ALGORITMOS E ESTRUTURAS DE DADOS II

Grafos – Caminhos Mínimos com Uma Fonte

Prof. Rafael Fernandes Lopes

http://www.dai.ifma.edu.br/~rafaelf

rafaelf@ifma.edu.br

Introdução

- Um motorista deseja encontrar o caminho mais curto da cidade de São Luís/MA a Fortaleza/CE

- Dado um mapa das rodovias brasileiras, no qual pares de cidades adjacentes são marcados, como podemos determinar a rota mais curta?
  - Uma possibilidade consiste em enumerarmos todos os possíveis caminhos de São Luís a Fortaleza
  - Adicionar as distâncias em cada rota
  - Selecionar a rota mais curta

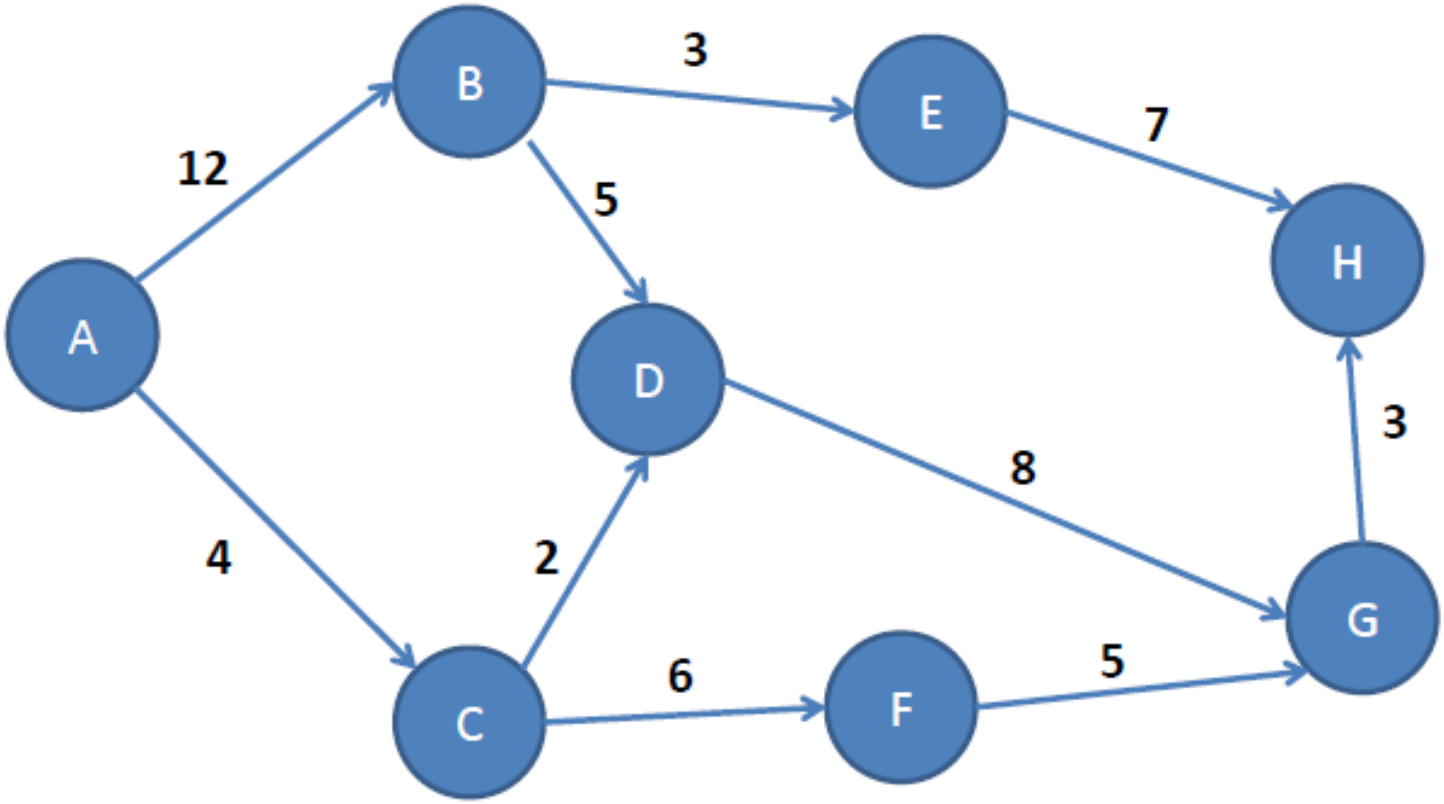
Introdução

- O problema de menor caminho consiste em encontrar menor caminho entre um vértice de origem $s \in V$ e um vértice $v \in V$

- O método da força bruta é satisfatório?

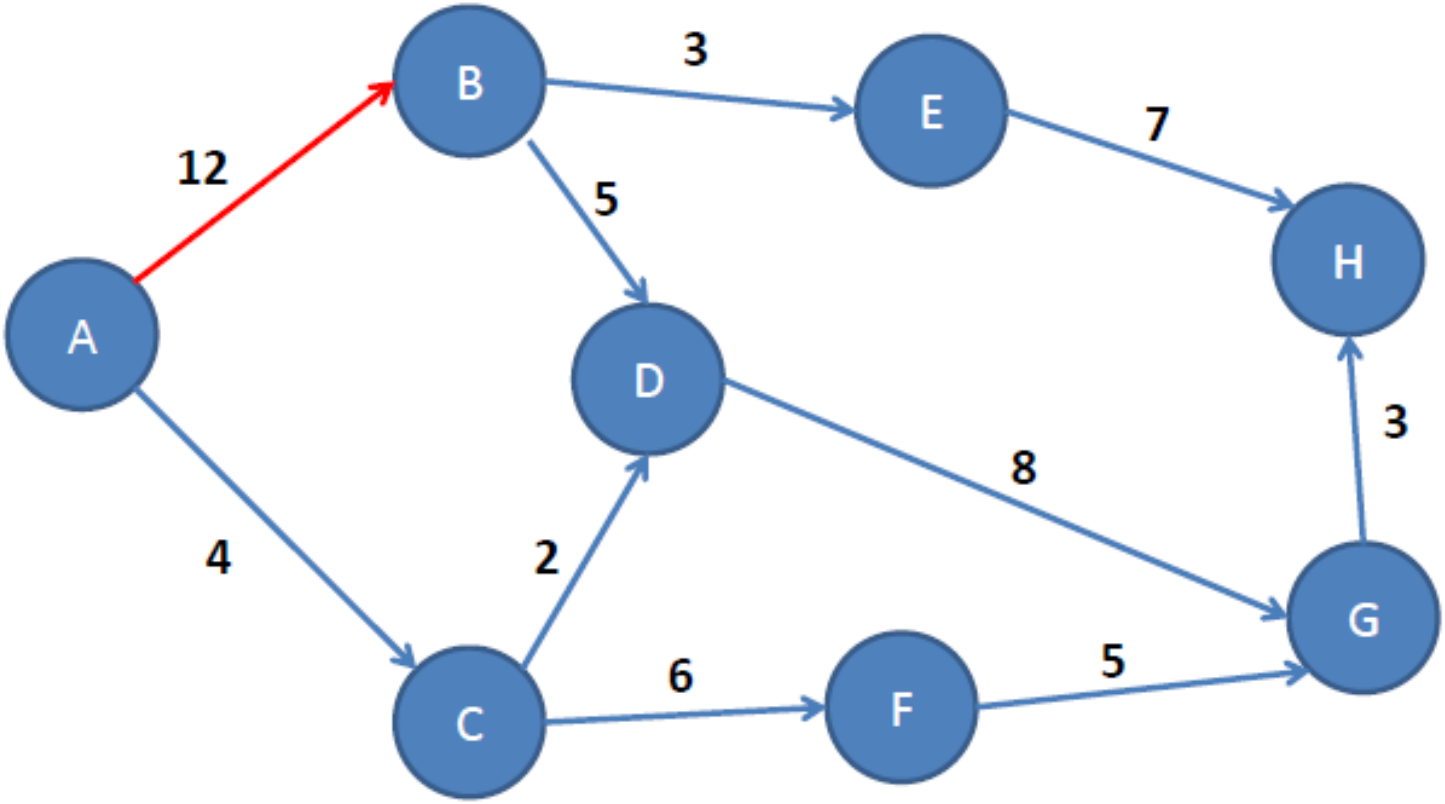
Força Bruta

- $A \rightarrow H$



Força Bruta

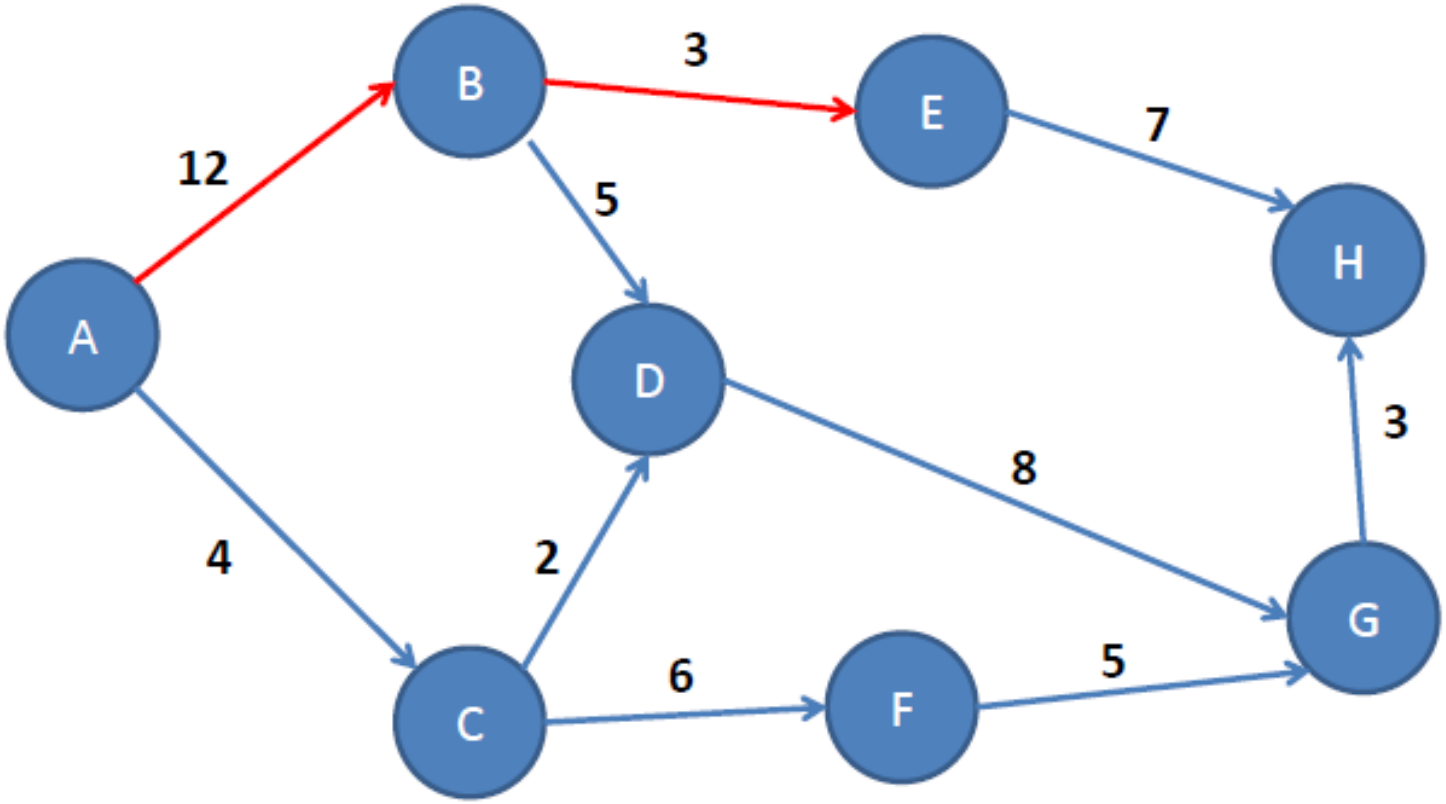
- $A \rightarrow H$



Força Bruta

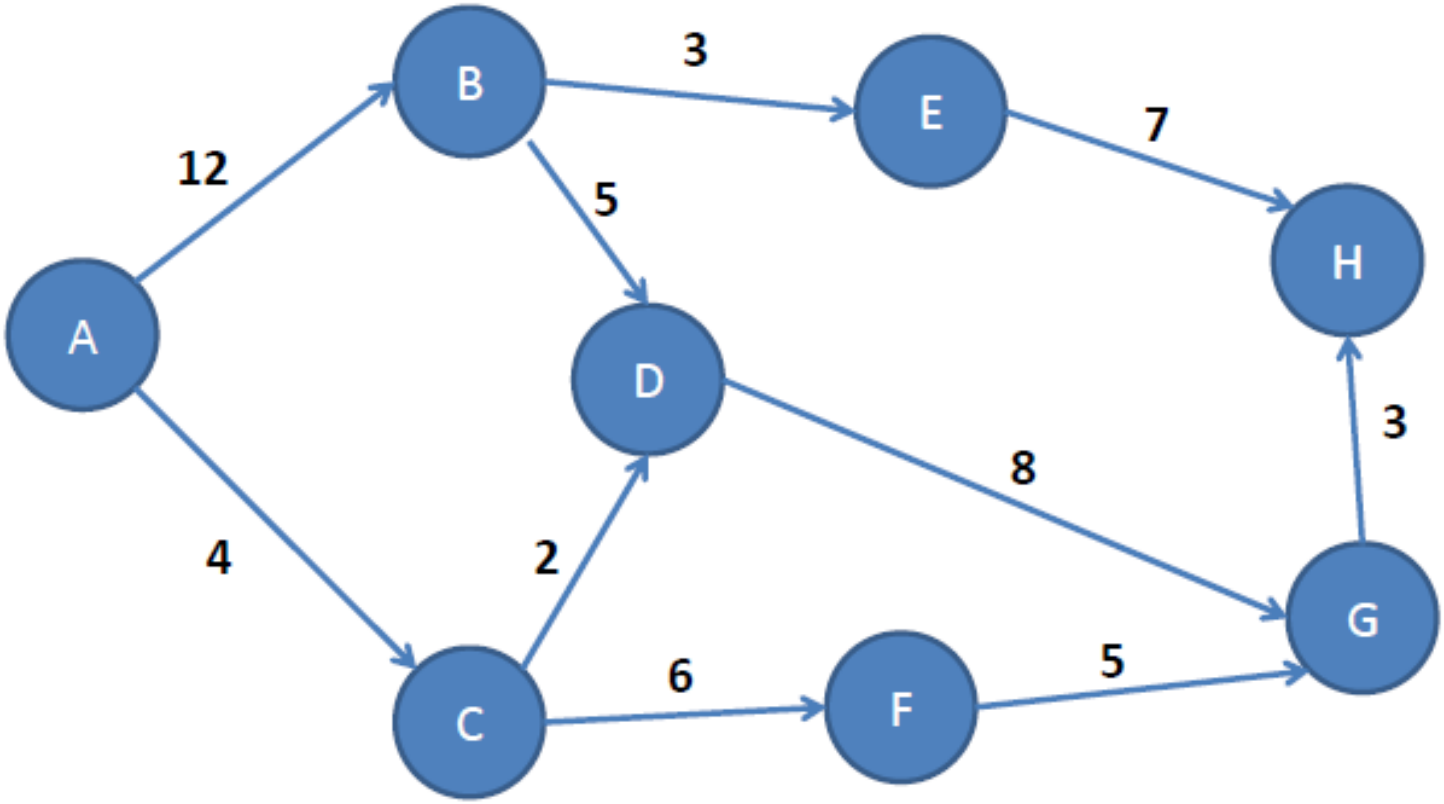
- A → H

A B E H = 22



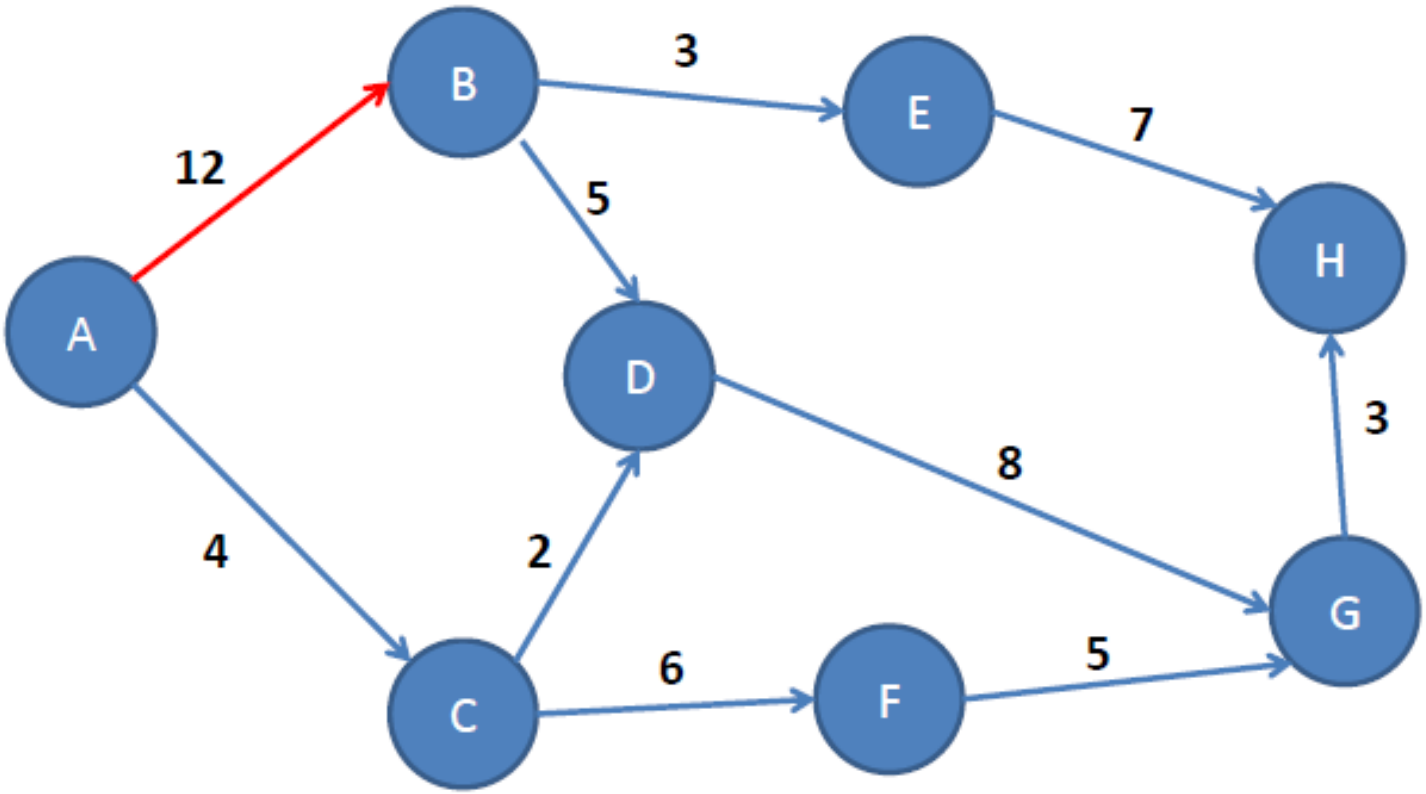
Força Bruta

- $A \rightarrow H$



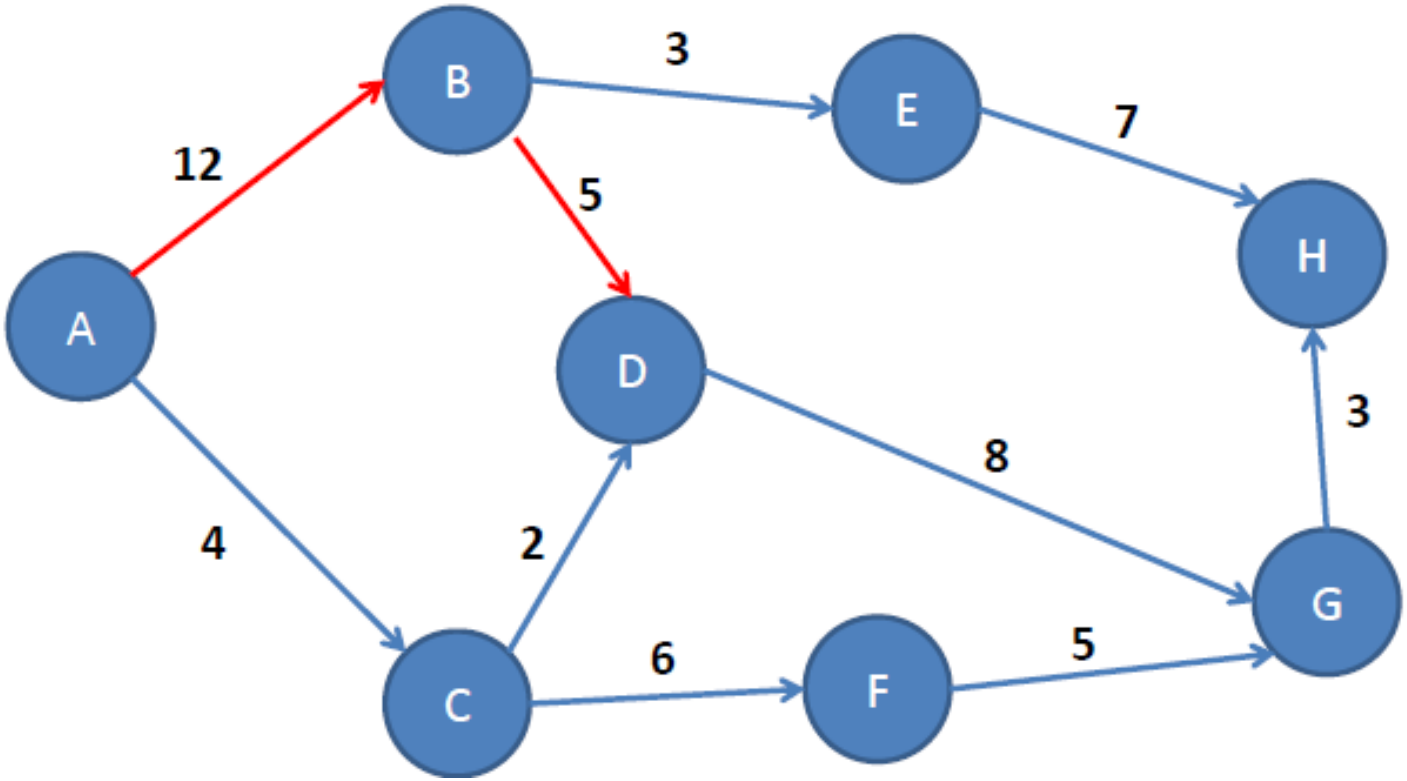
Força Bruta

- A → H



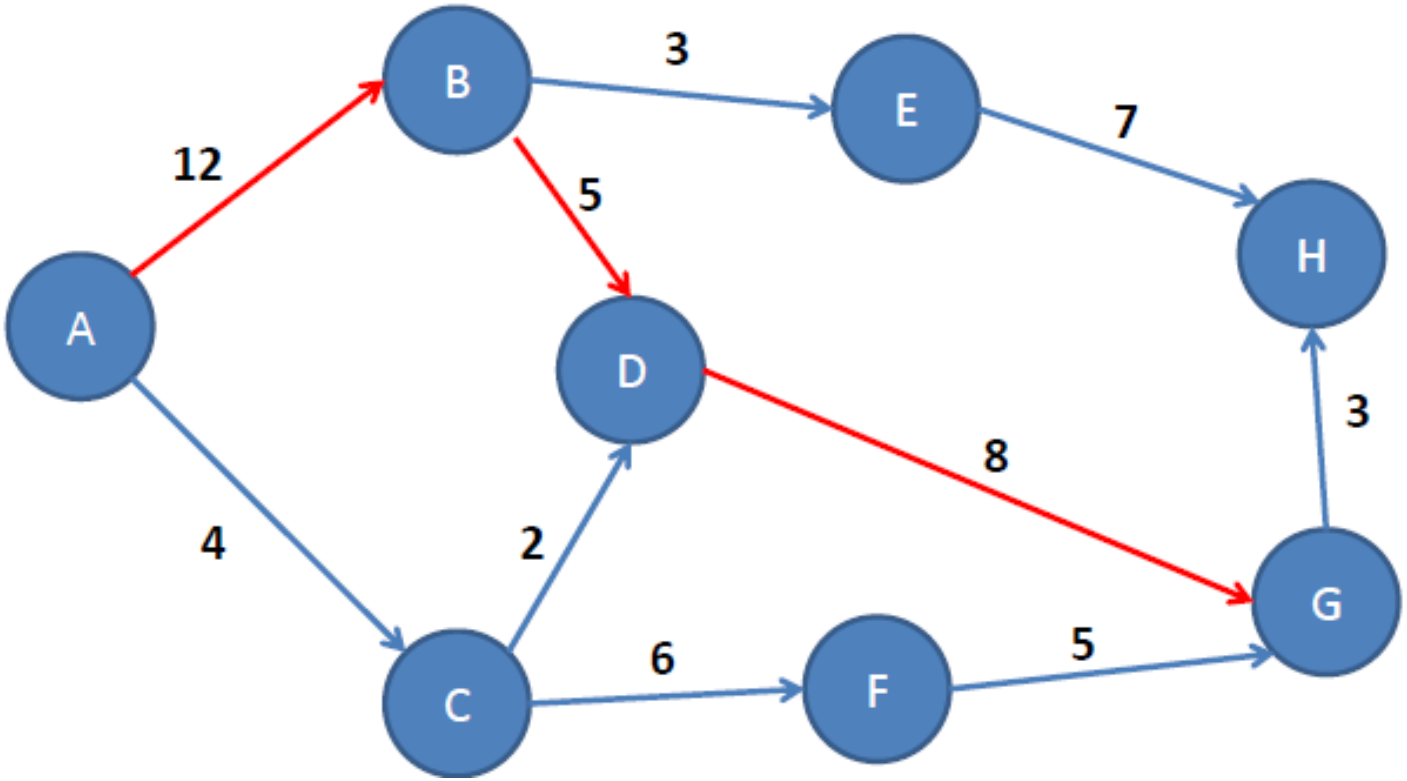
Força Bruta

- $A \rightarrow H$



Força Bruta

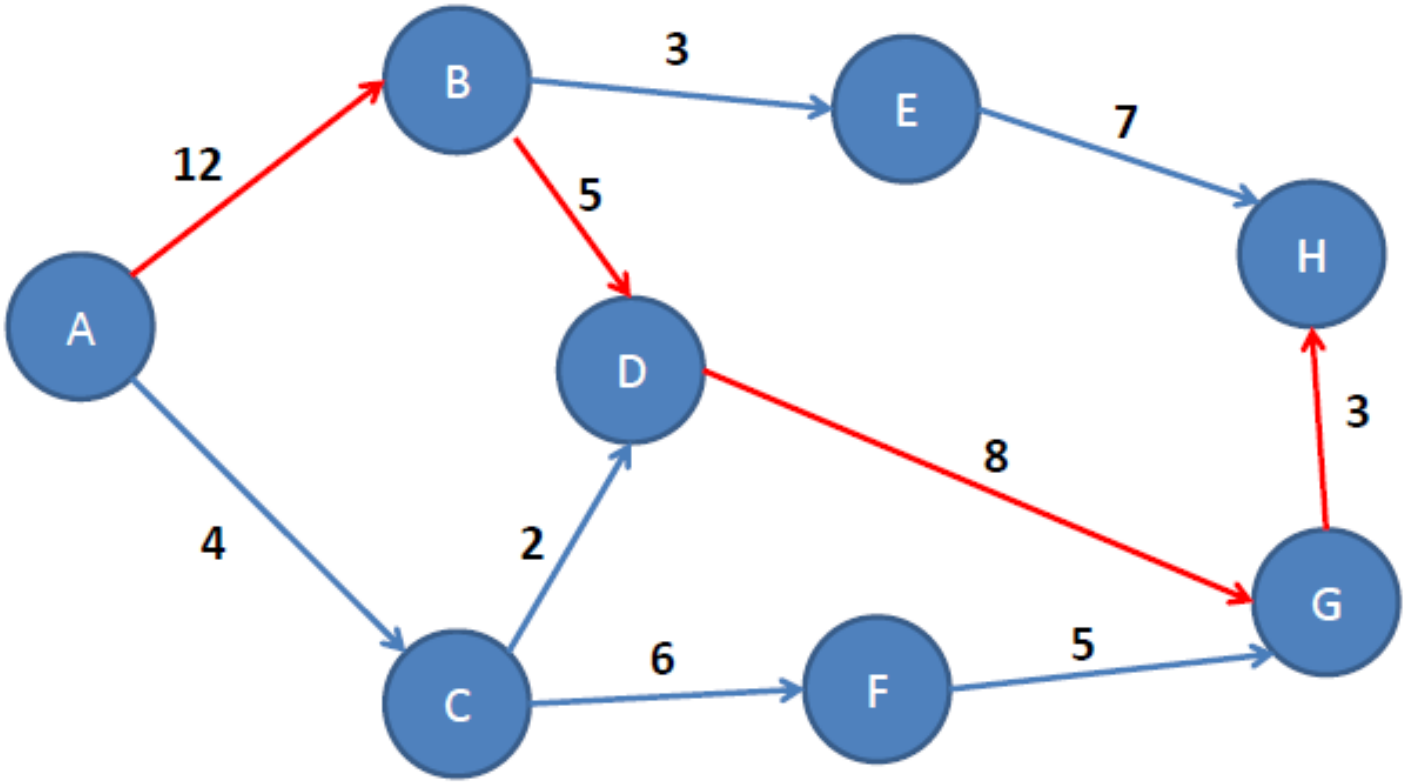
- A → H



Força Bruta

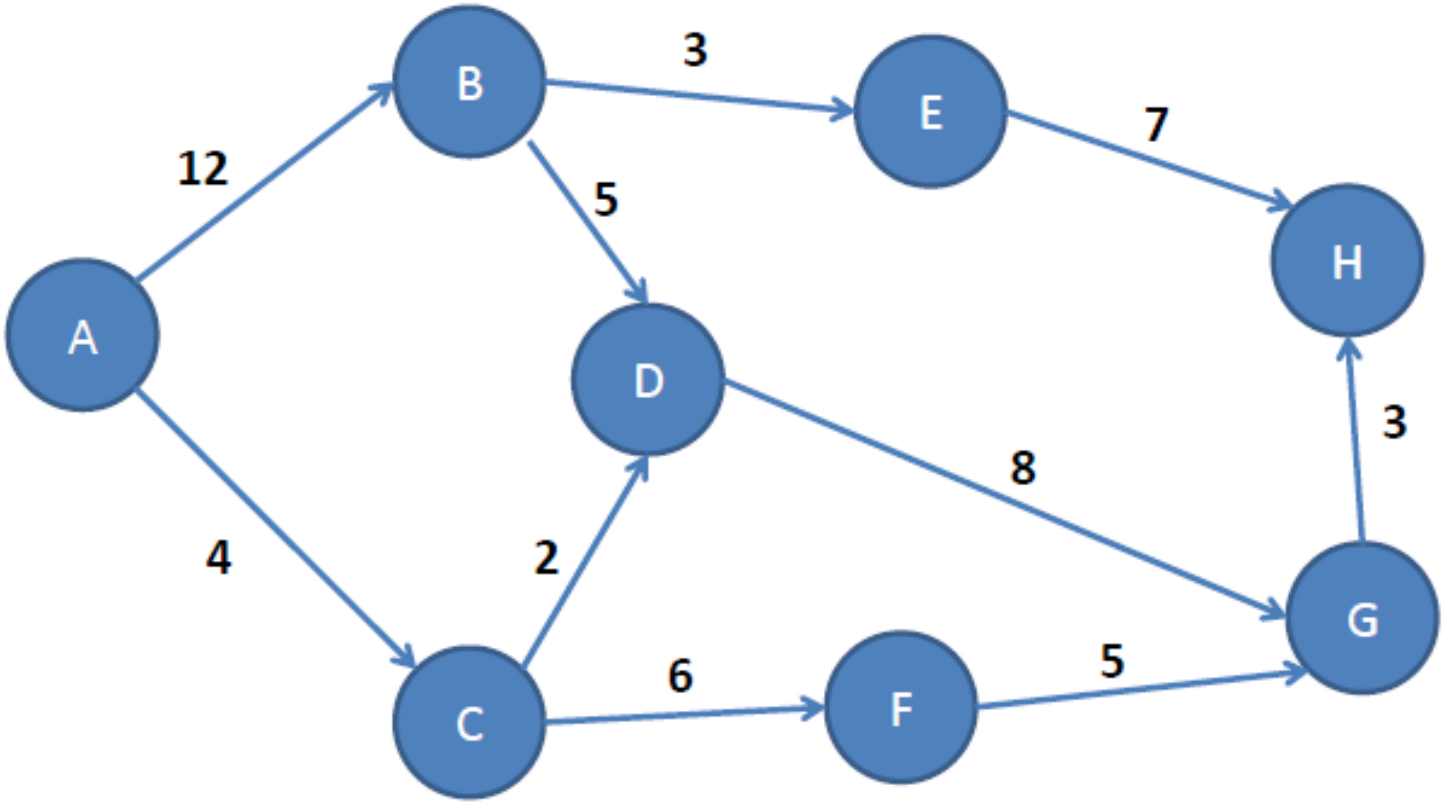
- A → H

A B E H = 22
A B D G H = 28



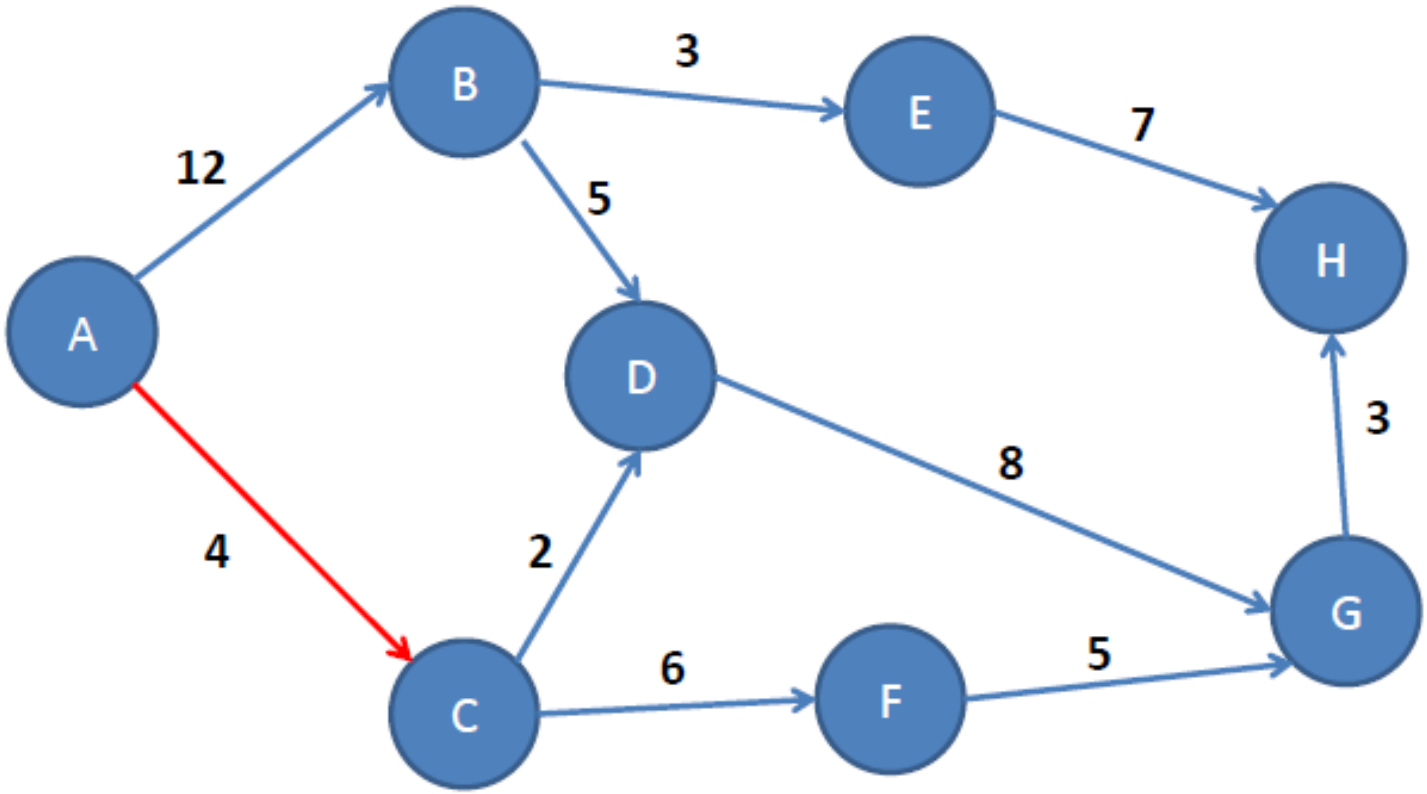
Força Bruta

- $A \rightarrow H$



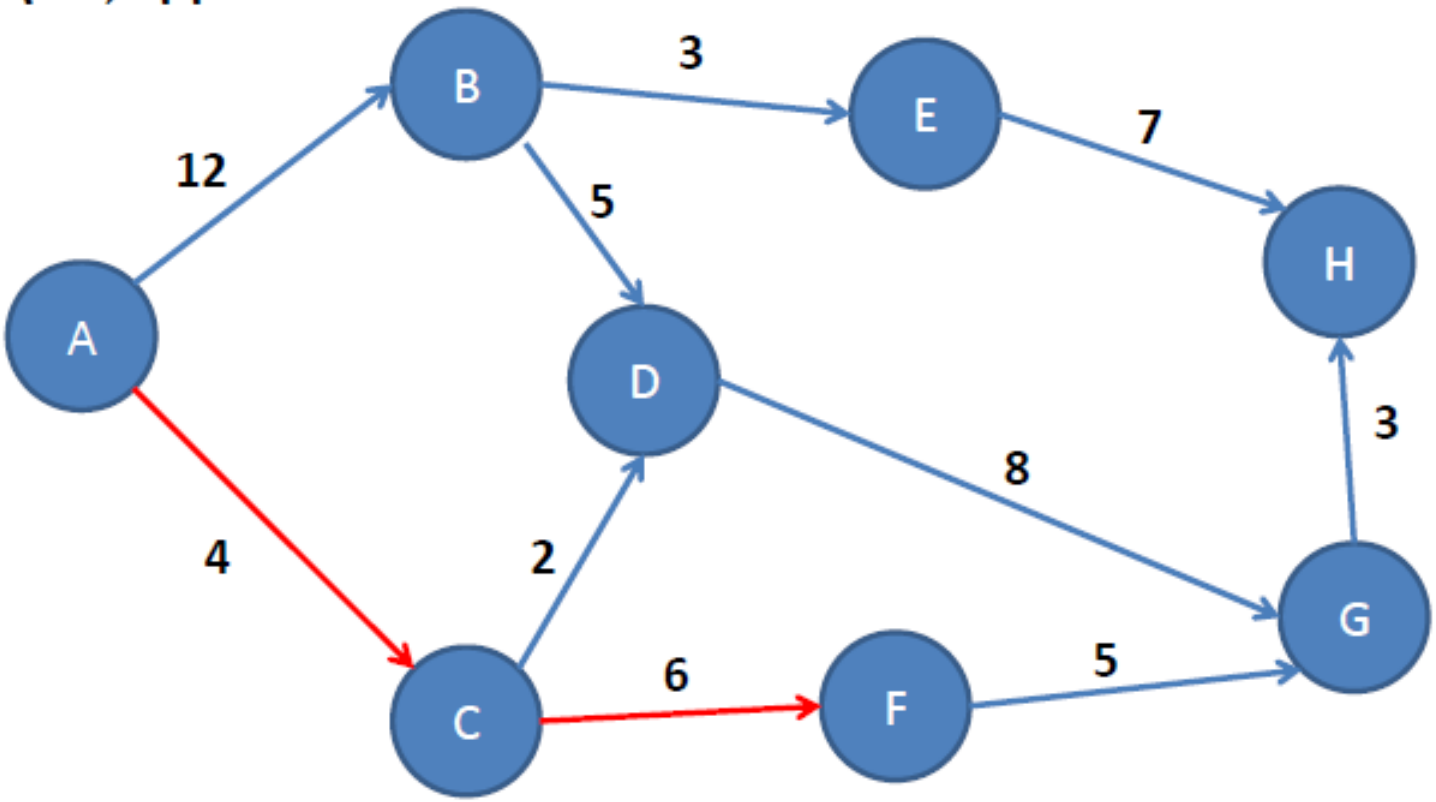
Força Bruta

- $A \rightarrow H$



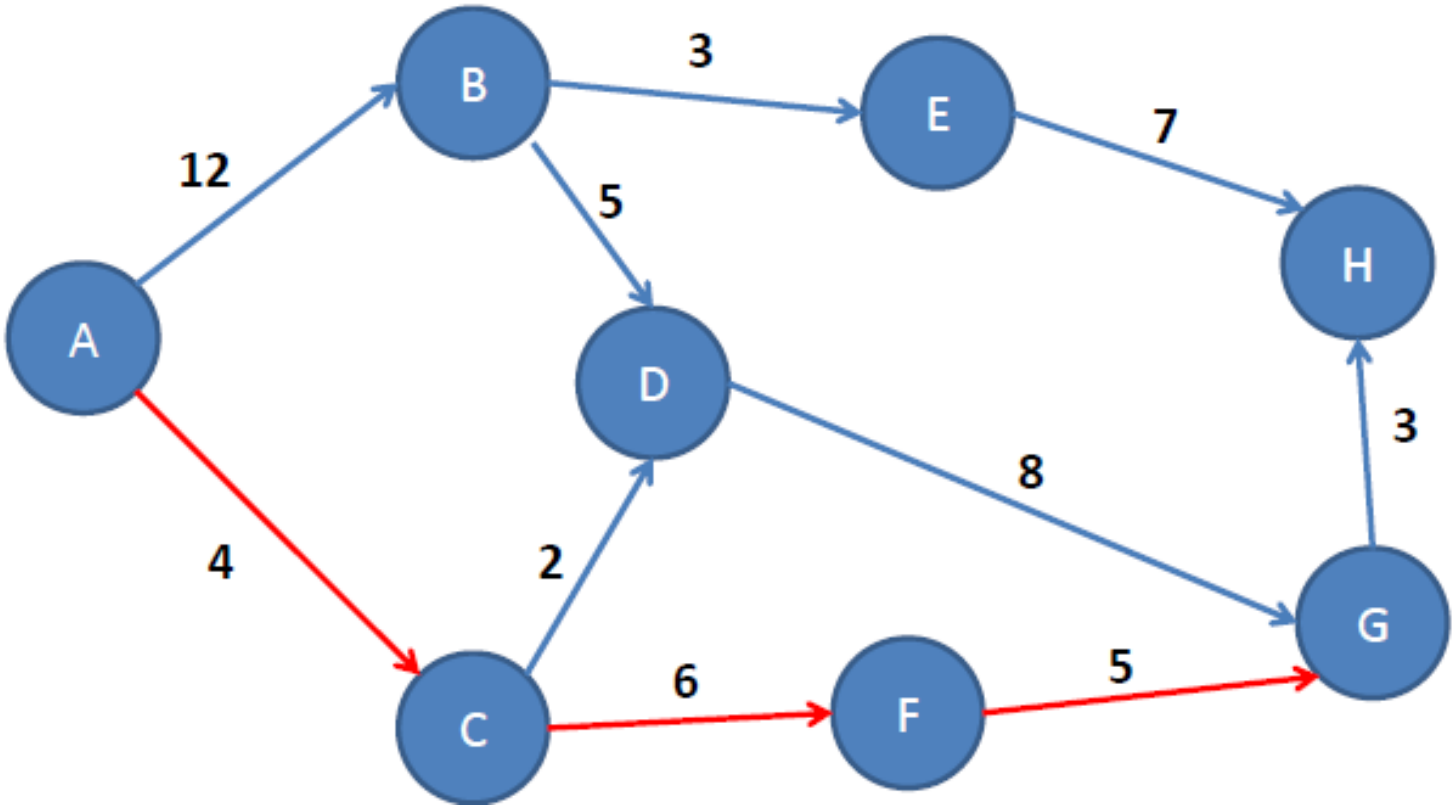
Força Bruta

- A → H



Força Bruta

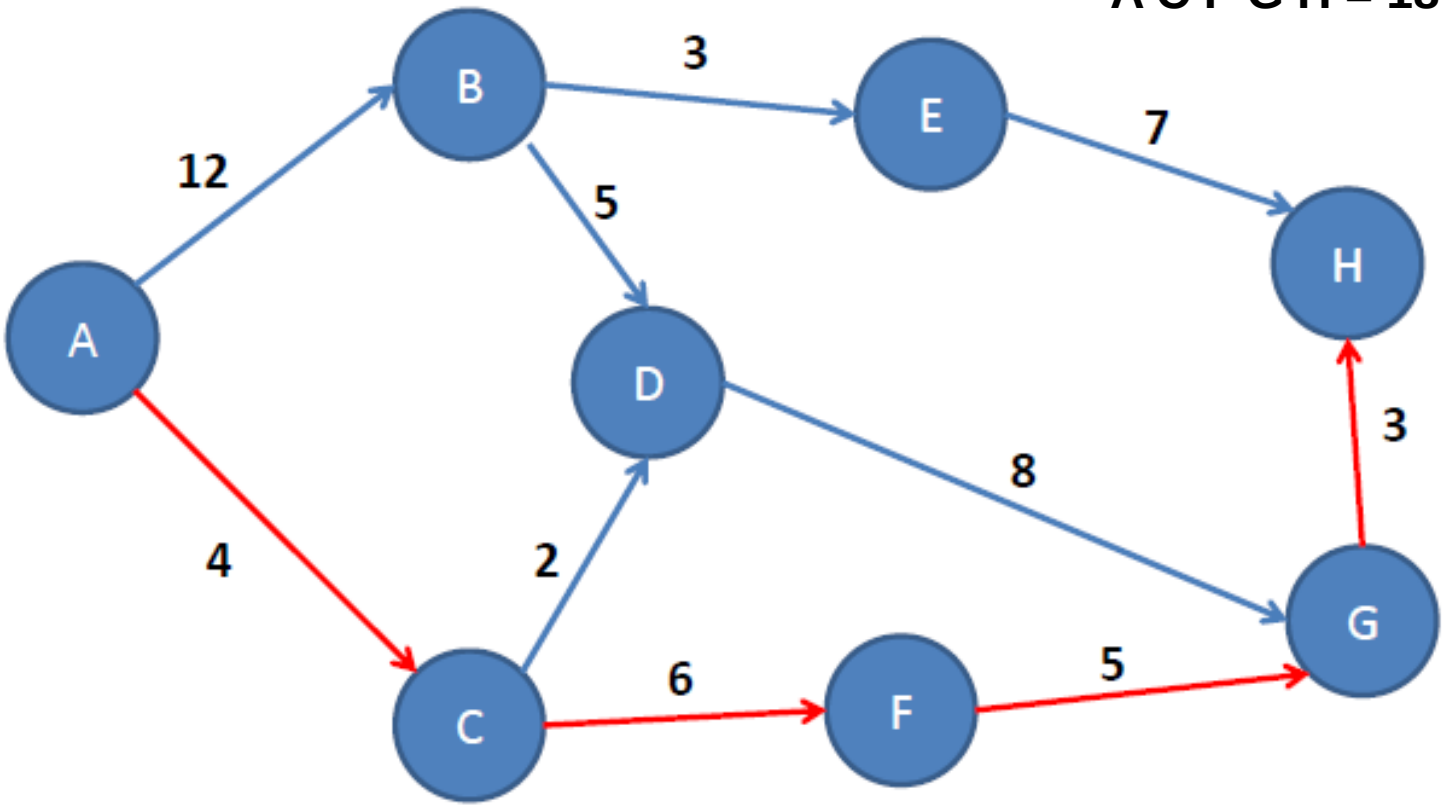
- A → H



Força Bruta

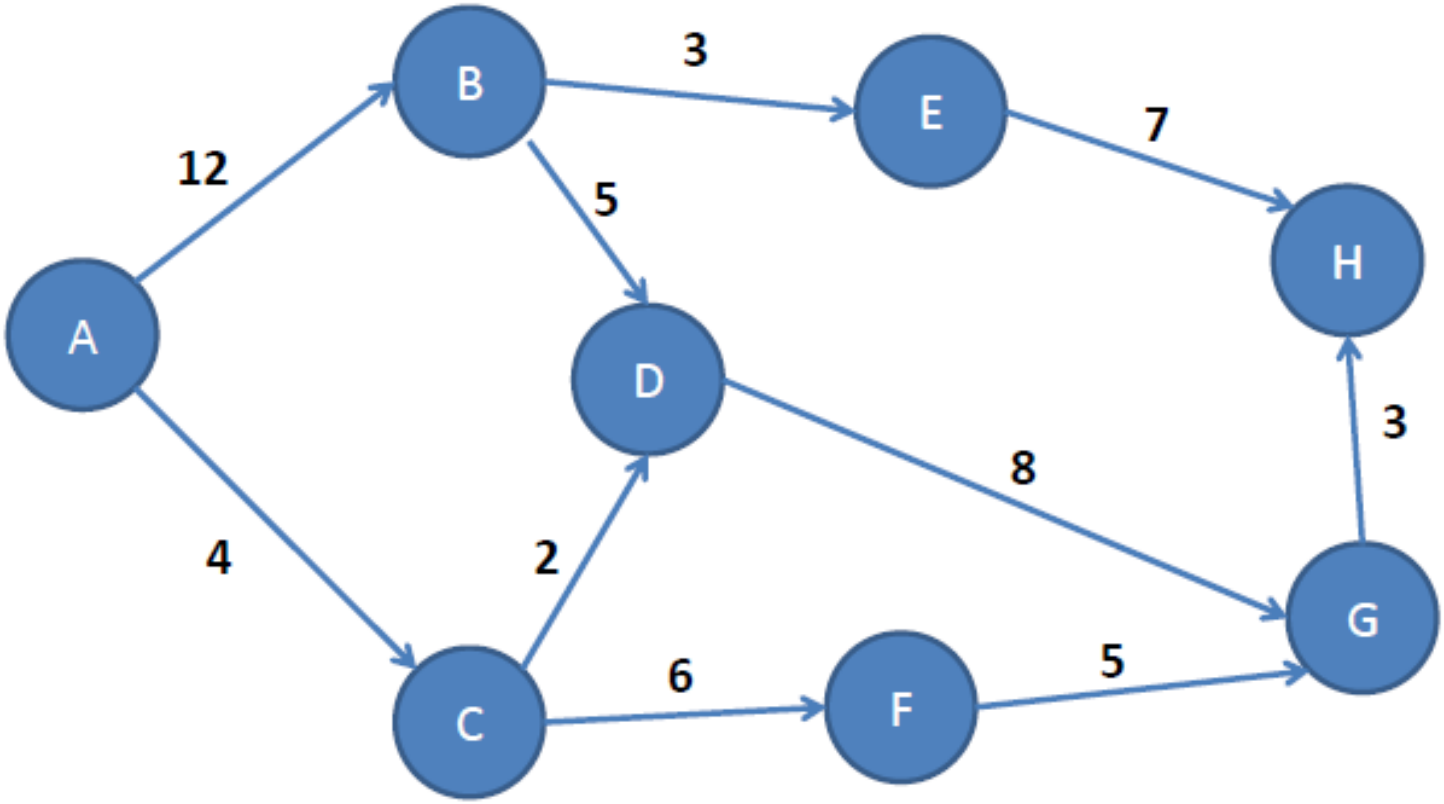
- A → H

A B E H = 22
A B D G H = 28
A C F G H = 18



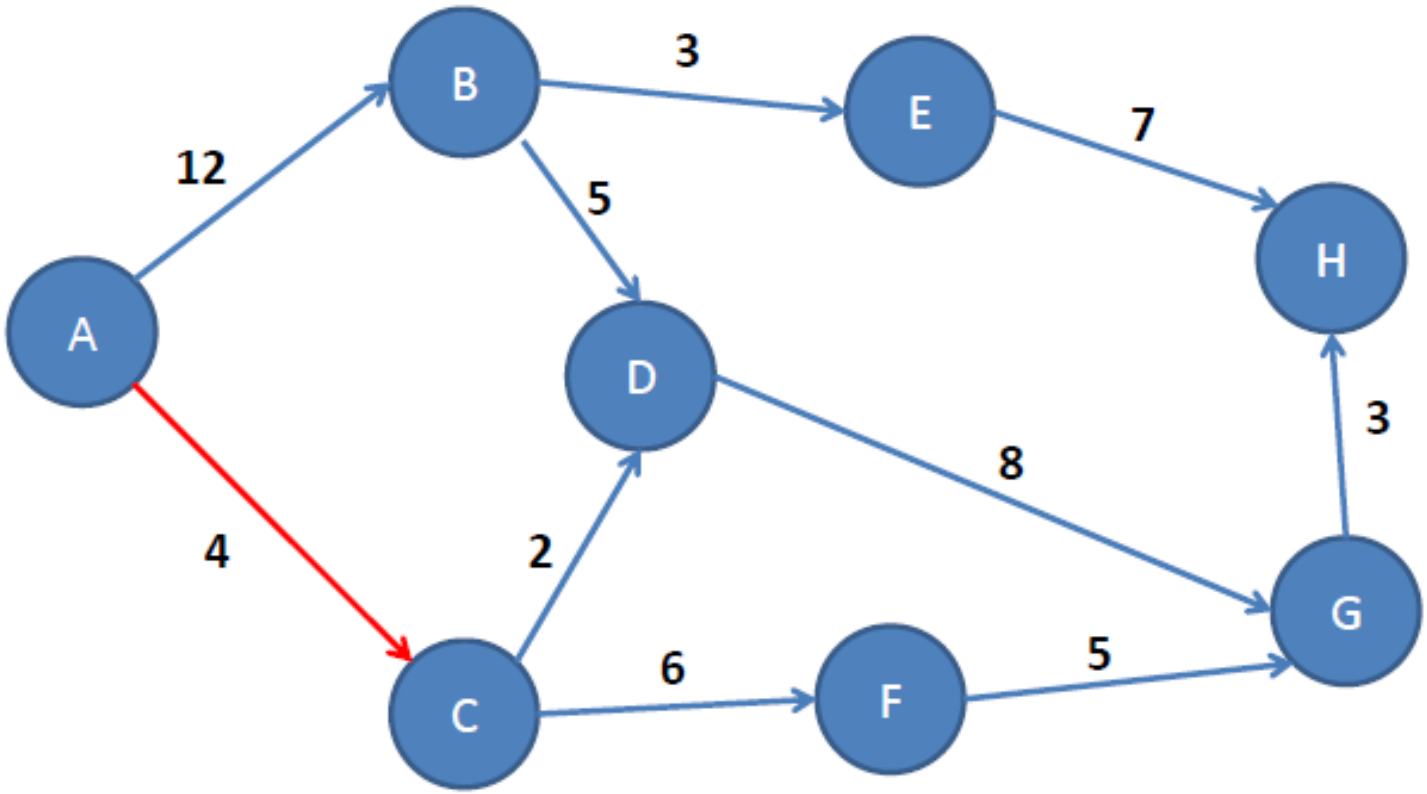
Força Bruta

- $A \rightarrow H$



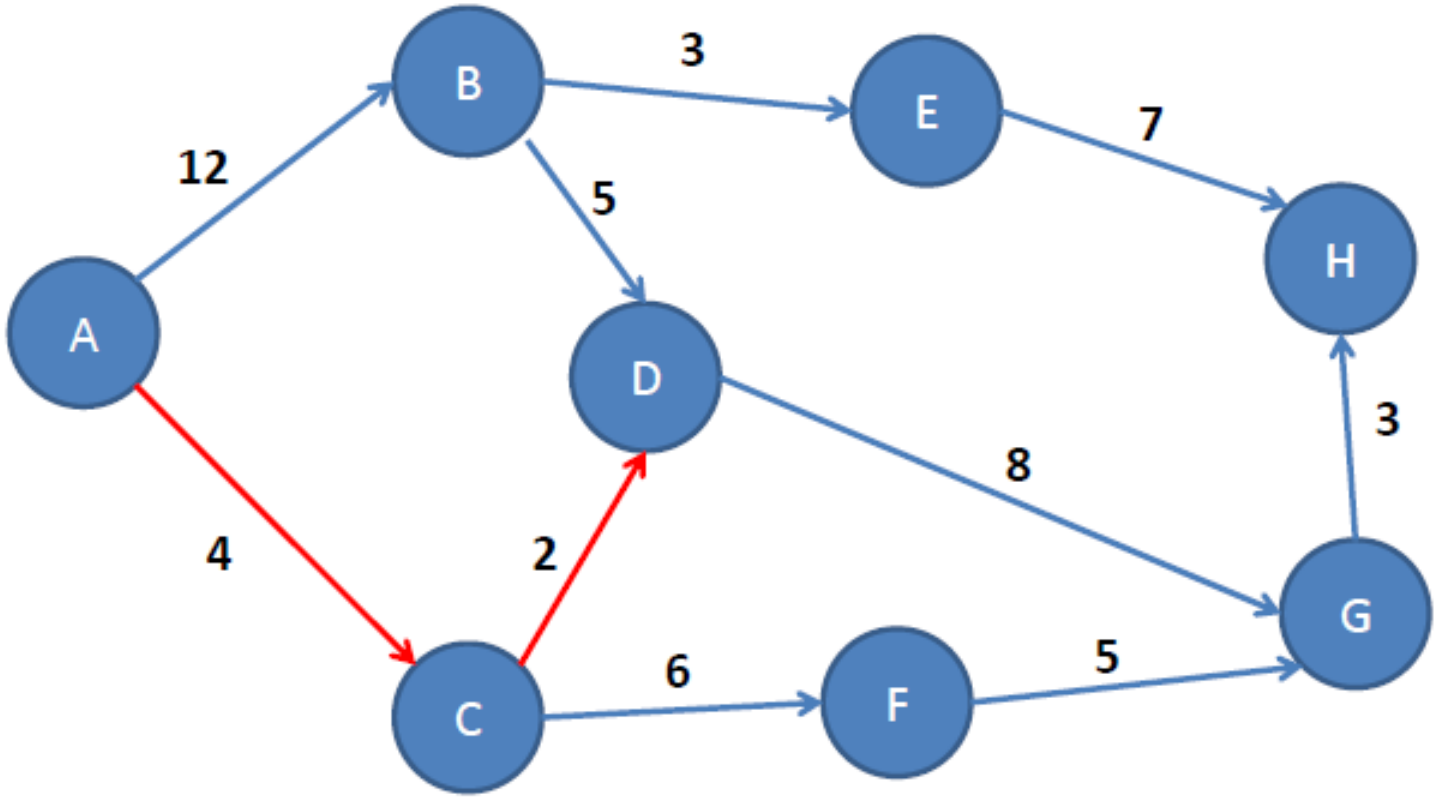
Força Bruta

- $A \rightarrow H$



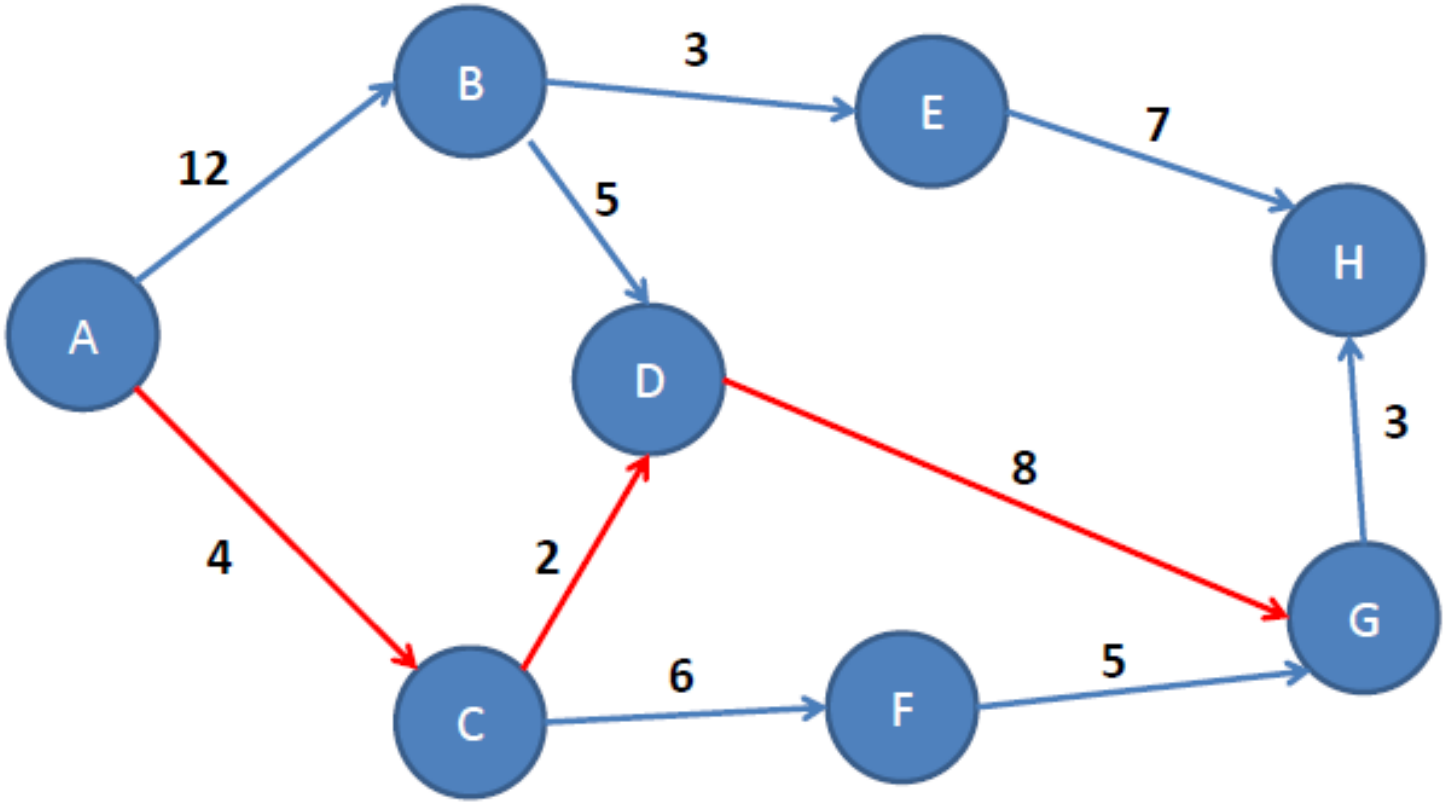
Força Bruta

- A → H



Força Bruta

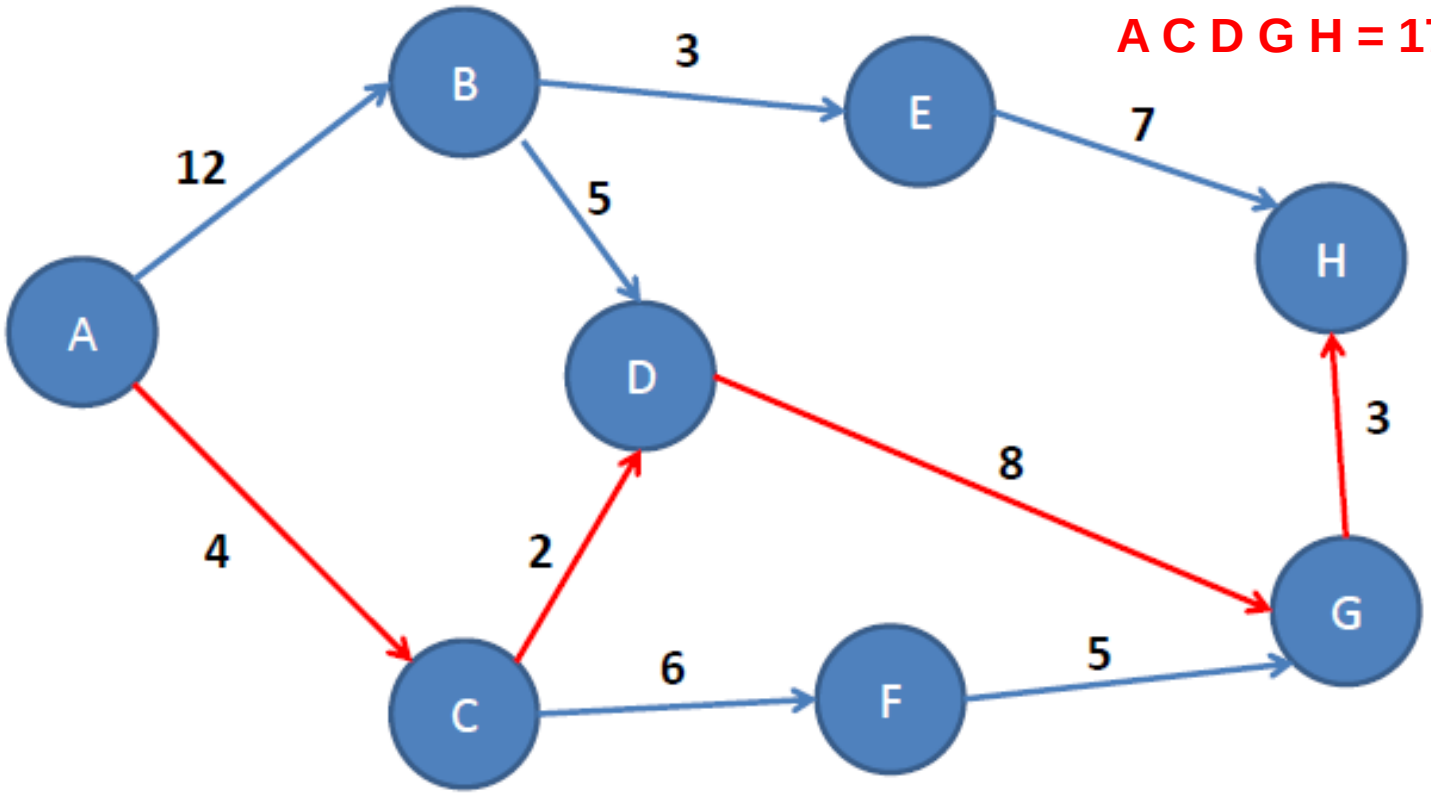
- $A \rightarrow H$



Força Bruta

- A → H

A B E H = 22
A B D G H = 28
A C F G H = 18
A C D G H = 17



Caminhos Mínimos

- Veremos como resolver problemas deste tipo de forma eficaz
- Em problemas de caminhos mínimos, é utilizado um grafo G = (V, E) cujas arestas têm pesos
- Os pesos são induzidos por uma função $w : E \rightarrow \Re$
- O peso de um caminho $p = (v_0, v_1, \ldots, v_k)$ é a soma dos pesos das arestas:

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

Caminhos Mínimos - Definição

- Definimos o peso (comprimento) do caminho de menor peso (mais curto) de um vértice u para v como:

$$\delta(u,v) = \begin{cases} \min\{w(p) : p \text{ é caminho de } u \rightsquigarrow v\} \\ +\infty \text{ se não existir caminho} \end{cases}$$

Caminhos Mínimos

- No exemplo do problema São Luís - Fortaleza, é possível modelar o mapa como um grafo:
  - Os vértices representam as cidades
  - As arestas representam as rodovias, estradas, etc
  - Os pesos das arestas representam as distâncias entre duas cidades

Caminhos Mínimos – Variantes

1. Caminhos mínimos com um destino
   - Encontre o caminho mais curto para um vértice t a partir de cada vértice u $\in$ V

2. Caminho mínimo entre um par de vértices
   - Encontre o caminho mais curto de u para v

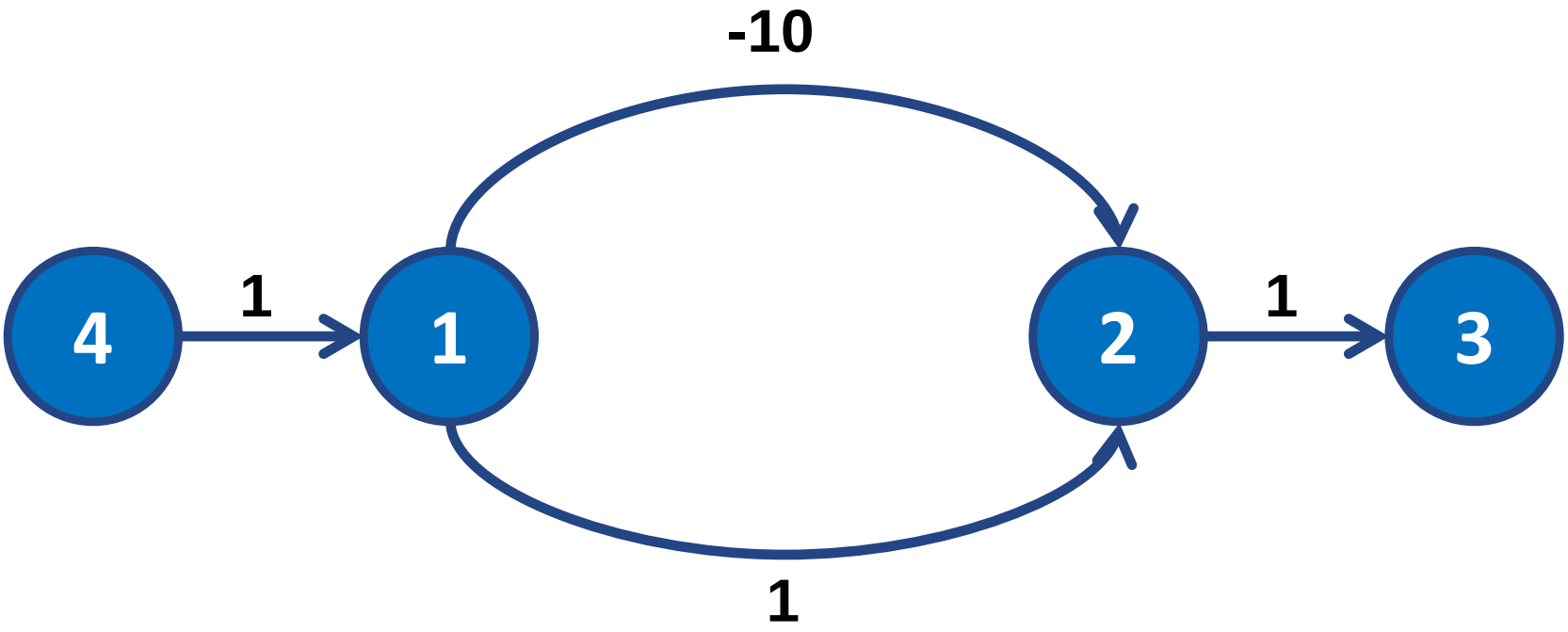
3. Todos os caminhos mínimos
   - Encontre o caminho mínimo entre cada par u, v $\in$ V

Caminhos Mínimos – Pesos negativos

- Em algumas instâncias do problema de caminhos mínimos com fonte única, podem existir arestas com pesos negativos

- Se o grafo G = (V, E) não tiver ciclo com peso negativo alcançável a partir da fonte s, então (s, v) permanece bem definido para todo $v \in V$, mesmo se o grafo contiver algum ciclo negativo

Caminhos Mínimos – Pesos negativos



Caminhos Mínimos – Pesos negativos

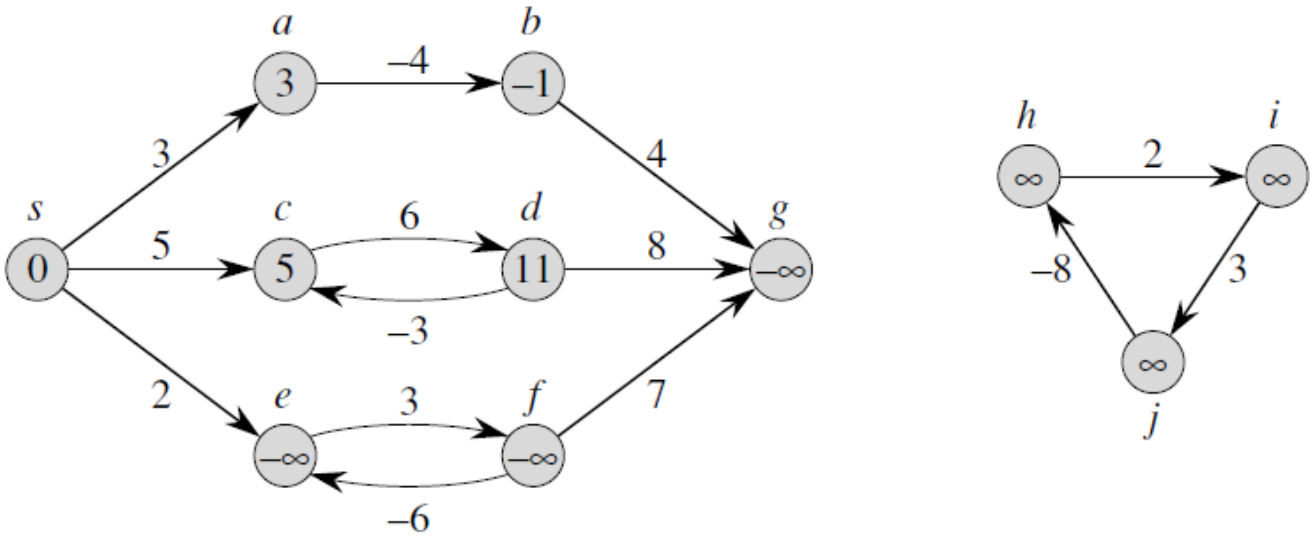


Figure 24.1 Negative edge weights in a directed graph. Shown within each vertex is its shortest-path weight from source s. Because vertices e and f form a negative-weight cycle reachable from s, they have shortest-path weights of $-\infty$. Because vertex g is reachable from a vertex whose shortest-path weight is $-\infty$, it, too, has a shortest-path weight of $-\infty$. Vertices such as h, i, and j are not reachable from s, and so their shortest-path weights are ∞, even though they lie on a negative-weight cycle.

Caminhos Mínimos – Pesos negativos

- Alguns algoritmos, como o algoritmo de Dijkstra, assumem que todos os pesos são não-negativos

- Algoritmos como Bellman-Ford e Floyd-Warshall, entretanto, podem operar com arestas de peso negativo, desde que não existam ciclos de peso negativo

- Tipicamente, estes algoritmos detectam a presença de ciclos negativos

Caminhos Mínimos – Representação

- Estamos interessados não apenas na distância do caminho mais curto, mas também no caminho em si
- A representação do caminho mais curto é similar àquela usada na busca em largura (BFS)
- Dado um grafo G = (V, E), mantemos para cada vértice $v \in V$ o seu predecessor $\pi[v]$ que é outro vértice ou *nil*
- Os algoritmos de caminhos mínimos definem os atributos π de maneira que a cadeia de predecessores originada em v, de trás para frente, nos dá o caminho mais curto de s para v

Caminhos Mínimos – Relaxações

- Algoritmos de caminhos mínimos usam a técnica de relaxação
  - Decresce um limite superior (*upper bound*) para a distância mínima de cada vértice até que o limite superior se torne a própria distância
- Os algoritmos também exploram a propriedade de que o caminho mais curto entre dois vértices também é formado por caminhos mais curtos

Caminhos Mínimos – Relaxações

- Lema:
  - Dado um grafo direcionado com peso nas arestas, G = (V, E), e a função peso nas arestas $w : E \rightarrow \Re$
  - Seja $p = (v_1, v_2, \ldots, v_k)$ um caminho mais curto de v_1 para v_k em G
  - Para quaisquer $1 \leq i < j \leq k$, seja $p_{ij} = (v_i, v_{i+1}, \ldots, v_j)$o subcaminho de p de v_i para v_j
    - Então, pij é um caminho mínimo de v_i para v_j

Caminhos Mínimos – Relaxações

- Lema:
  - Dado um grafo direcionado com peso nas arestas, G = (V, E), e a função peso nas arestas $w : E \rightarrow \Re$
  - Seja s um vértice correspondente à fonte
  - Então, para toda aresta $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Caminhos Mínimos – Relaxações

- Vários algoritmos são baseados na técnica de relaxação
- Para cada vértice $v \in V$, mantemos um atributo d[v], que é um limite superior (*upper bound*) no comprimento do caminho mais curto de s para v:

$$d[v] \geq \delta(s, v)$$

- d[v] é a estimativa para $\delta(s, v)$

Caminhos Mínimos – Relaxações

INITIALIZE-SINGLE-SOURCE(G, s)

```
1  for each vertex v ∈ V[G]
2       do d[v] ← ∞
3          π[v] ← NIL
4  d[s] ← 0
```

After initialization, $\pi[v] = \text{NIL}$ for all $v \in V$, $d[s] = 0$, and $d[v] = \infty$ for $v \in V - \{s\}$.

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
```

O propósito de relaxar uma aresta (u, v) consiste de testar se podemos melhorar a estimativa do caminho mais curto para v encontrado até então, por meio do caminho até u

Caminhos Mínimos – Relaxações

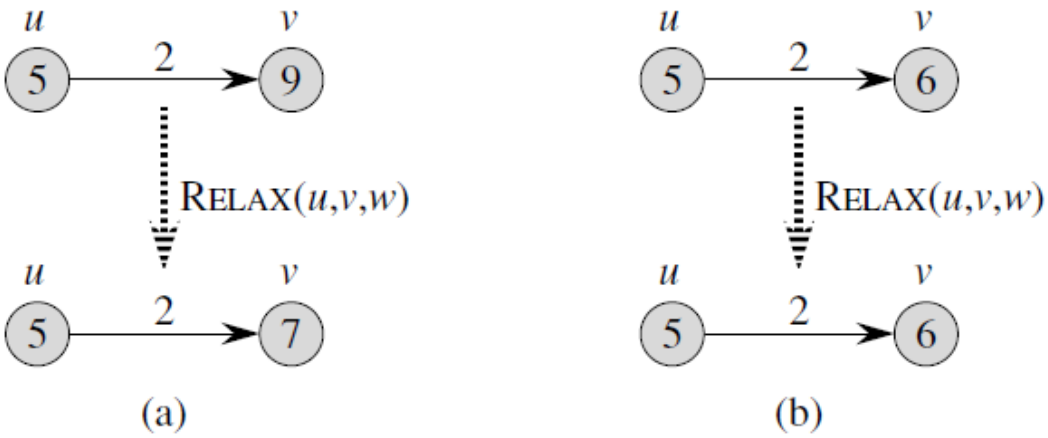


Figure 24.3 Relaxation of an edge (u, v) with weight $w(u, v) = 2$. The shortest-path estimate of each vertex is shown within the vertex. **(a)** Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases. **(b)** Here, $d[v] \leq d[u] + w(u, v)$ before the relaxation step, and so $d[v]$ is unchanged by relaxation.

Caminhos Mínimos – Propriedades

- A corretude de algoritmos de caminhos mínimos é baseada em propriedades de caminhos mínimos e das relaxações

- Essas propriedades foram omitidas aqui, mas podem ser encontradas no capítulo 24 do livro do Cormen

Algoritmo de Bellman-Ford

- O algoritmo de Bellman-Ford resolve o problema de caminhos mínimos com uma fonte no caso geral, em que as arestas podem possuir peso negativo

- O algoritmo retorna um valor lógico indicando se foi ou não encontrado um ciclo de comprimento negativo alcançável a partir de s

- Se 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)

Algoritmo de Bellman-Ford

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

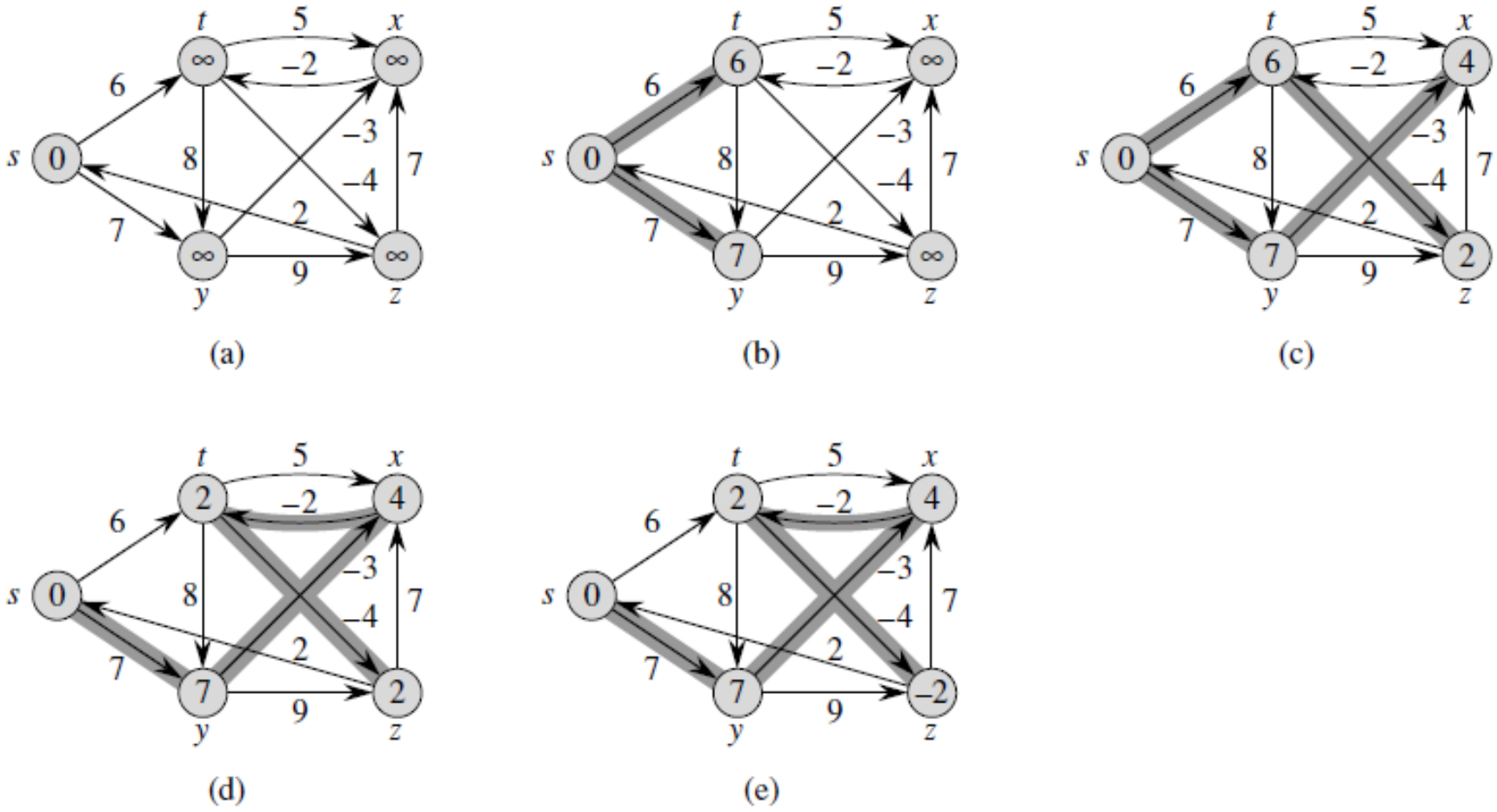


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y). **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

Algoritmo de Bellman-Ford – Complexidade

Definindo $|V| = n$ e $|E| = m$, podemos verificar que:

- Passo 1: $O(n)$
- Passo 2: $\Theta(n)$
- Passo 3-4: $\Theta(1)$
- Logo, o laço 2-4 leva $\Theta(nm)$
- Laço 5-7 leva $O(m)$

Conclusões:

- Portanto, Bellman-Ford executa em tempo $\Theta(nm)$
- Para um grafo denso, $m = \Theta(n^2)$, o tempo de execução do algoritmo é $\Theta(n^3)$

Algoritmo de Dijkstra

- O algoritmo de Dijkstra resolve o “problema de caminhos mínimos com uma fonte” em um grafo direcionado G = (V, E), cujas arestas apresentam pesos não-negativos

- O algoritmo mantém um conjunto S de vértices para os quais a distância do caminho mais curto a partir de s já foi computada

- Ou seja, para todo $v \in S$, temos $d[v] = \delta(s, v)$

- O algoritmo iterativamente seleciona um vértice $u \in V - S$ que possua a menor estimativa de distância, $d[u] = \min\{d[v] : v \in V - S\}$, e insere u em S

- Ao mesmo tempo que “relaxa” as arestas que emanam de u

Algoritmo de Dijkstra

- Princípio:
  - Se desejamos calcular o menor caminho de a para z em um grafo conexo simples com pesos:
    - Primeiro encontramos um menor caminho entre a $\in$ V e um vértice adjacente qualquer x $\in$ V
    - Depois entre a e um segundo vértice
    - O procedimento é repetido até que seja encontrado um menor caminho entre a e z
  - Funciona para grafos direcionados ou não

Algoritmo de Dijkstra

```
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 each vertex v ∈ Adj[u]
8                 do RELAX(u, v, w)
```

Algoritmo de Dijkstra

- Uma vez que o algoritmo de Dijkstra sempre escolhe o vértice de V − S mais próximo do conjunto S, dizemos que o algoritmo faz uso de uma técnica gulosa

- Algoritmos gulosos tipicamente produzem soluções subótimas

- No caso do “problema de caminhos mínimos com uma fonte”, o algoritmo guloso obtém a solução ótima

- A chave para mostrar que o algoritmo produz uma solução ótima é o fato que quando u é inserido em S, $d[u] = \delta(s, u)$

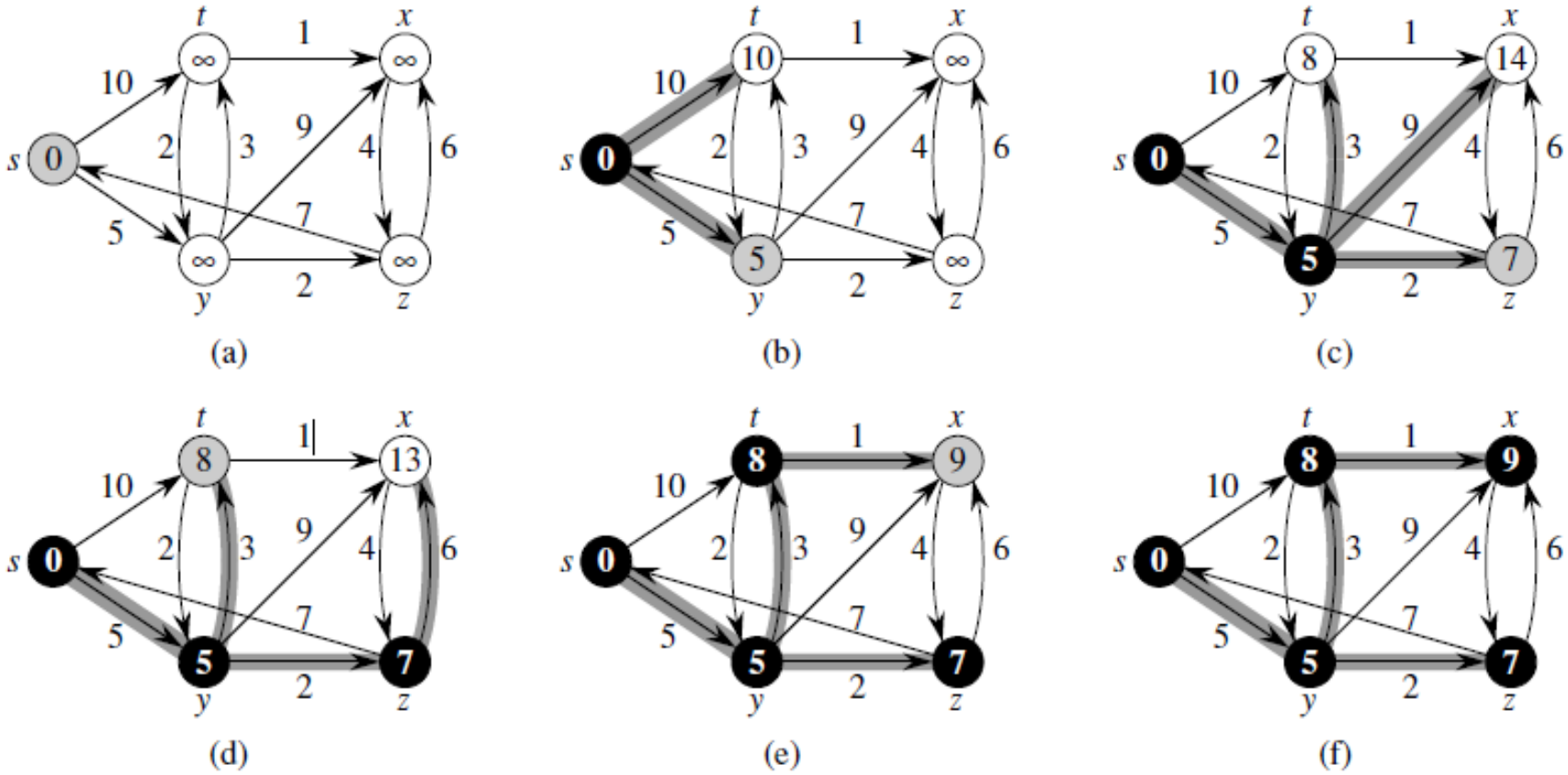


Figure 24.6 The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. (**a**) The situation just before the first iteration of the **while** loop of lines 4–8. The shaded vertex has the minimum d value and is chosen as vertex u in line 5. (**b**)–(**f**) The situation after each successive iteration of the **while** loop. The shaded vertex in each part is chosen as vertex u in line 5 of the next iteration. The d and π values shown in part (f) are the final values.

Algoritmo de Dijkstra – Complexidade

Vetor como fila de prioridades

- Neste caso, extract-min leva tempo O(n) e há n operações extract-min, com n = |V| e m = |E|

- Uma vez que cada aresta é examinada no máximo uma vez, a operação relax é executada no máximo uma vez, levando tempo O(1)

- Podemos concluir que o algoritmo leva tempo $O(n^2 + m) = O(n^2)$

Algoritmo de Dijkstra – Complexidade

Heap binário como fila de prioridades

- Quando o grafo é esparso, entretanto, é mais prático utilizarmos uma fila de prioridades implementada com heap binário
- Neste caso, extract-min leva tempo O(lgn) e há n operações extract-min, com n = |V| e m = |E|
- Cada aresta (u, v) é examinada uma vez, forçando uma operação relax(u,v,w) que custa no máximo O(lgn), quando a distância d[v] for reduzida
- Portanto o algoritmo executa em tempo O(nlgn + mlgn) = O(mlgn) assumindo que m > n

Grafos Acíclicos

- Relaxando as arestas segundo a ordem topológica de um grafo direcionado acíclico G = (V, E), leva ao cômputo da árvore de caminhos mínimos em tempo $\Theta(|V| + |E|)$

- Caminhos mínimos são sempre definidos em grafos acíclicos, pois mesmo na presença de arestas com pesos negativos, não existe ciclo de comprimento negativo

Grafos Acíclicos

```
DAG-SHORTEST-PATHS(G, w, s)
1   topologically sort the vertices of G
2   INITIALIZE-SINGLE-SOURCE(G, s)
3   for each vertex u, taken in topologically sorted order
4       do for each vertex v ∈ Adj[u]
5           do RELAX(u, v, w)
```

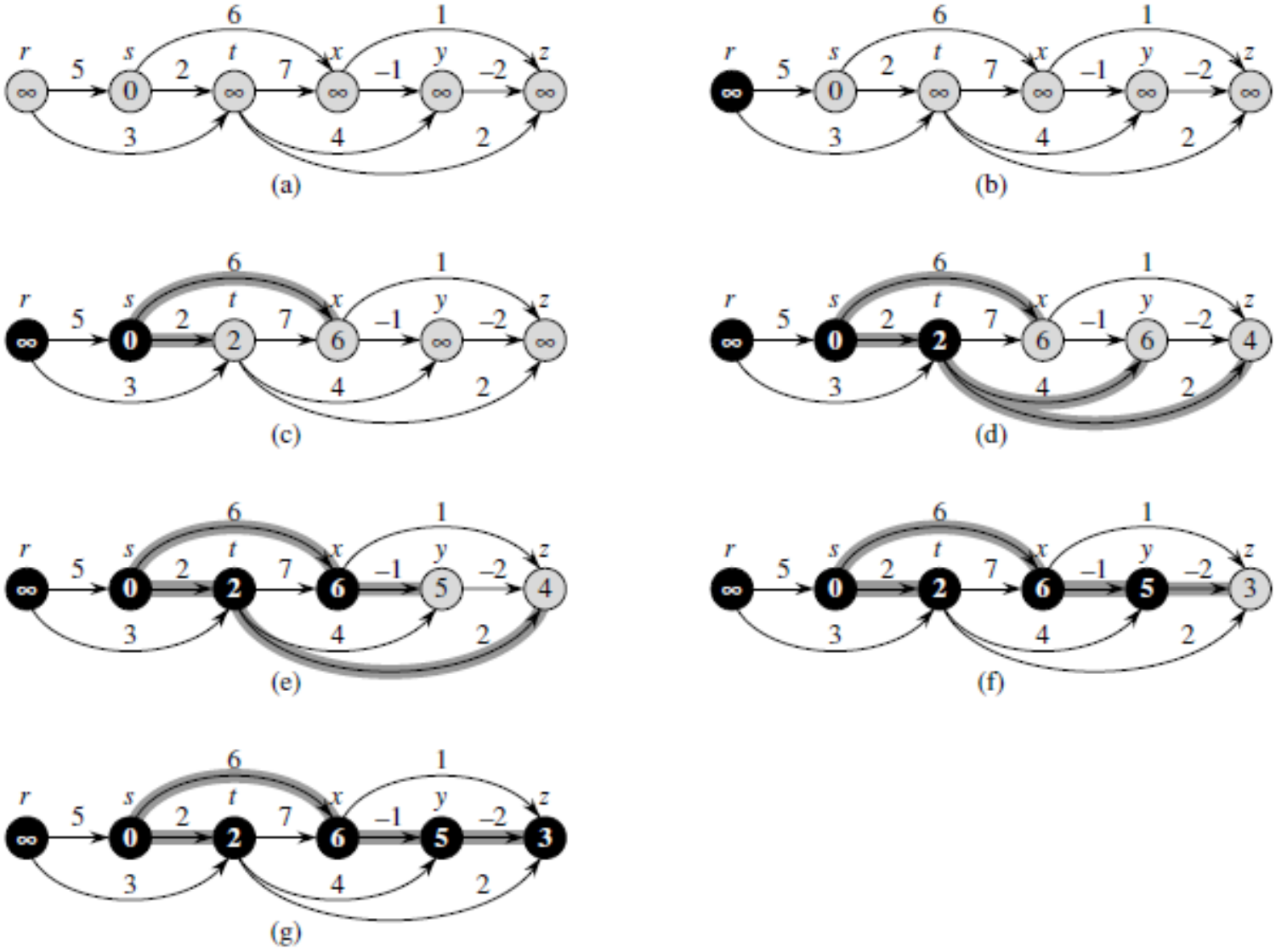


Figure 24.5 The execution of the algorithm for shortest paths in a directed acyclic graph. The vertices are topologically sorted from left to right. The source vertex is s. The d values are shown within the vertices, and shaded edges indicate the π values. (a) The situation before the first iteration of the **for** loop of lines 3–5. (b)–(g) The situation after each iteration of the **for** loop of lines 3–5. The newly blackened vertex in each iteration was used as u in that iteration. The values shown in part (g) are the final values.

Grafos Acíclicos – Complexidade

- Ordenação topológica pode ser realizada em tempo $\Theta(|V|+|E|)$ utilizando o algoritmo de busca em profundidade
- Cada vértice é examinado exatamente uma vez, quando sua lista de adjacência é examinada e a operação Relax é executada
- Logo, o laço 3-5 executa em tempo $\Theta(|V| + |E|)$
- O tempo total de execução é, portanto, $\Theta(|V| + |E|)$