Tópicos de Programação

Arthur Casals
(arthur.casals@usp.br)

IME - USP

Aula 11:

- Análise de algoritmos

Na aula passada...

Árvores geradoras:

- É um subgrafo de G que conecta, sem ciclos, todos os vértices deste grafo.

- "Uma árvore de um grafo não-dirigido G é ***geradora*** se contém todos os vértices de G."*

***https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/spanningtrees.html*

Na aula passada...

Árvores geradoras mínimas:

- **Definição:** "Uma árvore geradora mínima de G é qualquer árvore geradora de G que tenha custo mínimo.."*
- Algoritmos: **Kruskal** e **Prim**

**https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/mst.html*

Análise de algoritmos

Relembrando:

- **Passeio:** Qualquer sequência $(s_0, s_1, \ldots, s_k)$ tal que $(s_i, s_{i+1}) \in A_G$, $0 \leq i < k$
- **Caminho:** Qualquer passeio no qual nenhum dos vértices ocorre mais de uma vez
- **Passeio fechado:** É um passeio que começa e termina no mesmo vértice

Análise de algoritmos

Relembrando:

- **Circuito:** É um passeio fechado que não contém **arestas** repetidas
- **Circuito simples:** É um circuito que não contém **vértices** repetidos a não ser pelo primeiro/último

- Ciclos: É um caminho de comprimento maior ou igual a 1 entre um vértice e ele mesmo

Análise de algoritmos

Relembrando:

- Ciclo Euleriano:

- É um circuito que contém todos os vértices e arestas do grafo. Por ser um circuito, cada aresta é percorrida apenas uma vez.

Análise de algoritmos

Caminhos em grafos:

- Problema do caminho Euleriano
- Problema do caminho mínimo*
  - Algoritmo de Dijkstra
  - Algoritmo de Bellman-Ford

**adaptação: PCS3110, 2017, EPUSP*

Análise de algoritmos

Caminhos em grafos: Problema do caminho Euleriano

- "Dada uma representação gráfica de um grafo, determinar, caso exista, um caminho que passe por todas as arestas, sem repetir aresta começando e terminando no mesmo vértice."

Análise de algoritmos

Caminhos em grafos: Problema do caminho Euleriano

Exemplo: problema do carteiro chinês

- "Como passar pelas ruas da cidade uma única vez?"
- Curiosidade: cidade de *Königsberg*

Análise de algoritmos

Caminhos em grafos: Problema do caminho Euleriano

- (1) É possível?
- (2) Como fazer?

Análise de algoritmos

Caminhos em grafos: Problema do caminho Euleriano

- (1) Teorema dos caminhos Eulerianos: "Um grafo possui um caminho Euleriano se e somente se não possuir nenhum ou possuir no máximo dois vértices de grau ímpar. Caso não possua vértices de grau ímpar, o caminho pode começar e terminar em qualquer vértice. Caso contrário, o caminho obrigatoriamente começa em um vértice ímpar e terminar no outro."

Análise de algoritmos

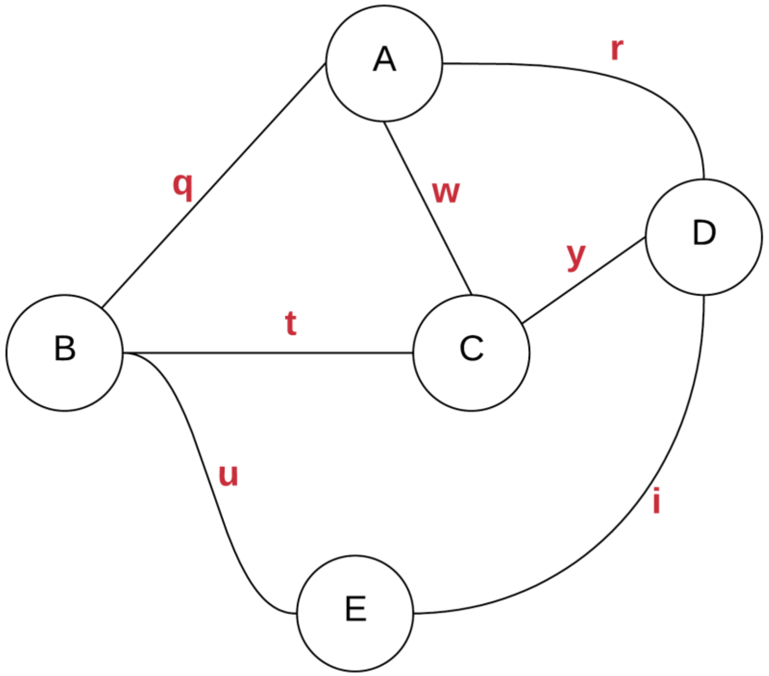
Caminhos em grafos: Problema do caminho Euleriano

- Para determinar a validade do Teorema dos caminhos eulerianos, podemos utilizar uma **matriz de incidência**

Fundamentos de estruturas de dados

Grafos:

- Criando o grafo a partir da **matriz de incidência**



$$
\begin{array}{c|c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} q \\ w \\ r \\ t \\ y \\ u \\ i \end{array} &
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1
\end{bmatrix}
\end{array}
$$

Grau: 3 3 3 3 2

Não possui caminho Euleriano!

Análise de algoritmos

Caminhos em grafos: Problema do caminho Euleriano

- (2) Como achar um caminho Euleriano?

Análise de algoritmos

Caminhos em grafos: Problema do caminho Euleriano

- (2) Como achar um caminho Euleriano?
  - Se o grafo não possuir peso em suas arestas, o caminho Euleriano pode ser determinado através de **busca em largura**.
  - ...e se eu quiser considerar os **pesos** das arestas?

Análise de algoritmos

Caminhos em grafos: Problema do caminho mínimo

- "Suponha um grafo simples, ponderado (com pesos em suas arestas) e conexo, no qual **todos os pesos são positivos**. Como encontrar um caminho cujo somatório entre os pesos seja o menor possível?"

Análise de algoritmos

Caminhos em grafos: Problema do caminho mínimo

- Única origem: determinar o menor caminho entre um **vértice origem** (dado) e todos os demais vértices do grafo
- Único destino: determinar o menor caminho entre cada um dos vértices do grafo e um **vértice destino** (dado)
- Par: menor caminho entre **dois vértices dados**
- Todos os pares: menor caminho entre **cada par de vértices** do grafo

Análise de algoritmos

Caminhos em grafos: Algoritmo de Dijkstra

- Problema de **única origem**
- Grafo de entrada possui pesos não-negativos
- Grafo pode ser dirigido ou não-dirigido

Análise de algoritmos

Caminhos em grafos: Algoritmo de Dijkstra

› Atributos adicionais nos vértices:

- **caminho**: estimativa de caminho mínimo
- **predecessor**: vértice anterior no caminho

› Escolhe sempre o vértice com menor estimativa de caminho mínimo com caminho ainda não determinado

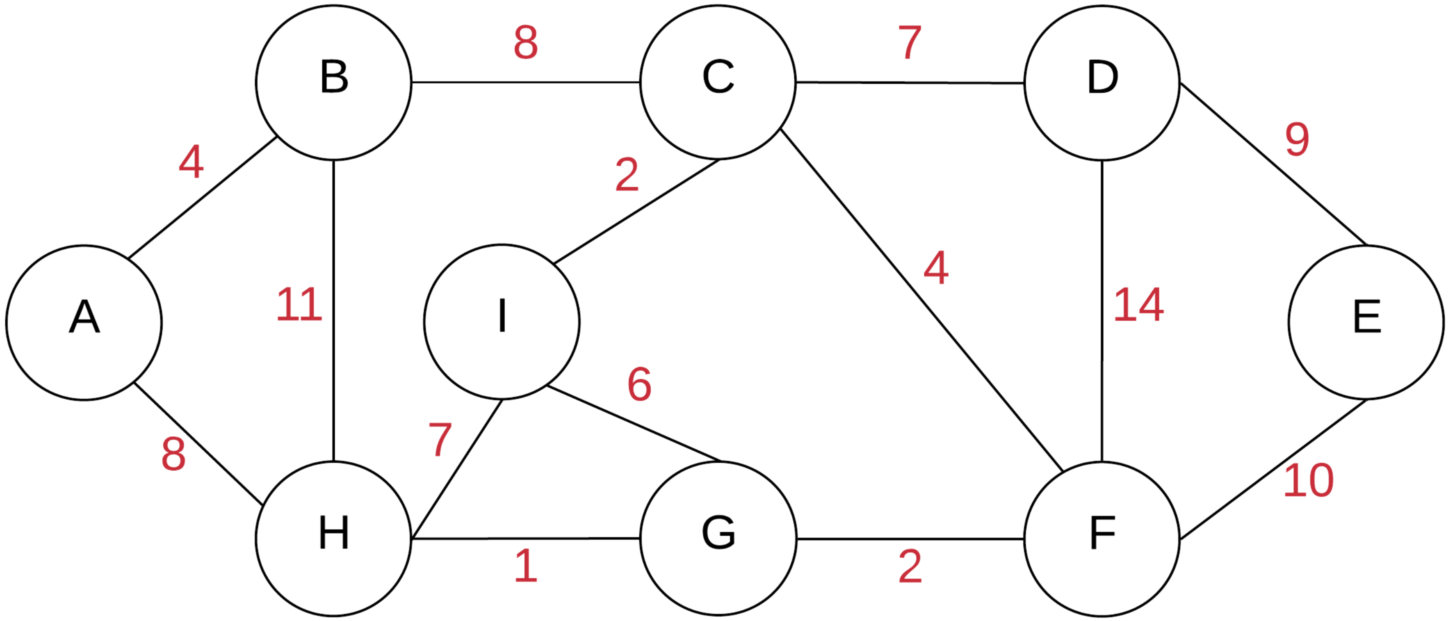
Análise de algoritmos

Caminhos em grafos: Algoritmo de Dijkstra

1. Atribua 0 à estimativa de caminho mínimo do vértice origem e infinito (∞) aos demais vértices
2. Atribua **null** ao predecessor de todos os vértices
3. Enquanto houver vértice não visitado
   - Seja k um vértice não visitado cuja estimativa seja a menor dentre todos os vértices não visitados
     - Marque k como **visitado**
   - Para todo vértice j ainda não visitado que seja adjacente a k, faça:
     - Some a estimativa do vértice k com o custo da aresta (k,j)
     - Se a soma for menor que a estimativa anterior para o vértice j, **substitua a estimativa** pelo resultado da soma e anote k como predecessor de j

Análise de algoritmos

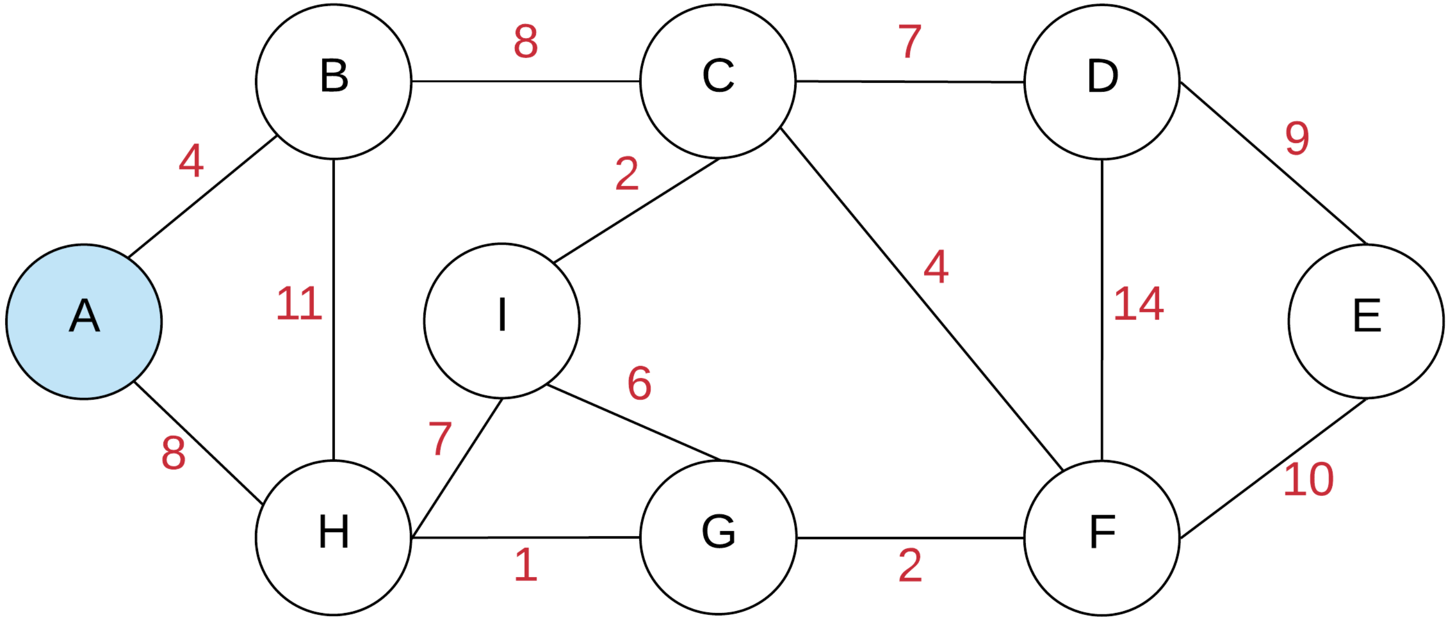
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	∞	∞	∞	∞	∞	∞	∞	∞
predecessor	null	null	null	null	null	null	null	null	null

Análise de algoritmos

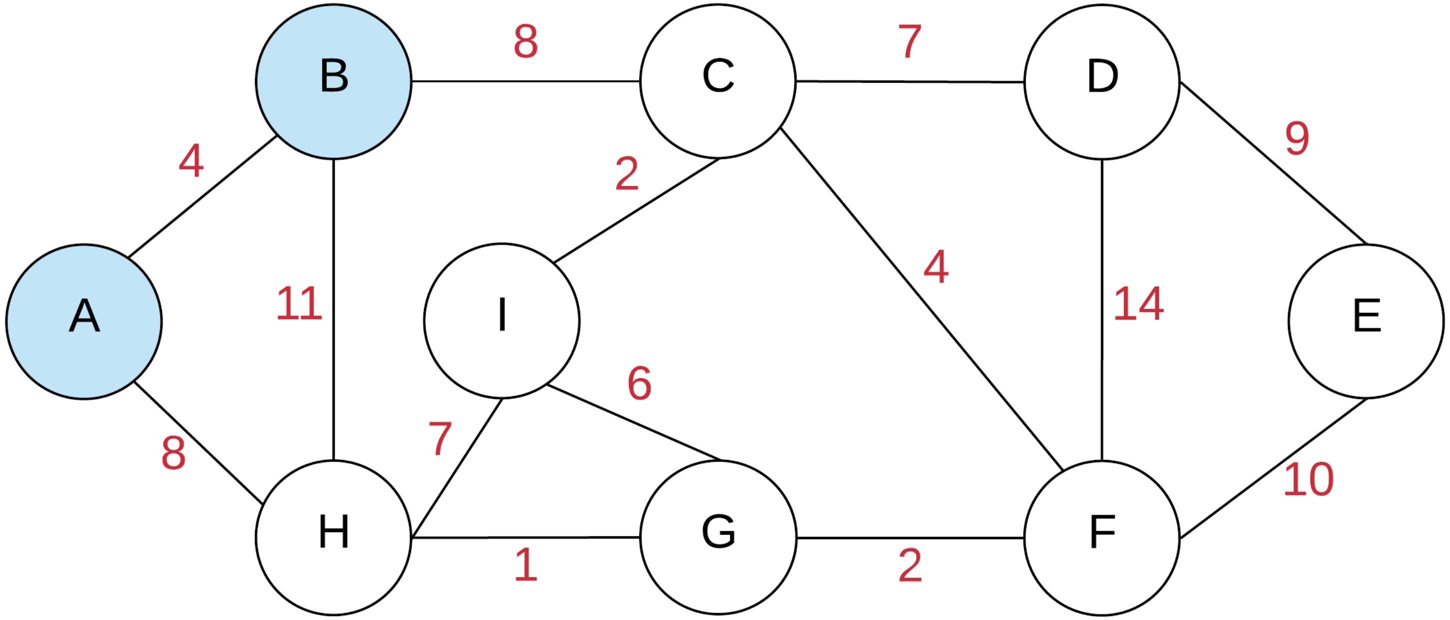
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	∞	∞	∞	∞	∞	8	∞
predecessor	null	A	null	null	null	null	null	A	null

Análise de algoritmos

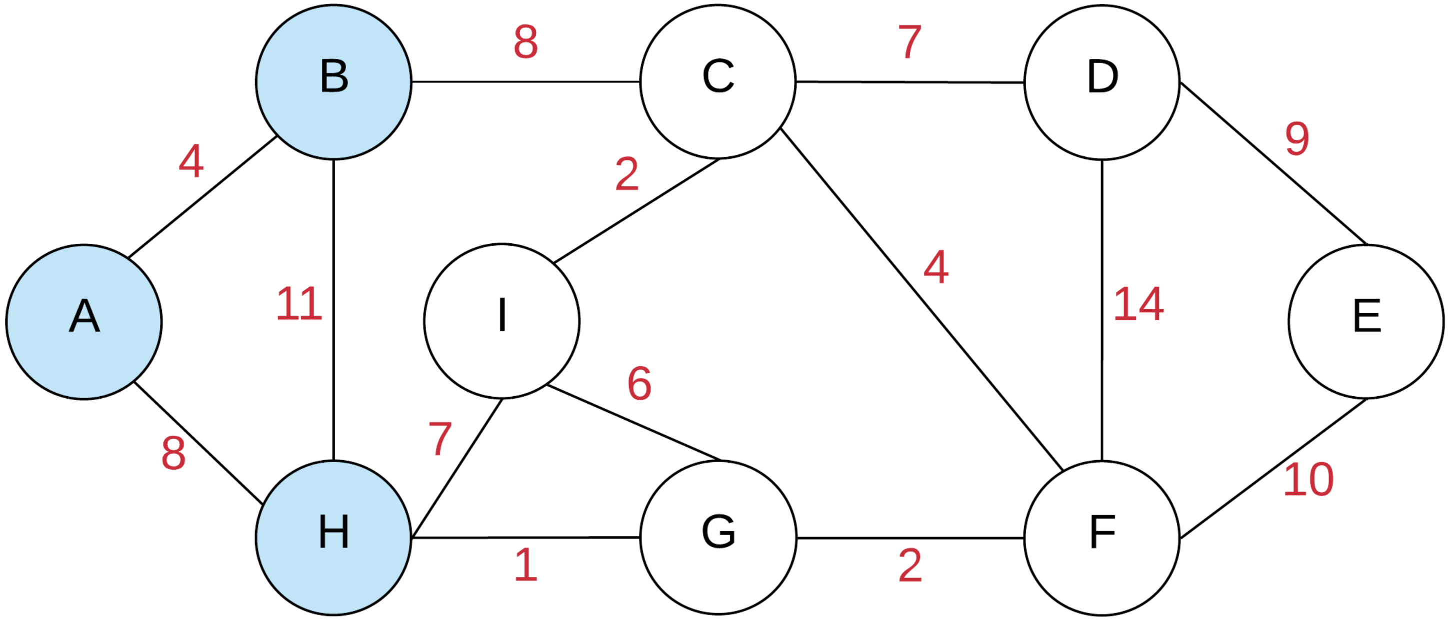
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	∞	∞	∞	∞	*8*	∞
predecessor	null	A	B	null	null	null	null	*A*	null

Análise de algoritmos

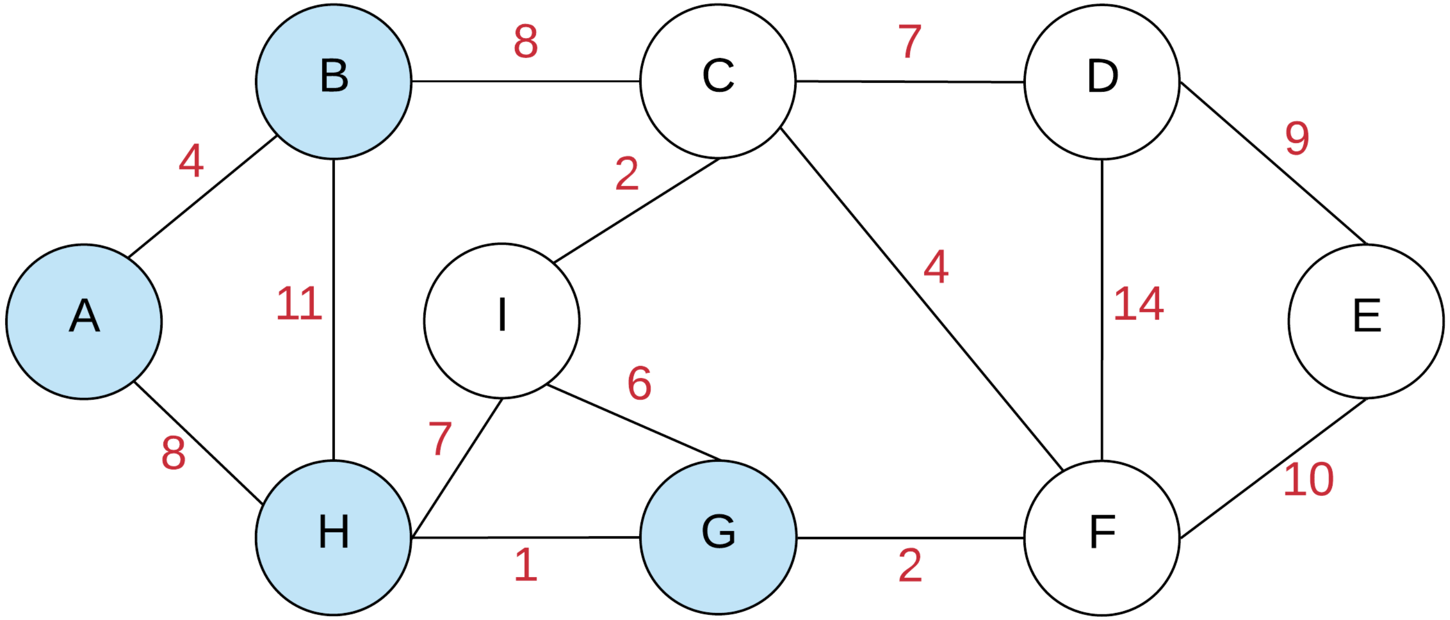
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	∞	∞	∞	9	8	15
predecessor	null	A	B	null	null	null	H	A	H

Análise de algoritmos

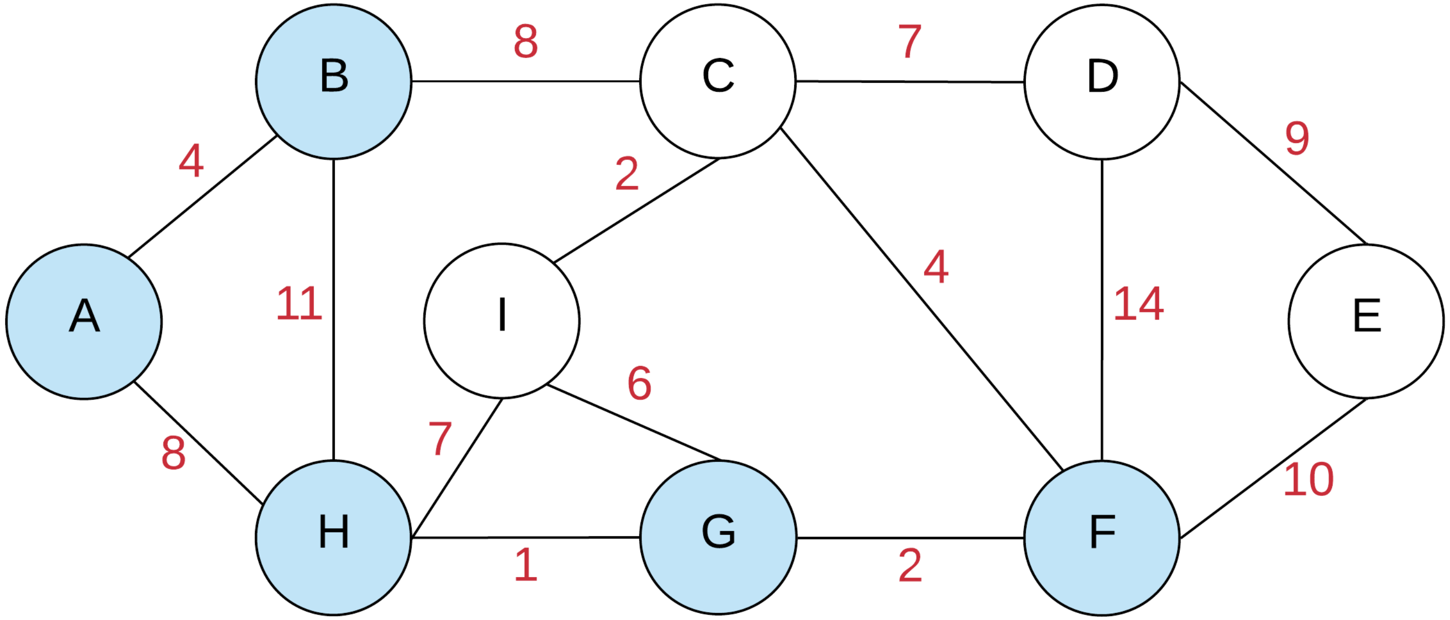
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	∞	∞	11	9	8	*15*
predecessor	null	A	B	null	null	G	H	A	*H*

Análise de algoritmos

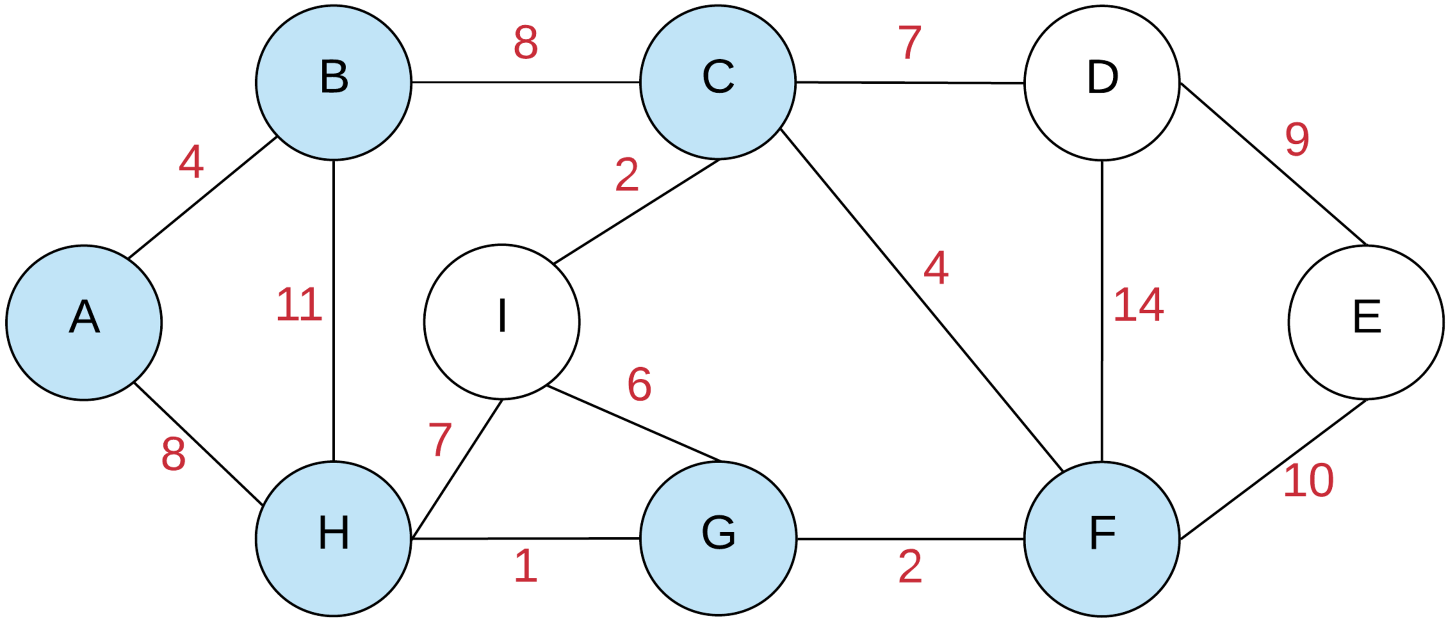
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	25	21	11	9	8	15
predecessor	null	A	B	F	F	G	H	A	H

Análise de algoritmos

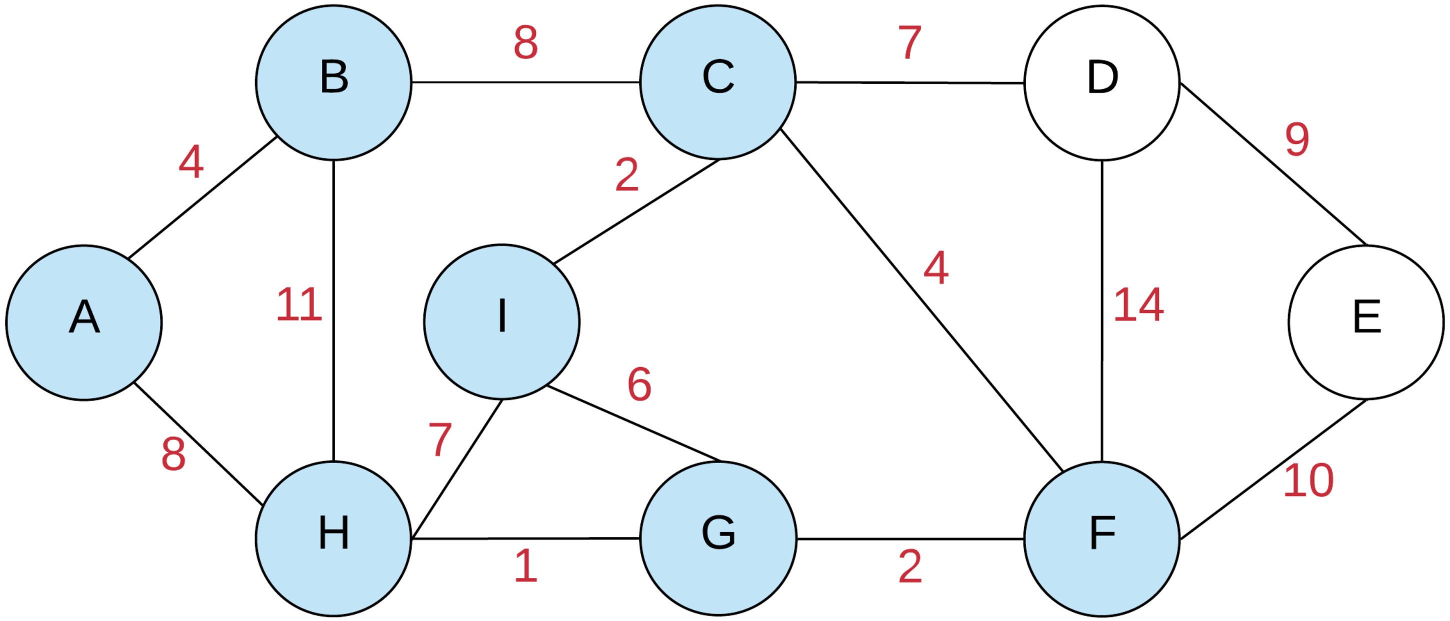
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	*19*	21	11	9	8	*14*
predecessor	null	A	B	*C*	F	G	H	A	*C*

Análise de algoritmos

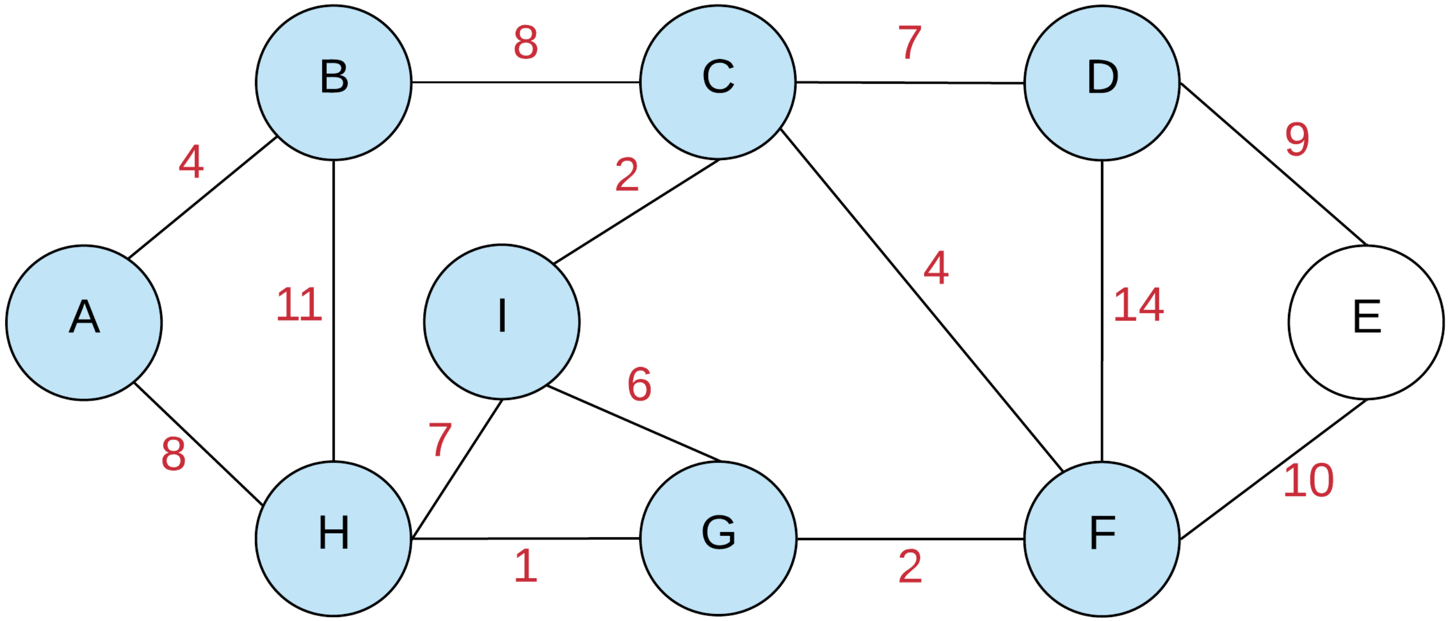
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	19	21	11	9	8	14
predecessor	null	A	B	C	F	G	H	A	C

Análise de algoritmos

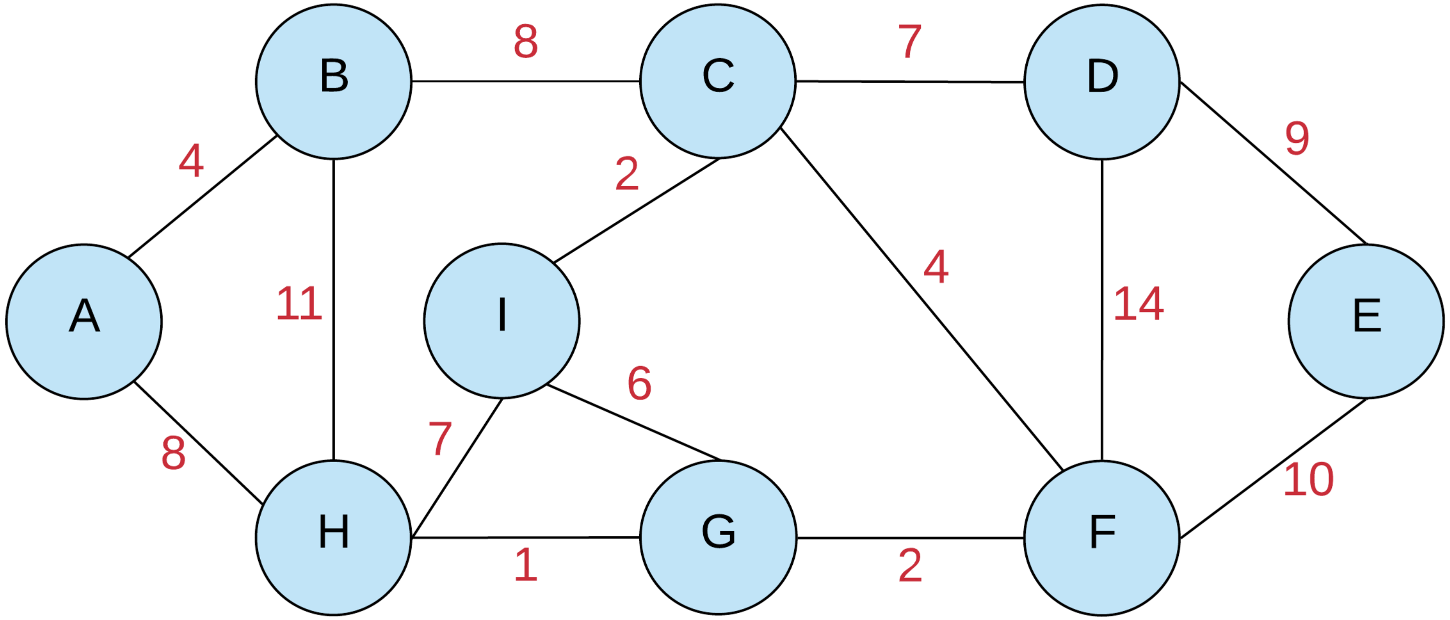
Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	19	21	11	9	8	14
predecessor	null	A	B	C	F	G	H	A	C

Análise de algoritmos

Caminhos em grafos: Algoritmo de Dijkstra



vértices	A	B	C	D	E	F	G	H	I
caminho	0	4	12	19	21	11	9	8	14
predecessor	null	A	B	C	F	G	H	A	C

Análise de algoritmos

Caminhos em grafos: Algoritmo de Dijkstra

```
Dijkstra(G, origem)
for each u ∈ G.V // Inicializa caminho
    u.caminho = ∞
    u.predecessor = null
origem.caminho = 0
Seja Q uma Fila de Prioridade
Q = G.V // Coloca em Q todos os vértices
while not Queue-Empty(Q)
  menor = Extrair-Menor(Q) // Q = Q – {menor}
  for each v ∈ G.Adjacencia[menor]
    if v.caminho > peso(G, menor, v) + menor.caminho
       v.predecessor = menor
       v.caminho = peso(G, menor, v) + menor.caminho
       Altera-Prioridade(Q, v)
```

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford

› Problema de **única origem**

› Grafo **dirigido e ponderado**

› Grafo de entrada **pode ter pesos negativos**

› Atributos adicionais nos vértices:

- **caminho**: estimativa de caminho mínimo
- **predecessor**: vértice anterior no caminho

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford

› Devolve um **valor booleano** que indica se existe ou não um ciclo de peso negativo que pode ser alcançado na origem

- Se existe, não há solução
- Se não existe, calcula todos os caminhos mínimos a partir da origem dada

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford

1. Atribua 0 à estimativa de caminho mínimo do vértice origem e infinito (∞) aos demais vértices
2. Atribua **null** ao predecessor de todos os vértices
3. Faça |G.V| - 1 vezes
   - Para cada **aresta (u,v)**, verifique se o uso dela faz com que o caminho até **v** seja menor
   - Se for, atualize o **predecessor** e a **estimativa de caminho mínimo**
4. Se ao processar as arestas novamente algum caminho ficar **menor**, então há um **ciclo de peso negativo**

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford

```
Conjunto-De-Arestas(G)
Seja Arestas um conjunto de arestas (u,v)
Arestas = { } // inicialmente Arestas está vazio
for each x ∈ G.V
   for each y ∈ G.V
     if G.Adjacencia[x][y] ≠ ∞
        if (x, y) ∉  Arestas
           Arestas = Arestas ∪ (x, y)
return Arestas
```

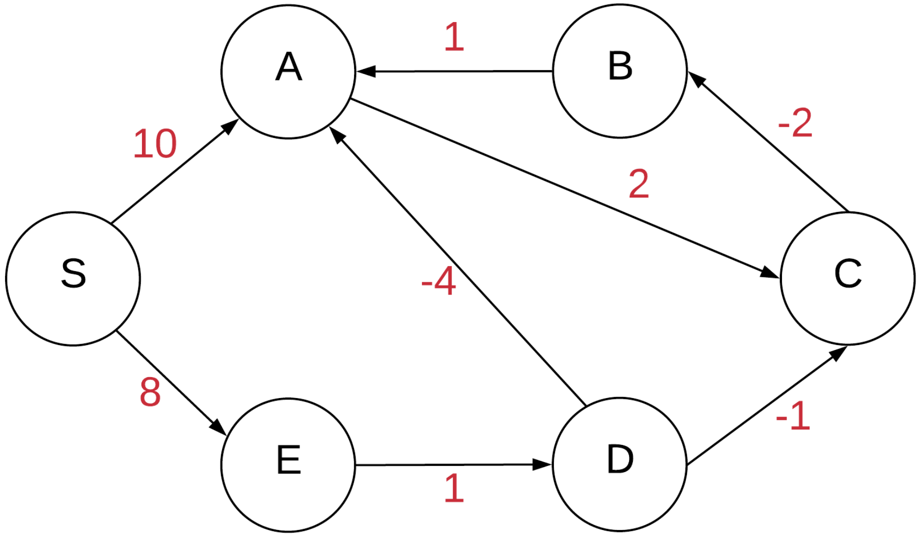
Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford

```
Bellman-Ford(G, origem)
for each u ∊ G.V // Inicializa caminho
   u.caminho = ∞;   u.predecessor = NIL
origem.caminho = 0
Arestas = Conjunto-De-Arestas(G)
for i=1 to |G.V|-1
  for each (u,v) ∊ Arestas
    if v.caminho > u.caminho + peso(G, u, v)
       v.predecessor = u
       v.caminho = u.caminho + peso(G, u, v)
for each (u,v) ∊ Arestas
  if v.caminho > u.caminho + peso(G, u, v)
    return FALSE
return TRUE
```

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford



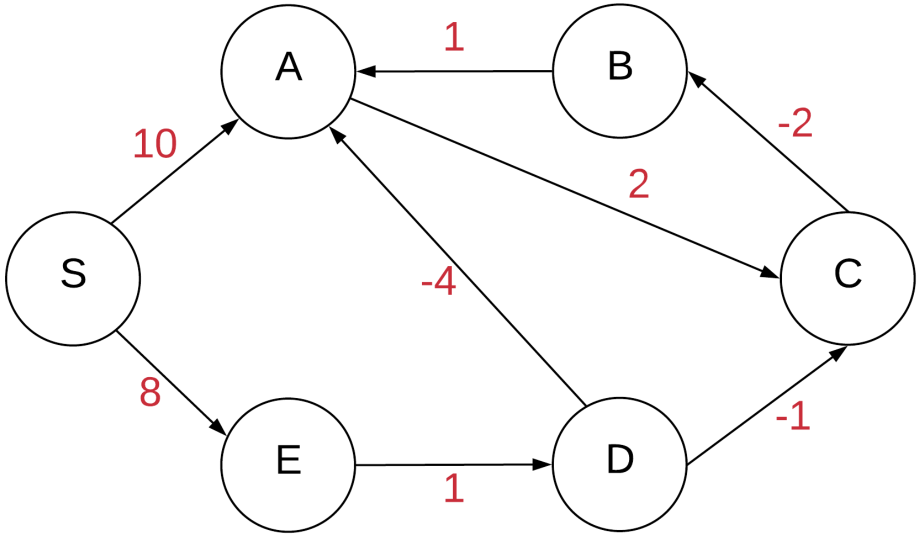
Arestas: (s,a), (s,e), (a,c), (b,a), (c,b), (d,c), (d,a), (e,d)

vértices	S	A	B	C	D	E
caminho	0	∞	∞	∞	∞	∞
predecessor	null	null	null	null	null	null

Início

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford



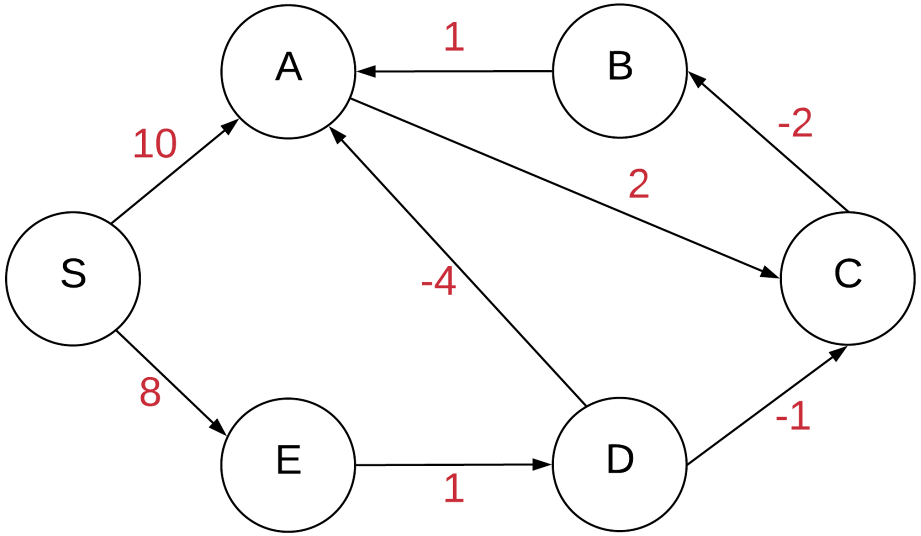
Arestas: (s,a), (s,e), (a,c), (b,a), (c,b), (d,c), (d,a), (e,d)

vértices	S	A	B	C	D	E
caminho	0	10	10	12	9	8
predecessor	null	S	C	A	E	S

Após a primeira iteração

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford



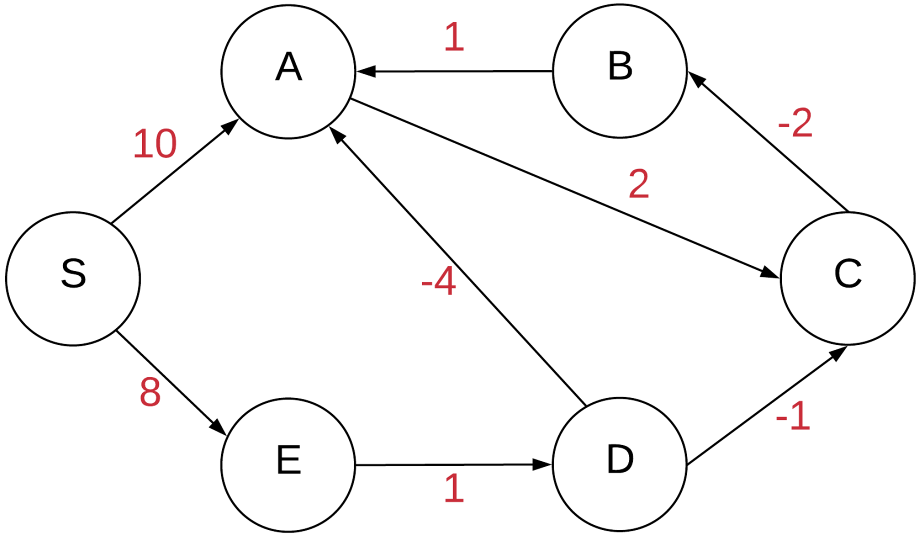
Arestas: (s,a), (s,e), (a,c), (b,a), (c,b), (d,c), (d,a), (e,d)

vértices	S	A	B	C	D	E
caminho	0	5	10	8	9	8
predecessor	null	D	C	D	E	S

Após a segunda iteração

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford



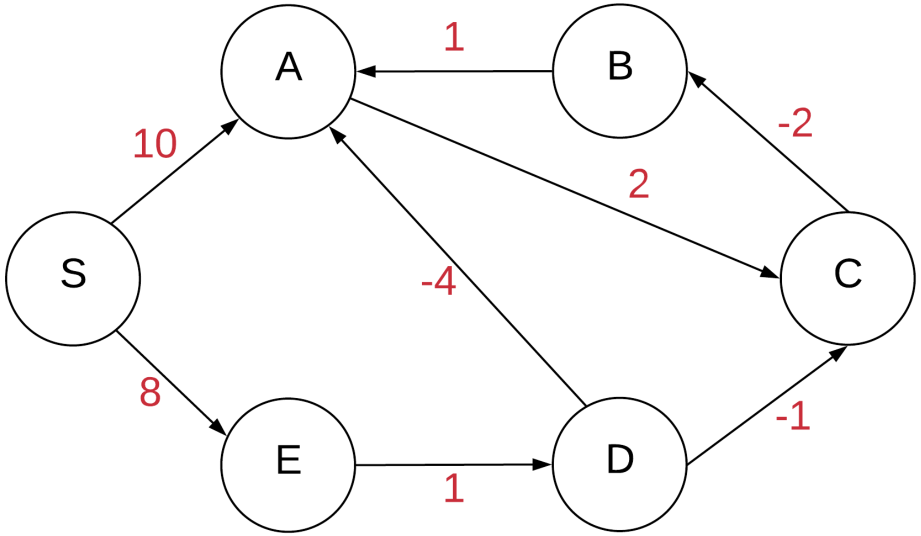
Arestas: (s,a), (s,e), (a,c), (b,a), (c,b), (d,c), (d,a), (e,d)

vértices	S	A	B	C	D	E
caminho	0	5	5	7	9	8
predecessor	null	D	C	A	E	S

Após a terceira iteração

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford



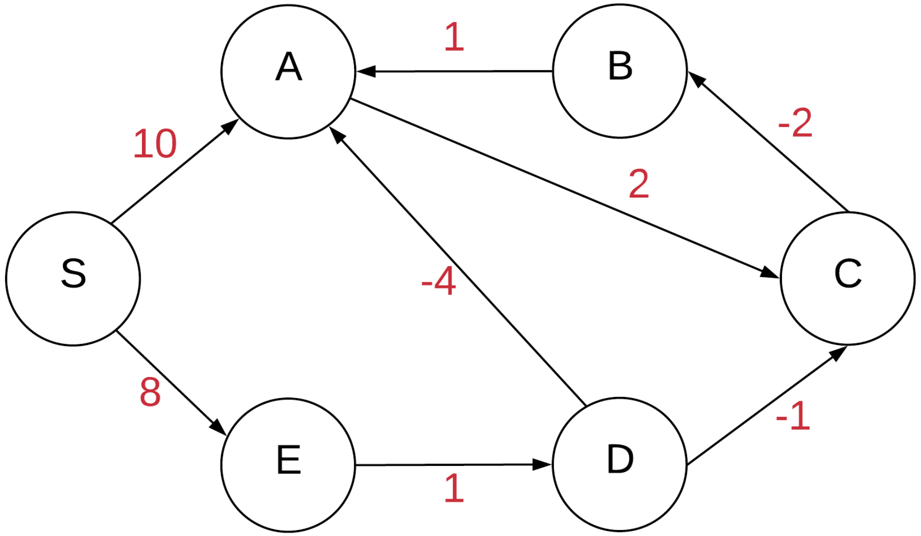
Arestas: (s,a), (s,e), (a,c), (b,a), (c,b), (d,c), (d,a), (e,d)

vértices	S	A	B	C	D	E
caminho	0	5	5	7	9	8
predecessor	null	D	C	A	E	S

Após a quarta iteração (valores não mudam)

Análise de algoritmos

Caminhos em grafos: Algoritmo de Bellman-Ford



Arestas: (s,a), (s,e), (a,c), (b,a), (c,b), (d,c), (d,a), (e,d)

vértices	S	A	B	C	D	E
caminho	0	5	5	7	9	8
predecessor	null	D	C	A	E	S

Após a quinta iteração (valores não mudam)

Análise de algoritmos

Caminhos em grafos:

- Encontrar caminhos mínimos é útil em diversas situações
- Se não houver ponderação, **busca em largura**!
  - Comprimento do caminho: número de arestas desde a origem
- Grafo dirigido e ponderado **com pesos negativos:** Bellman-Ford
- Grafo dirigido e ponderado **com pesos positivos:** Djikstra
  - **É mais eficiente!**

Análise de algoritmos

Exercício: para 25/01 **Djikstra(G,A)**

(Submeter pela página do curso)

Vale por 3 exercícios diários