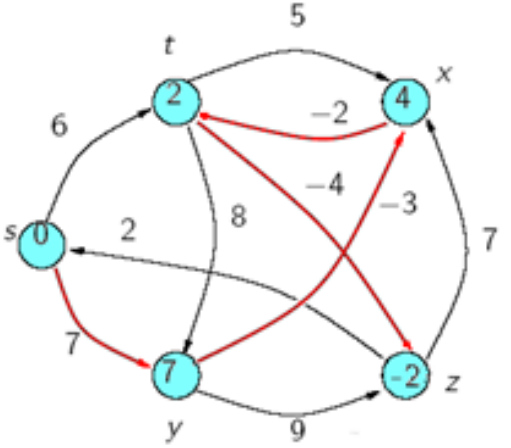
ii) 1ª iteração (i = 1, linhas 2-4)



iii) 2ª iteração (i = 2, linhas 2-4)

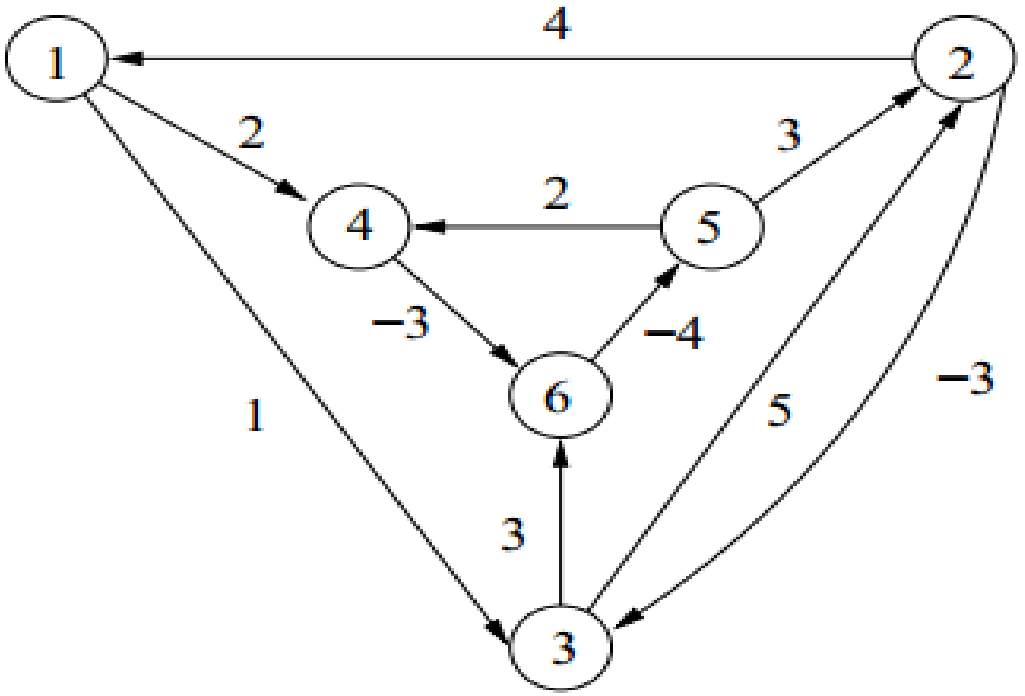


iv) 3ª iteração (i = 3, linhas 2-4)

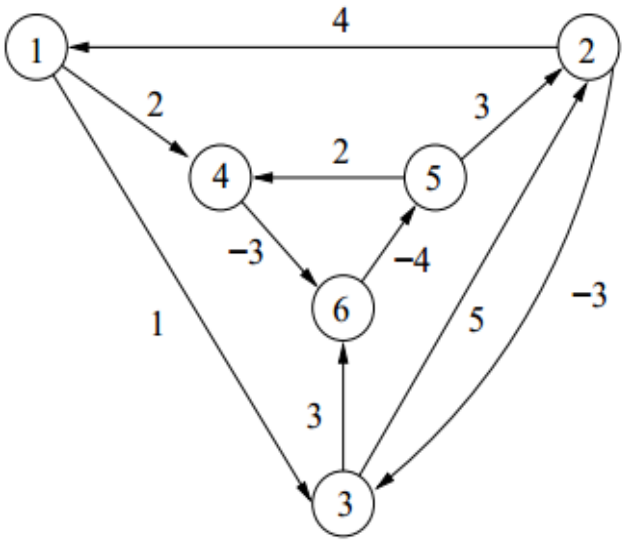


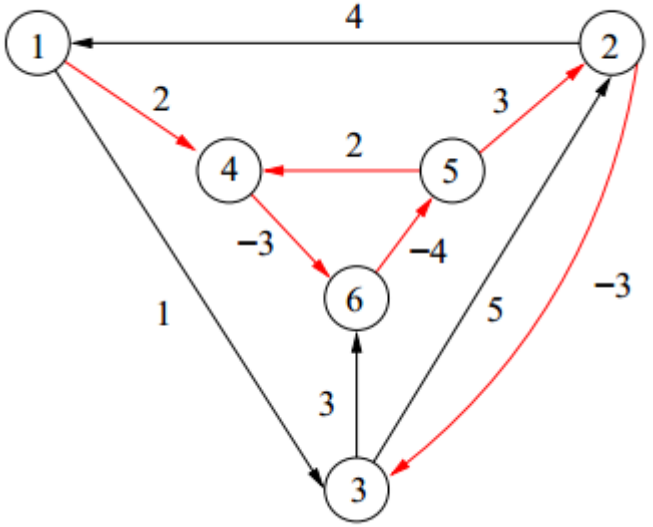
v) iteração final (i = 4, linhas 2- 4)

+ exemplo:



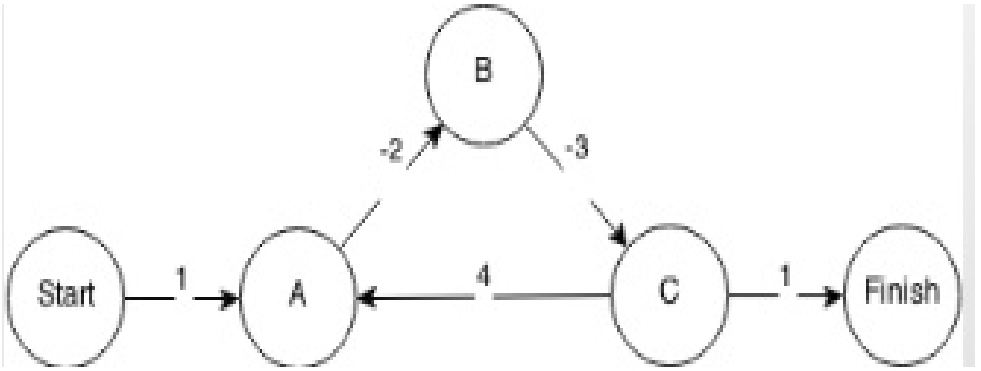
+ exemplo: … solução



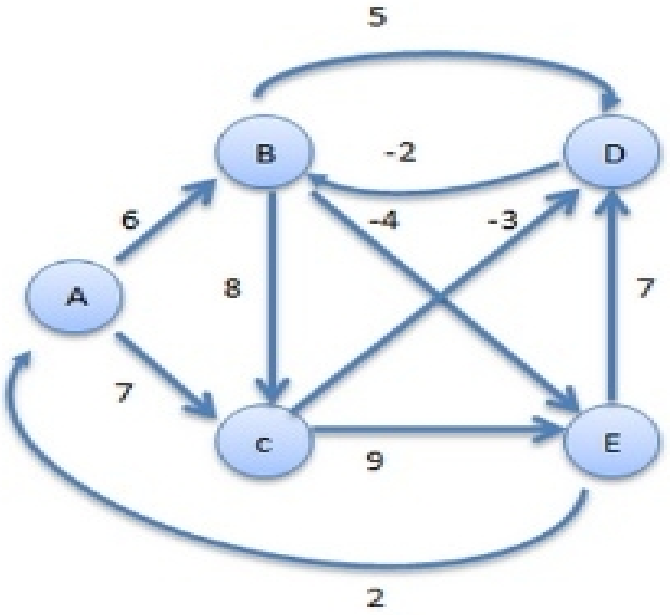


Checagem de Ciclos Negativos

Depois de executado (V-1) a técnica do relaxamento, precisamos verificar se o grafo não contém um ciclo negativo.

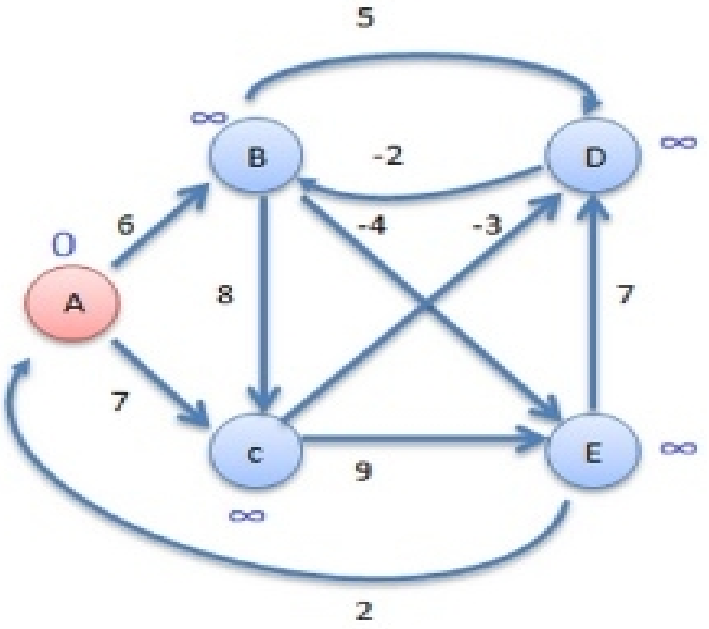


Exemplo:

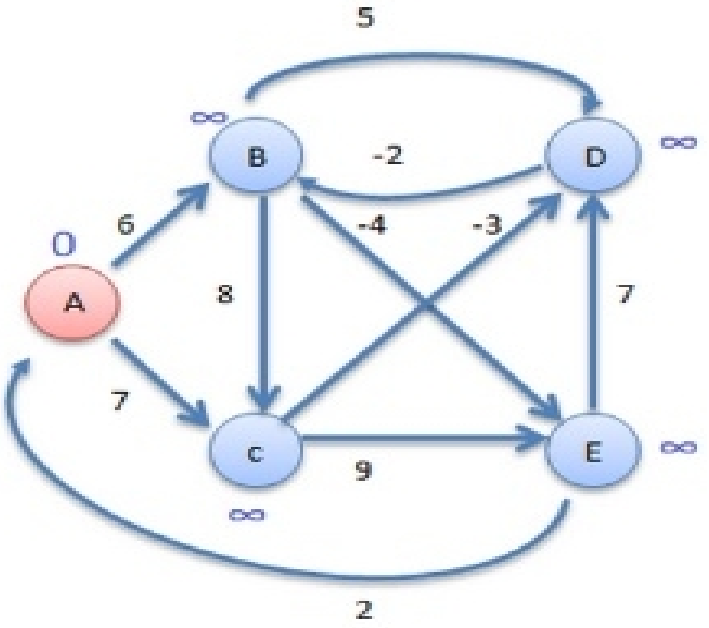


Passo 1:
Definimos o vértice A como fonte:

Vértices:	A	B	C	D	E
Distância	0	6	7	∞	∞
Distância de:	A	A	A		

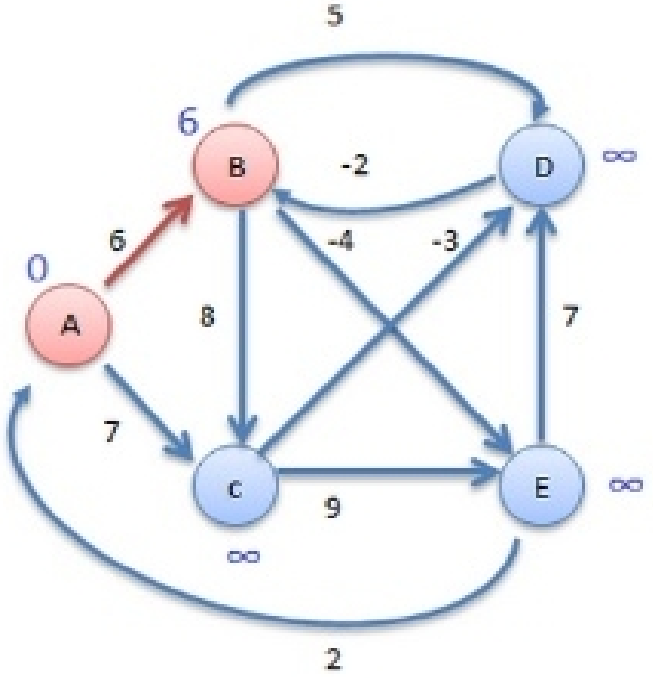


Exemplo:

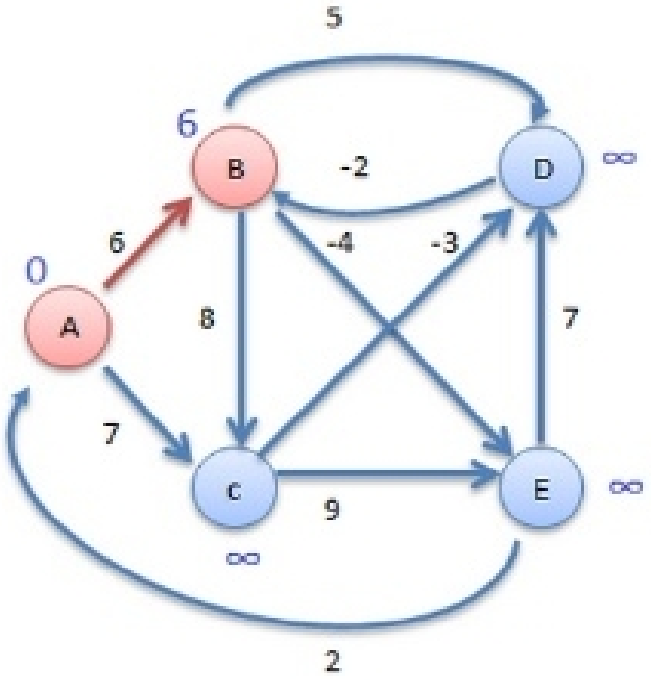


Passo 2:
Agora o vértice B como fonte:

Vértices:	A	B	C	D	E
Distância	0	6	7	11	2
Distância de:	A	A	A	B	B

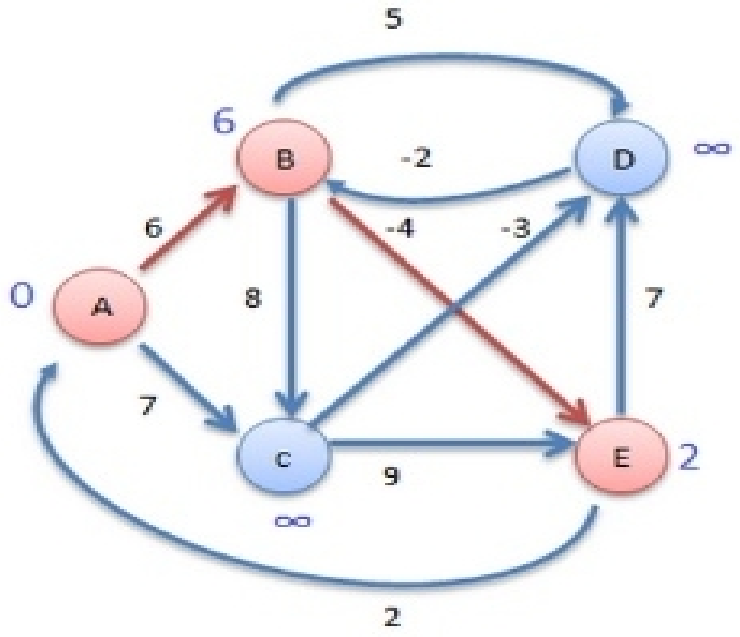


Exemplo:

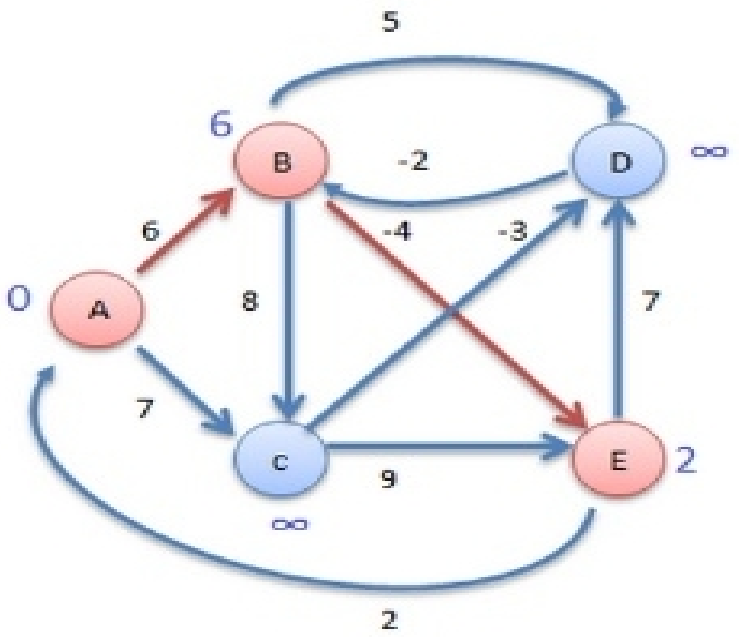


Passo 3:
Agora o vértice E como fonte:

Vértices:	A	B	C	D	E
Distância	0	6	7	9	2
Distância de:	A	A	A	E	B

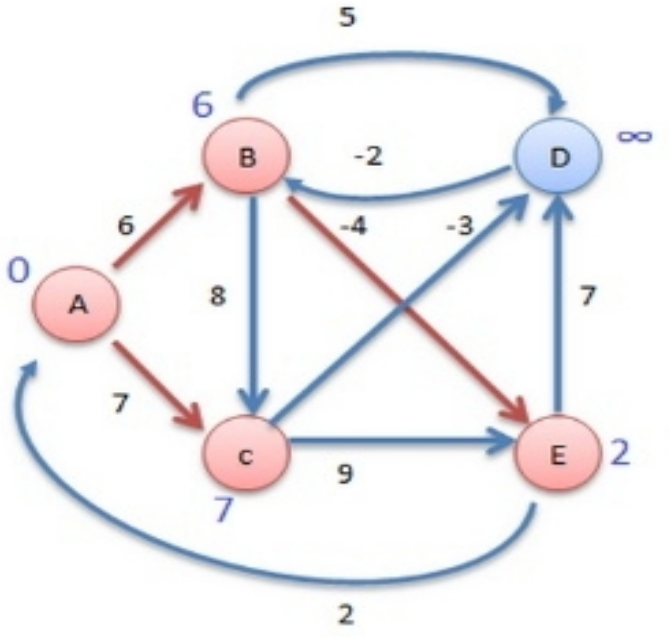


Exemplo:

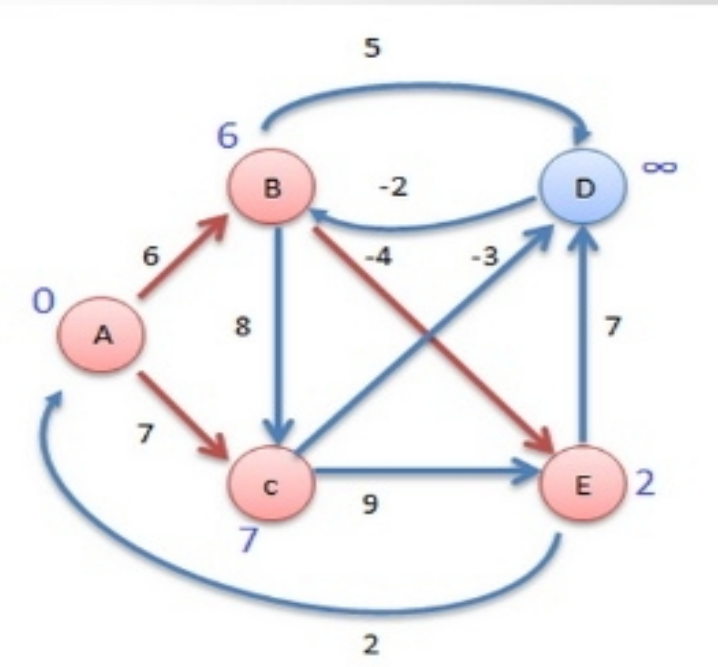


Passo 4:
Agora o vértice C como fonte:

Vértices:	A	B	C	D	E
Distância	0	6	7	4	2
Distância de:	A	A	A	C	B

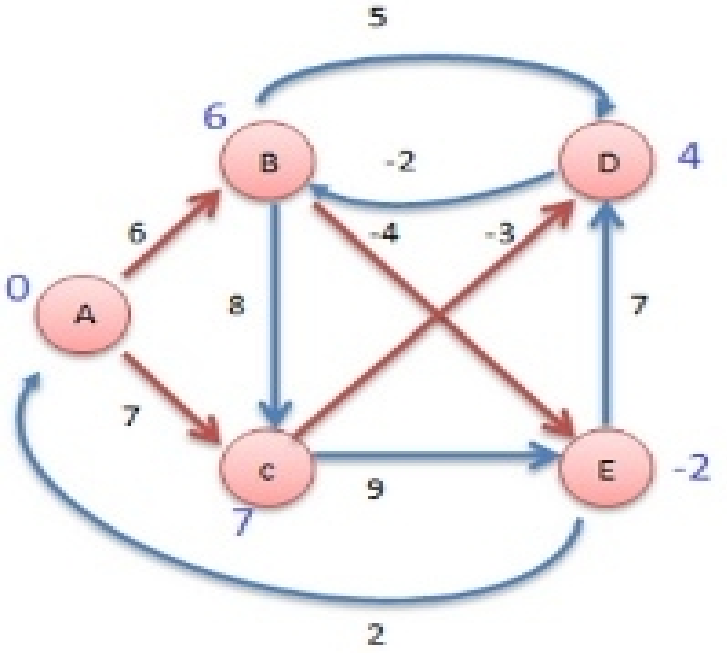


Exemplo:

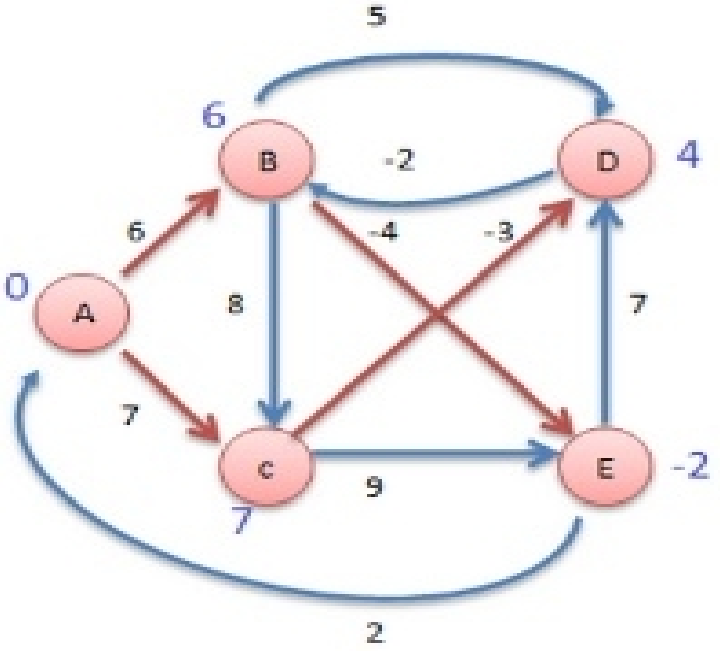


Passo 5:
Agora o vértice D como fonte:

Vértices:	A	B	C	D	E
Distância	0	2	7	4	2
Distância de:	A	D	A	C	B



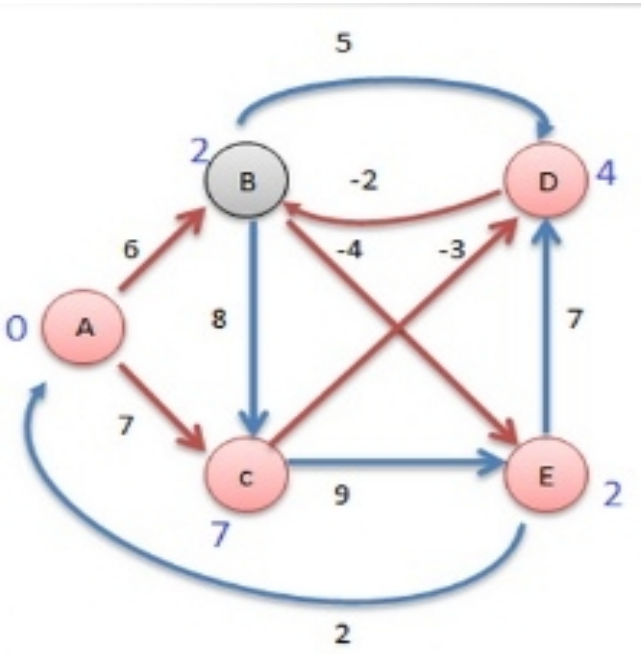
Exemplo:



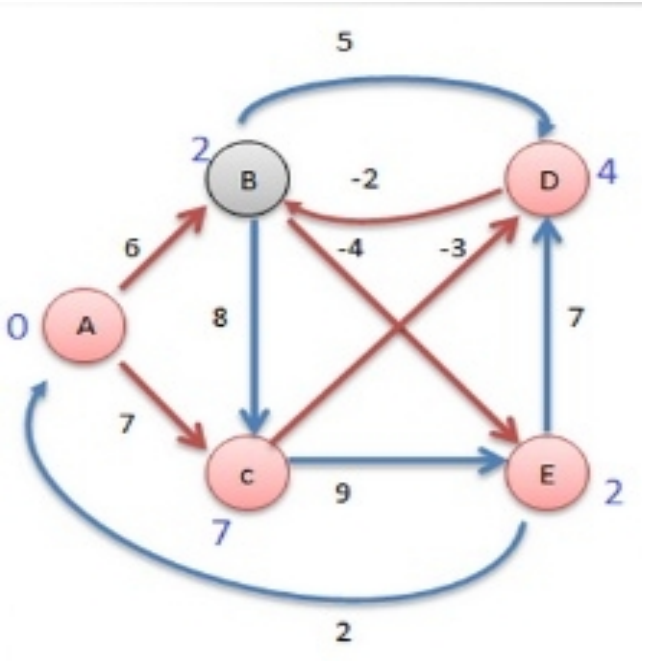
2ªiteração

Passo 6:
Agora o vértice B como fonte:

Vértices:	A	B	C	D	E
Distância	0	2	7	4	-2
Distância de:	A	D	A	C	B



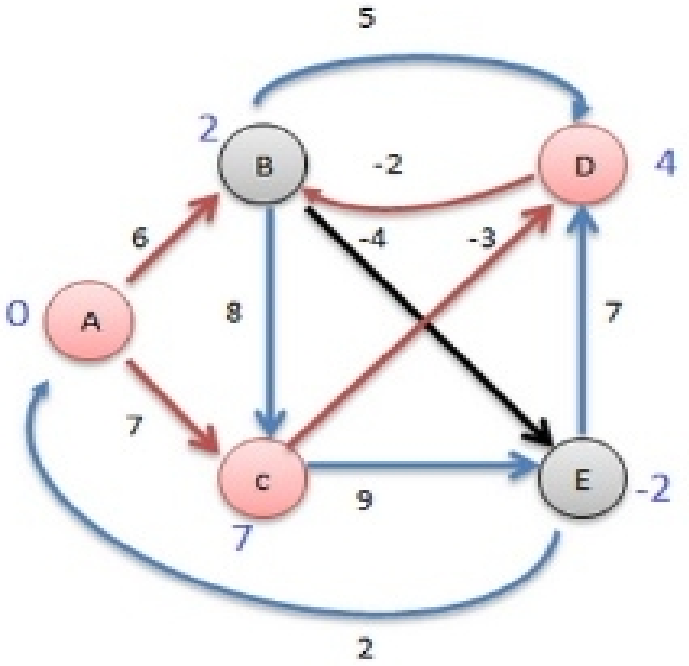
Exemplo:



3ªiteração

Passo 7:
Agora o vértice E como fonte:

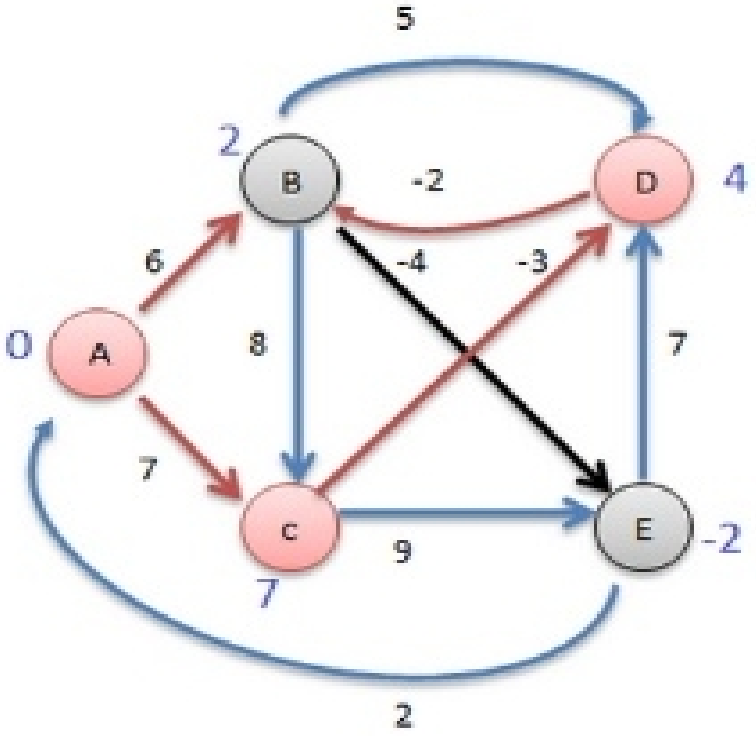
Vértices:	A	B	C	D	E
Distância	0	2	7	4	-2
Distância de:	A	D	A	C	B



Exemplo:

Conclusão:

Vértice	Distância de A
A	0
B	2
C	7
D	4
E	-2



caminhos mínimos entre todos os pares (u,v)

Algoritmo de Floyd-Warshall

O algoritmo *Floyd-Warshall* resolve o problema de calcular o caminho mais curto entre todos os pares de vértices em um dígrafo $G = (V, E)$.

Arestas com *peso negativo* podem estar presentes, mas assumimos por conveniência que não há ciclos com pesos negativos. (O algoritmo é capaz de detectar tais ocorrências.)

Caminhos mínimos entre todos os pares:

- Consideramos o problema de encontrar o caminho mais curto entre cada par de vértices de um grafo $G = (V, E)$.
- Este problema pode surgir na construção de uma tabela de distâncias entre cidades de um atlas.
- É dado um dígrafo $G = (V, E)$ com função peso das arestas dada por $w : E \to R$.
- Problema: para cada par $u, v \in V$, encontre o caminho de menor peso de *u* para *v*.

Algoritmo de Floyd-Warshall

Métodos de solução

- Podemos resolver o "problema de caminhos mínimos entre todos os pares de vértices" resolvendo |V| = n "problemas de caminhos mínimos com uma fonte."
- Se todos os pesos são não negativos, podemos utilizar o algoritmo de Dijkstra em cada vértice.
- Se há arestas com pesos negativos, precisaremos utilizar o algoritmo de Bellman-Ford, que deve ser executado a partir de cada vértice: um processo mais lento se comparado com Dijkstra.
- Temos uma solução mais eficiente: algoritmo de Floyd-Warshall:

Algoritmo de Floyd-Warshall

Definições

- Comparando com os algoritmos anteriores, que utilizaram lista de adjacência, representaremos o dígrafo por meio de uma matriz de adjacência.
- Por conveniência, assumiremos que os vértices são numerados 1, 2, . . . , $|V| = n$.
- Ciclos com peso negativo podem ocorrer, mas assumiremos que o grafo não contém ciclo negativo.
- A saída tabulada do algoritmo é uma matriz $D = [d_{ij}]$.
- Cada entrada d_{ij} contém o comprimento do caminho mais curto do vértice i ao vértice j, ou seja, $d_{ij} = \delta(i, j)$.
- Computamos não apenas a distância, mas também o caminho. Para tanto, utilizamos uma matriz $\Pi = [\pi_{ij}]$ com os predecessores.
- A entrada π_{ij} = NIL se i = j e π_{ij} é o predecessor de j em um caminho mais curto de i para j .

Algoritmo de Floyd-Warshall

- Da mesma forma que nos problemas anteriores, o subdígrafo predecessor G_π é uma árvore de caminhos mínimos a partir de um vértice.
- O subdígrafo predecessor $G_{\pi,i}$ induzido pela i-ésima linha é uma árvore de caminhos mínimos em G a partir de i .
- O dígrafo predecessor $G_{\pi,i}$ é definido por:

$$V_{\pi,i} = \{ j \in V : \pi_{i,j} \neq NIL \} \cup \{ i \}$$
$$E_{\pi,j} = \{ (\pi_{i,j}, j) : j \in V_{\pi,j} - \{ i \} \}$$

- Como os vértices são V = {1, 2, ..., n} considere um subconjunto {1, 2, ..., k};
- Para qualquer par de vértices (i,j) em V, considere todos os caminhos de i a j cujos vértices intermediários pertencem ao subconjunto {1, 2, ..., k}, e p como o mais curto de todos eles;
- O algoritmo explora um relacionamento entre o caminho p e os caminhos mais curtos de i a j com todos os vértices intermediários em {1, 2, ..., (k-1)};
- O relacionamento depende de k ser ou não um vértice intermediário do caminho p.

Algoritmo de Floyd-Warshall

Algoritmo:

Floyd-Warshall (*G*)

1. **n** ← |V|
2. $D^0 \leftarrow G$ (iteração zero = matriz de adjacência de *G*)
3. **para** k ← 1 até n **faça**
4. **para** i ← 1 até n **faça**
5. **para** j ← 1 até n **faça**
6. $d^k_{ij} \leftarrow$ Mínimo{ d^{k-1}_{ij}, $d^{k-1}_{ik} + d^{k-1}_{kj}$ }
7. **devolve** D^n .

Algoritmo de Floyd-Warshall

- Há uma variedade de métodos para construção do caminho mais curto no algoritmo de Floyd-Warshall.
- Um jeito é computar a matriz D^n de distâncias e depois construir a matriz predecessora Π a partir de D^n.
- Outra maneira é computar a matriz predecessora Π *"on-line"*, da mesma forma que o algoritmo Floyd-Warshall calcula as matrizes D^k.
- Podemos computar $\Pi^0, \Pi^1, \ldots, \Pi^n$, onde $\Pi = \Pi^n$.
- Π^k_{ij} é definida como a matriz predecessora do vértice j no caminho mais curto a partir de i, tendo como vértices intermediários os elementos do conjunto $\{1, 2, \ldots, k\}$.
- Podemos dar uma fórmula recursiva para Π^k_{ij}.
- Quando $k = 0$, um caminho mais curto de i para j não tem vértices intermediários, logo:

$$\pi^0_{ij} = \begin{cases} nil & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

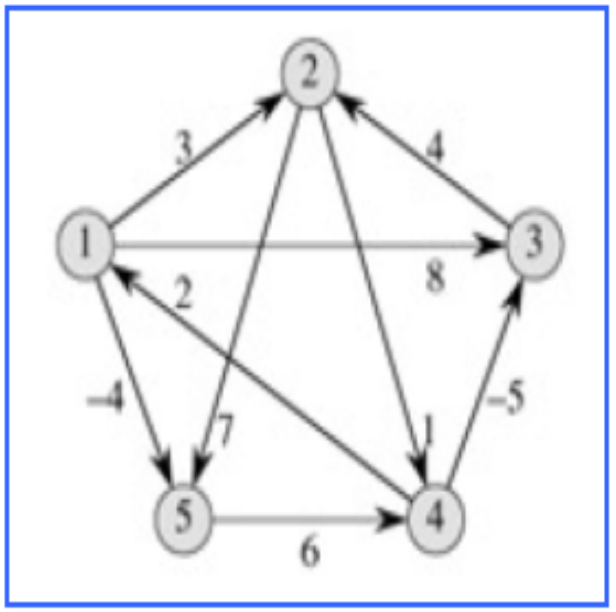
- Para $k \geqslant 1$, se tomamos o caminho $i \rightsquigarrow k \rightsquigarrow j$, com $k \neq j$, então o predecessor de j é o mesmo que o predecessor de j escolhido no caminho mais curto de k para j, com vértices intermediários do conjunto $\{1, \ldots, k-1\}$.
- Ou seja, o predecessor de j no caminho p_{kj}^{k-1}.
- Formalmente, temos: $$\pi_{ij}^{k} = \begin{cases} \pi_{ij}^{k-1} & \text{if } d_{ij}^{k-1} \leqslant d_{ik}^{k-1} + d_{kj}^{k-1} \\ \pi_{kj}^{k-1} & \text{if } d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$$

Obtidas as matrizes D^n e Π^n podemos imprimir um caminho mais curto desde o vértice i até o vértice j, onde i, j $\in$ V.

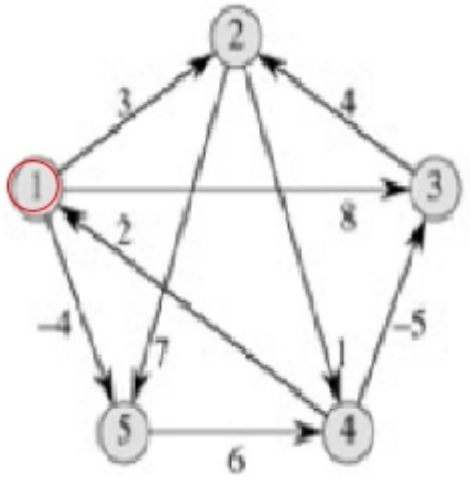
```
MostraCaminhoMaisCurto( Π, i, j )
1. se i = j
2.     então escreve i
3.     senão se π_ij = NIL
4.             então escreve i"não há caminho de" i "para" j
5.             senão MostraCaminhoMaisCurto( Π, i, π_ij )
6.                   escreve j.
```

Exemplo:

Considere o dígrafo abaixo, que contêm pesos positivos e negativos e não tem ciclo negativo



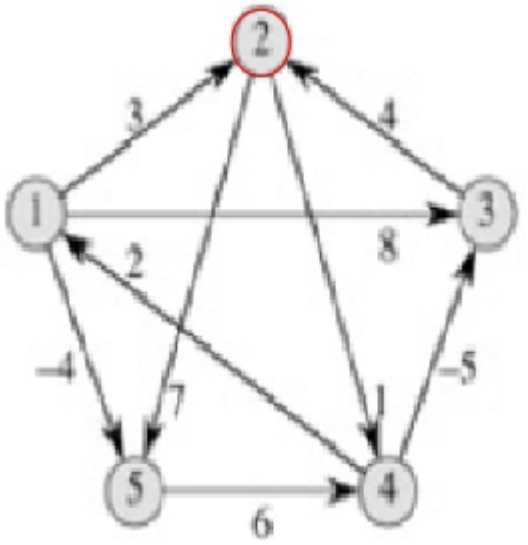
Aplicando o algoritmo de Floyd-Warshall obtemos a sequência de matrizes D^k e Π^k:



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

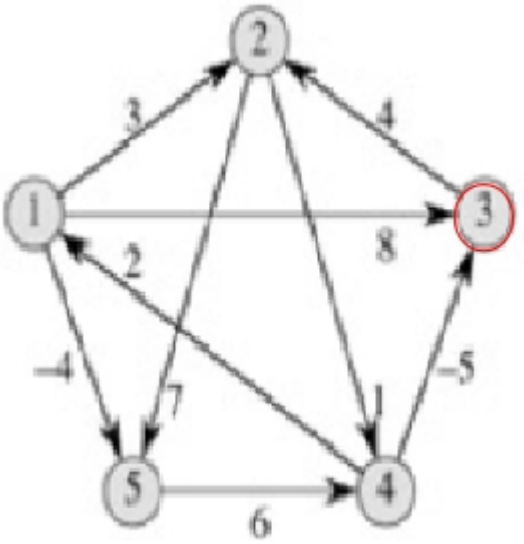
Aplicando o algoritmo de Floyd-Warshall obtemos a sequência de matrizes D^k e Π^k:



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

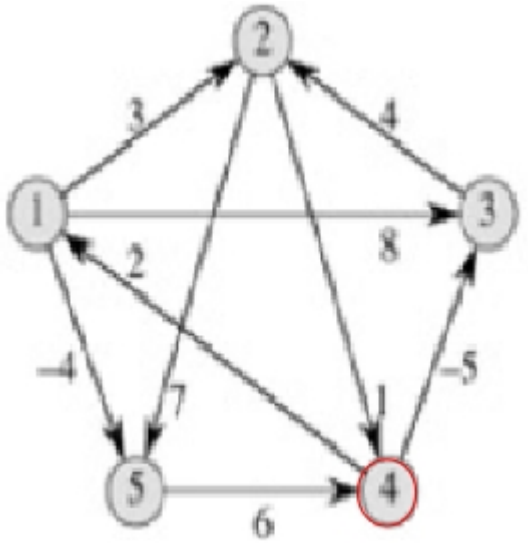
Aplicando o algoritmo de Floyd-Warshall obtemos a sequência de matrizes D^k e Π^k:



$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

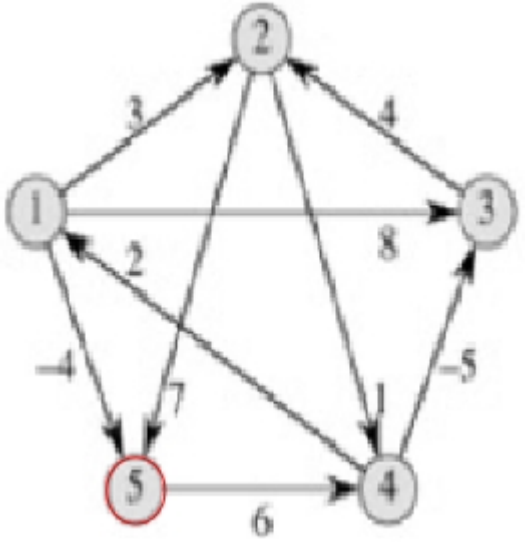
Aplicando o algoritmo de Floyd-Warshall obtemos a sequência de matrizes D^k e Π^k:



$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & \text{NIL} & 4 & 5 & \text{NIL} \end{pmatrix}$$

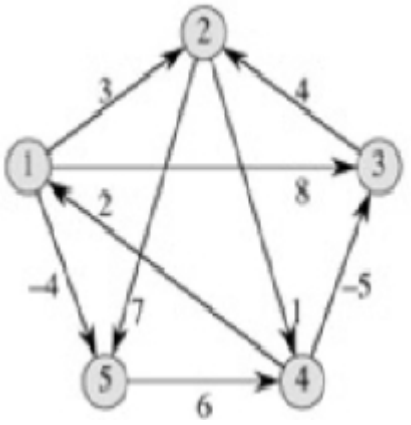
Aplicando o algoritmo de Floyd-Warshall obtemos a sequência de matrizes D^k e Π^k:



$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & \text{NIL} & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Aplicando o algoritmo de Floyd-Warshall obtemos a sequência de matrizes D^k e Π^k:



$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

E se queremos verificar o caminho mais curto entre os vértices:

- "3" e "4" obtemos como MostraCaminhoMaisCurto(Π, 3, 4): 3 → 2 → 4 .
- Ou, entre "5" e "2" aplicamos MostraCaminhoMaisCurto(Π, 5, 2): 5 → 4 → 3 → 2 .

+ exemplo...

Encontre os caminhos mínimos de todos os pares do dígrafo:

	1	2	3	4
1	-	-1	∞	3
2	∞	-	3	2
3	∞	3	-	1
4	2	-1	∞	-