



Universidade de Brasília

# Grafos

Algoritmos e Estruturas de Dados

Prof. Daniel Guerreiro e Silva

# Roteiro

- Definição e Representação
- Percurso em Grafos
- Caminhos mínimos
- Árvore geradora mínima

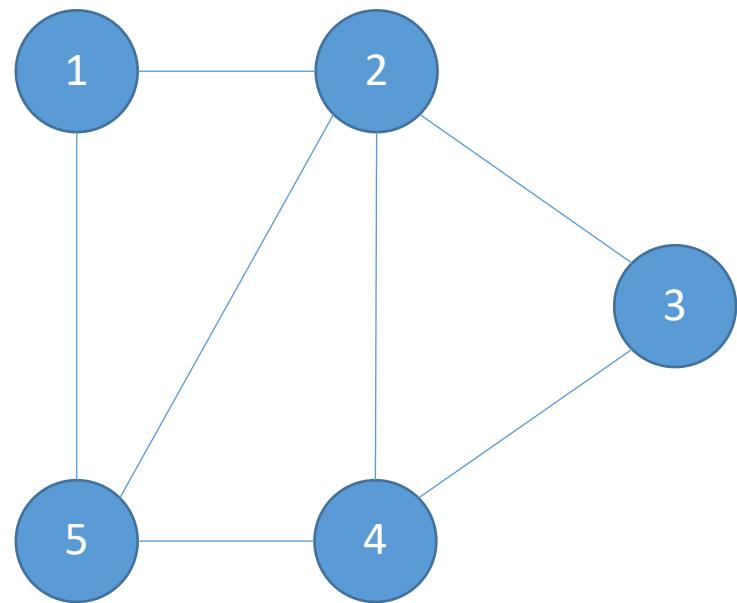
Leitura sugerida: Seções 8.1 a 8.5 do livro-texto  
(Drozdek)

# Definição e Representação

# Definição

Um grafo é uma coleção de vértices (nós) e as conexões entre eles.

- Um grafo  $G = (V, E)$  é composto por um conjunto  $V$  de vértices e um conjunto  $E$  de arestas
- O número de vértices e de arestas é denotado por  $|V|$  e  $|E|$ , respectivamente



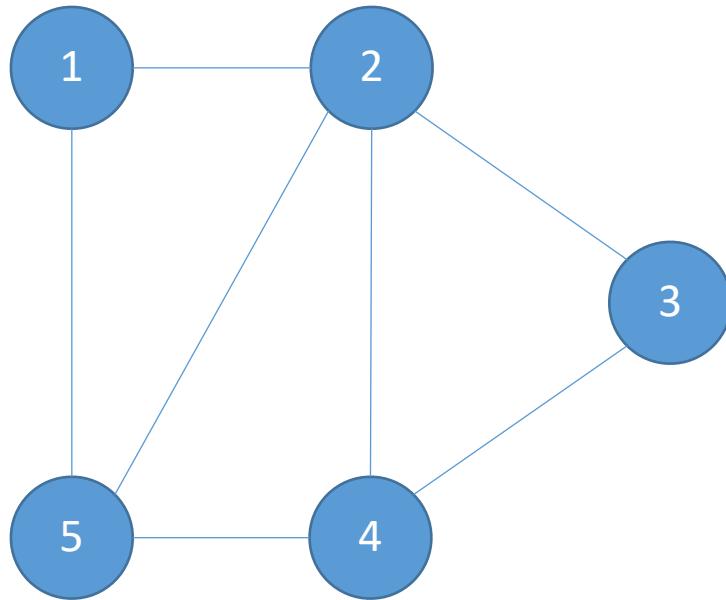
# Grafo não-dirigido

- Vértices:

$$V = \{1, 2, 3, 4, 5\}$$

- Arestas:

$$E = \left\{ \{1,2\}, \{1,5\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{4,5\} \right\}$$



# Grafo dirigido (digrafo)

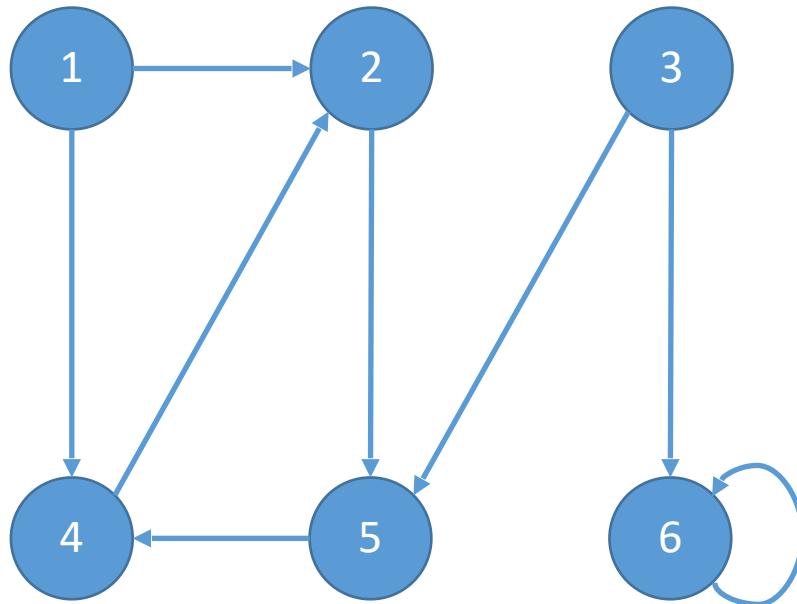
- Vértices:

$$V = \{1, 2, 3, 4, 5, 6\}$$

- Arestas:

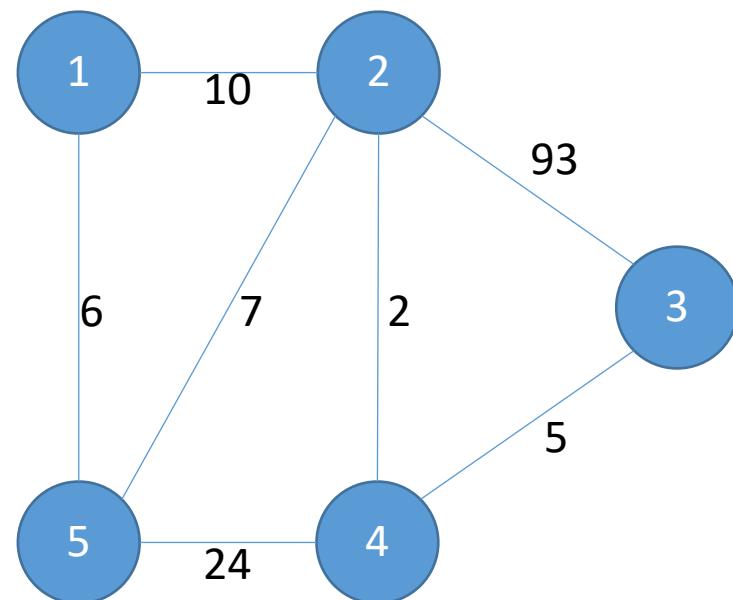
$$E = \{(1, 2), (1, 4), (2, 5), (3, 5), (3, 6), (4, 2), (5, 4), (6, 6)\}$$

Repare que  
 $(v_i, v_j) \neq (v_j, v_i)$



# Grafo ponderado

- É um grafo onde cada aresta tem um valor numérico (peso) associado
- Dependendo do problema, o valor pode representar custo, distância, comprimento, etc.



# Representação de grafos

Há duas formas principais de se representar um grafo em um programa de computador

## 1. Lista de adjacências

Para cada vértice  $u \in V$ , contrói-se uma lista com todos os vértices  $v$  tais que existe uma aresta  $(u, v) \in E$ , i.e. todos os vértices adjacentes a  $u$

# Representação de grafos

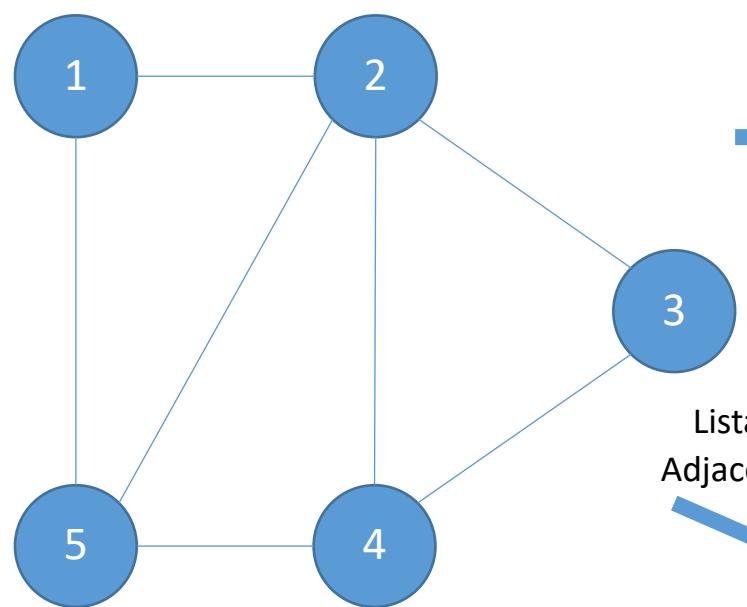
Há duas formas principais de se representar um grafo em um programa de computador

## 2. Matriz de adjacências

$\mathbf{A} = \{a_{ij}\}$  é uma matriz  $|V| \times |V|$  tal que

$$a_{ij} = \begin{cases} 1, & (i, j) \in E \\ 0, & \text{caso contrário} \end{cases}$$

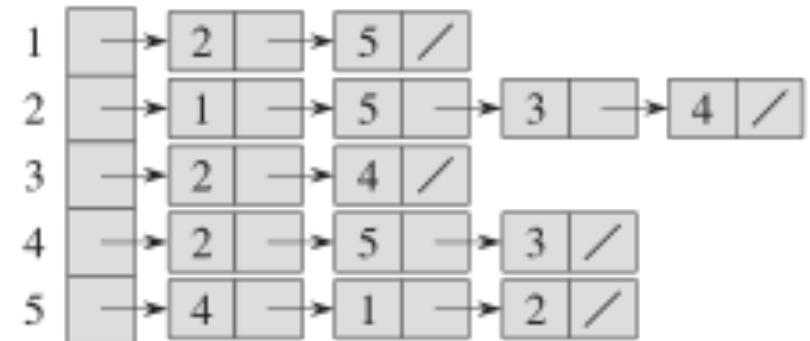
# Representação de grafos



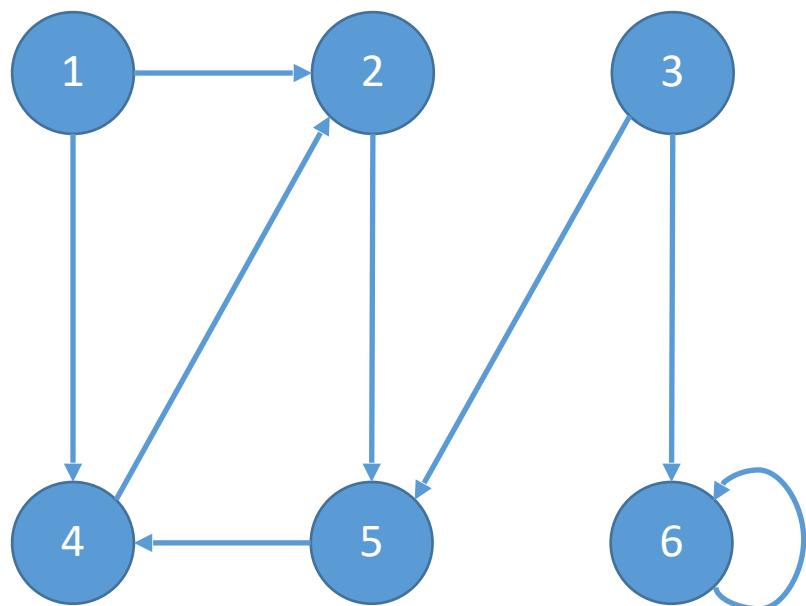
Matriz de  
Adjacências

Lista de  
Adjacências

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



# Representação de grafos



Matriz de  
Adjacências

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Lista de  
Adjacências

1	→	2	→	4	/
2	→	5	/		
3	→	6	→	5	/
4	→	2	/		
5	→	4	/		
6	→	6	/		

# Representação de grafos

- Estude os aspectos de representação nos arquivos de exemplo grafo.h e grafo.cpp

# Percorso em Grafos

# Percursos em um grafo

- Como em árvores, percorrer um grafo consiste em visitar todos os seus vértices somente uma vez.
- Tem-se ao final um caminho dado por uma sequência de arestas

$$(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$$

ou de vértices

$$v_1, v_2, \dots, v_{n-1}, v_n.$$

- Para evitar repetições, "pintamos" ou "marcamos" os vértices assim que são visitados
- Duas abordagens são consideradas: busca em profundidade (depth-first) e busca em largura (breadth-first)

# Percorso em profundidade (*depth-first*)

1. Visite o vértice  $v$
2. Visite vértice adjacente a  $v$ , ainda não visitado
  - Se não há vértices adjacentes ou todos foram visitados, retorne ao predecessor
3. Quando chegar ao primeiro vértice em que o algoritmo começou, se houver vértice não visitado em  $G$ , volte ao passo 1 com ele.

# Percorso em profundidade (*depth-first*)

```

DFS(v)
    num(v) = i++;
    for todo vértice u adjacente a v
        if num(u) é 0
            acrescente aresta(vu) a arestas;
            DFS(u);

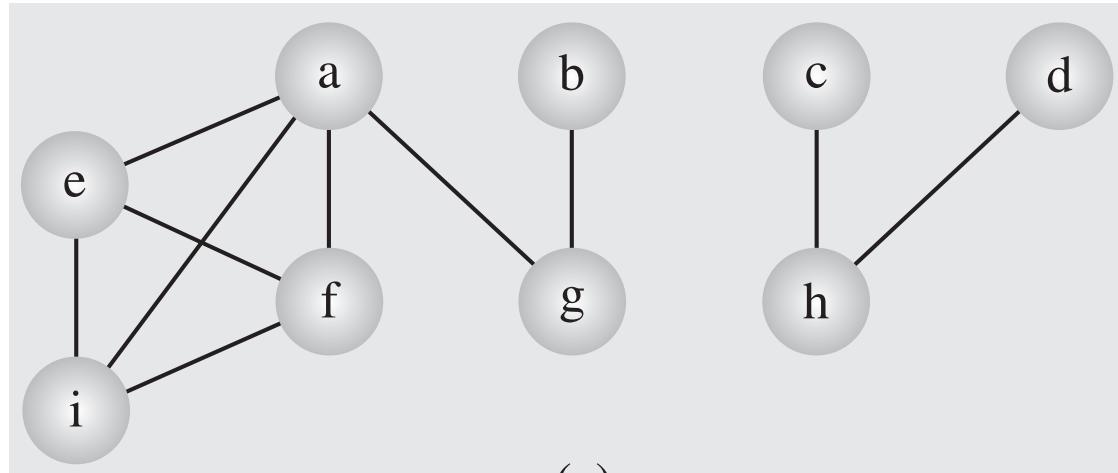
```

```

depthFirstSearch()
for todo vértice v
    num(v) = 0;
    arestas = null;
    i = 1;
while há vértice v tal que num(v) é 0
    DFS(v);
retorne arestas

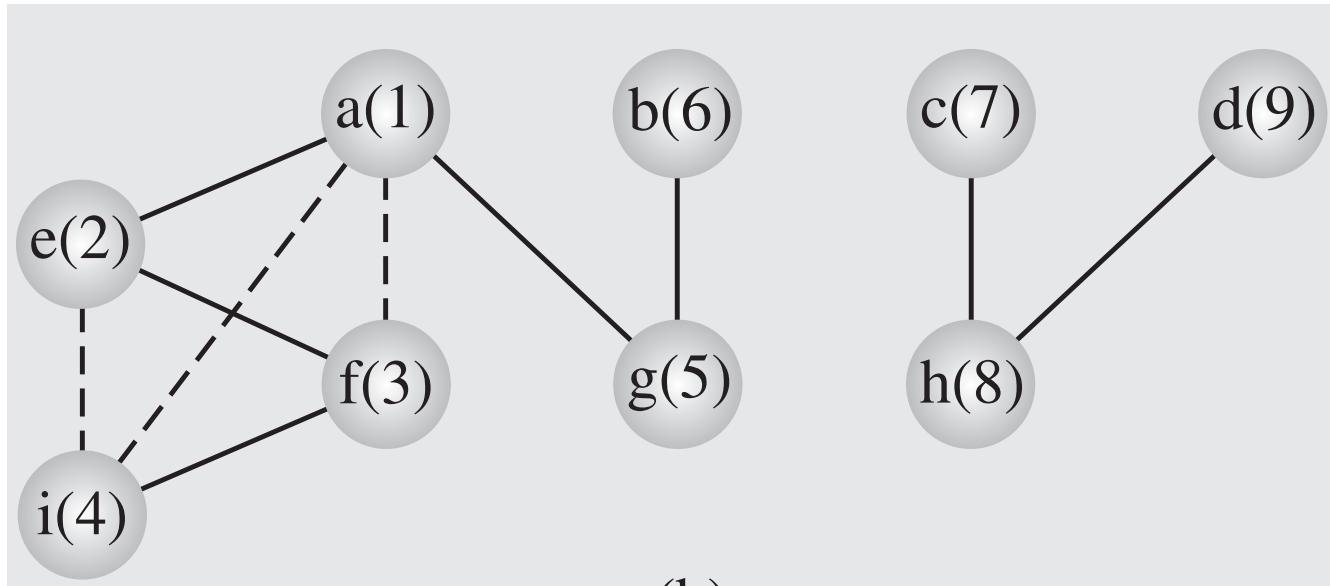
```

# *depth-first - exemplo*



```
depthFirstSearch ()  
    for todo vértice v  
        num(v) = 0;  
        arestas = null;  
        i = 1;  
        while há vértice v tal que num(v) é 0  
            DFS(v);  
        retorno arestas  
  
DFS (v)  
    num(v) = i++;  
    for todo vértice u adjacente a v  
        if num(u) é 0  
            acrescente aresta(vu) a arestas;  
            DFS(u);
```

# *depth-first - exemplo*



Os números em parênteses representam a ordem de visitação  
e as arestas contínuas compreendem o percurso

Explore mais exemplos em <https://visualgo.net/dfsbfs>

# *depth-first*

- Observe no exemplo anterior que o algoritmo gera uma árvore (ou conjunto de árvores) contendo todos os vértices de  $G$
- Uma árvore que atende esta condição chama-se **árvore geradora (árvore de espalhamento)** ou ***spanning tree***
- A complexidade computacional do algoritmo é  $O(|V| + |E|)$

# Percorso em largura (*breadth-first*)

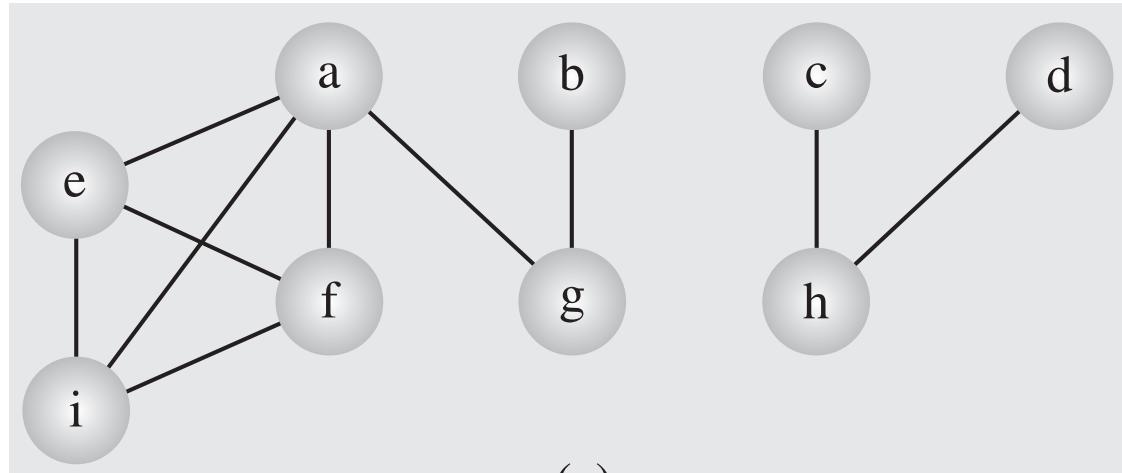
- A busca em largura parte do princípio de marcar primeiro todos os vizinhos de um vértice  $v$ , para só depois prosseguir para os demais
- Para isso, usaremos uma fila como estrutura auxiliar

# Percorso em largura (breadth-first)

```
breadthFirstSearch ()  
    for todo vértice u  
        num(u) = 0;  
        arestas = null;  
        i = 1;  
        while há vértice v tal que num(v) is 0  
            num(v) = i++;  
            enqueue(v);  
            while fila não está vazia  
                v = dequeue();  
                for todo vértice u adjacente a v  
                    if num(u) is 0  
                        num(u) = i++;  
                        enqueue(u);  
                        acrescente aresta(vu) a arestas;  
    retorno arestas;
```

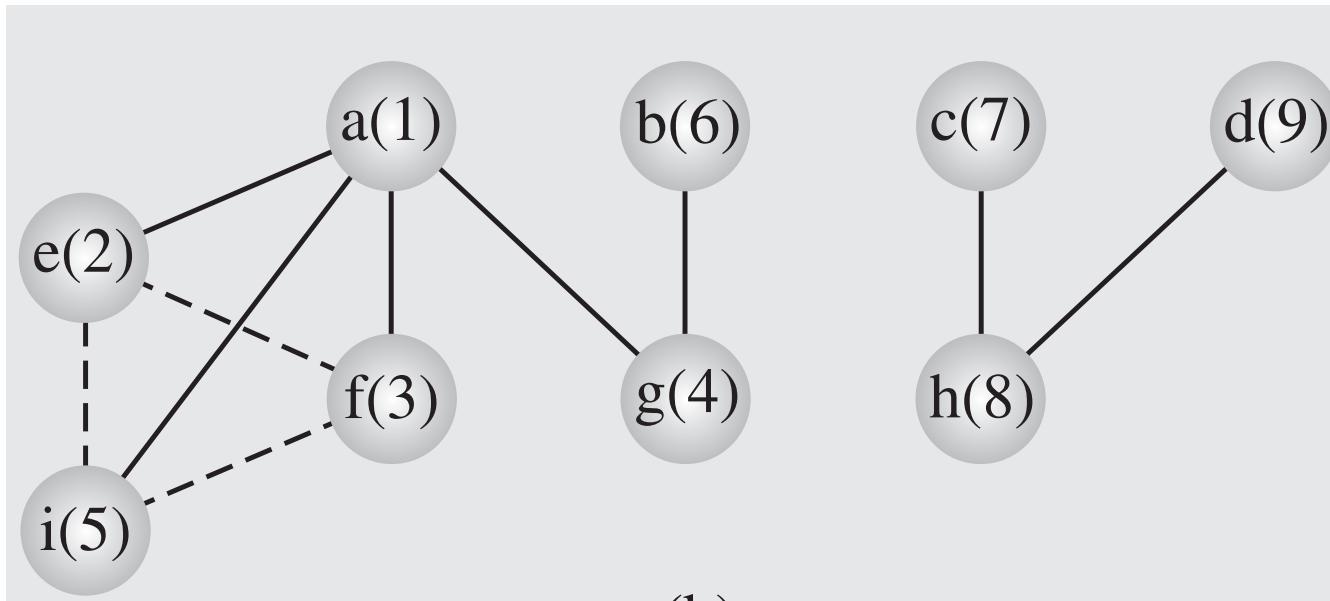
Complexidade  
 $O(|V| + |E|)$

# *breadth-first - exemplo*



```
breadthFirstSearch()
  for todo vértice u
    num(u) = 0;
    arestas = null;
    i = 1;
    while há vértice v tal que num(v) is 0
      num(v) = i++;
      enqueue(v);
      while fila não está vazia
        v = dequeue();
        for todo vértice u adjacente a v
          if num(u) is 0
            num(u) = i++;
            enqueue(u);
            acrescente aresta(vu) a arestas;
  retorno arestas;
```

# *breadth-first* - exemplo



Os números em parênteses representam a ordem de visitação  
e as arestas contínuas compreendem o percurso

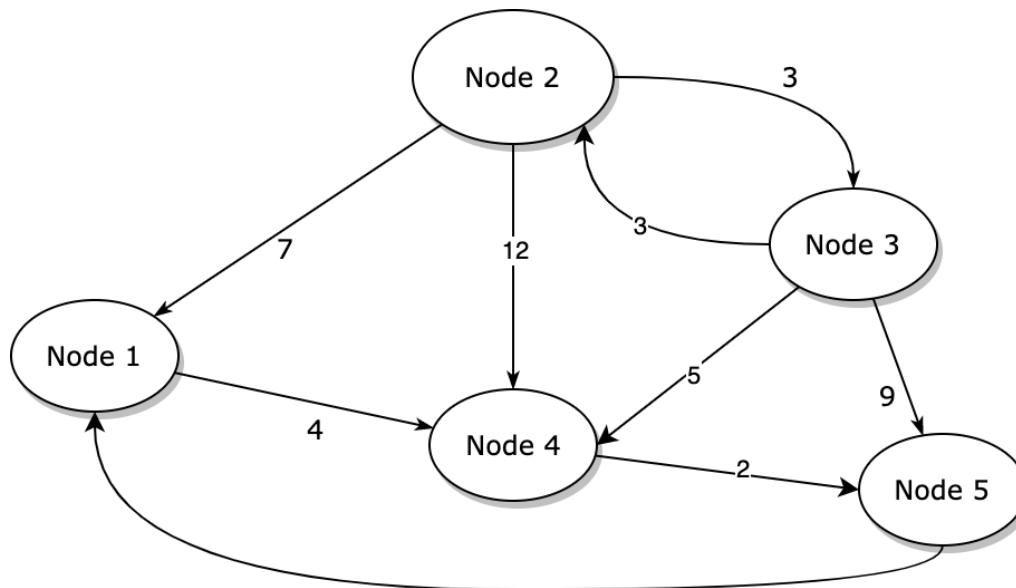
Explore mais exemplos em <https://visualgo.net/dfsbfs>

# Percorso em grafo

- Estude as implementações de percurso em profundidade e percurso em largura nos arquivos grafo.h, grafo.cpp

# Exercício de Assimilação de Conceitos

- Com base na biblioteca grafo.h e grafo.cpp, crie um programa para representar o grafo abaixo e exiba sua lista de adjacências, o resultado das operações de percurso em profundidade e de percurso em largura.

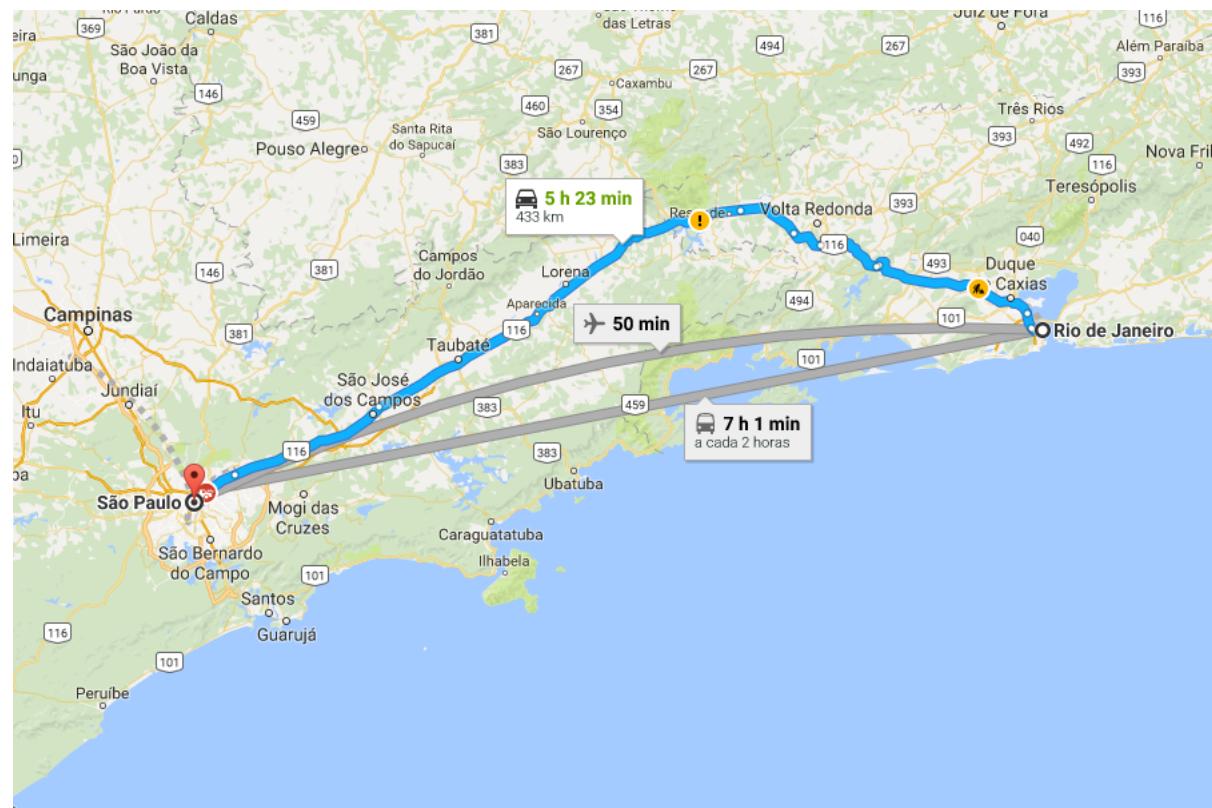


# Caminhos mínimos

# Caminhos mínimos

Considere um mapa rodoviário, onde a distância entre cada par de cidades vizinhas está marcada.

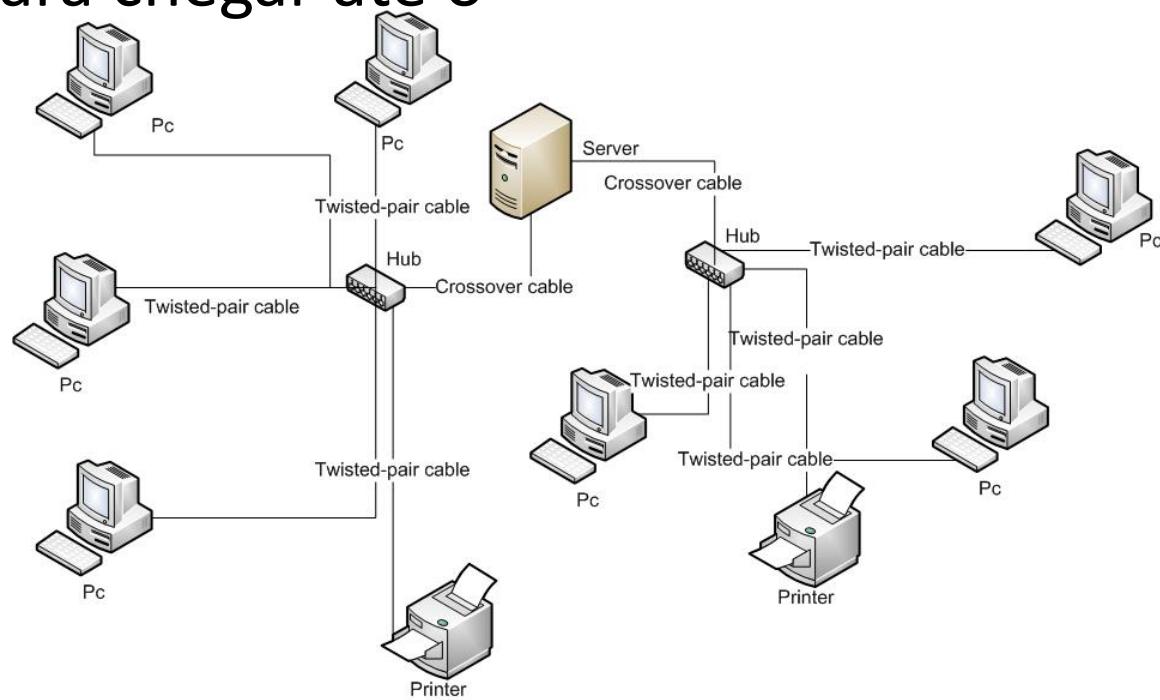
Qual é a rota mais curta para um motorista ir do Rio de Janeiro até São Paulo?



# Caminhos mínimos

Agora considere uma rede de comunicação: o roteador deve definir qual caminho um pacote de dados deve seguir para chegar até o roteador de destino

Uma forma de fazer isso é encontrando o caminho mais curto no grafo representativo da rede



# Caminhos mínimos – formalização

Considere um grafo dirigido ponderado, com pesos  
 $w: E \rightarrow \mathbb{R}$

- O peso do caminho  $p = (v_0, v_1, \dots, v_k)$  é a soma dos pesos de suas arestas constituintes

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

Para um vértice de origem  $v_0$ , queremos encontrar os caminhos  $p_i$  para os demais vértices que **tenham peso mínimo**.

# Algoritmo de Dijkstra

- Resolve o problema de caminhos mínimos para grafos ponderados com pesos de arestas **não-negativos**.
- Monta uma **árvore de caminhos mínimos** da fonte até os demais vértices.
- É um **algoritmo guloso**: a cada iteração, constrói cada caminho mínimo escolhendo o vértice “mais próximo”.

# Algoritmo de Dijkstra

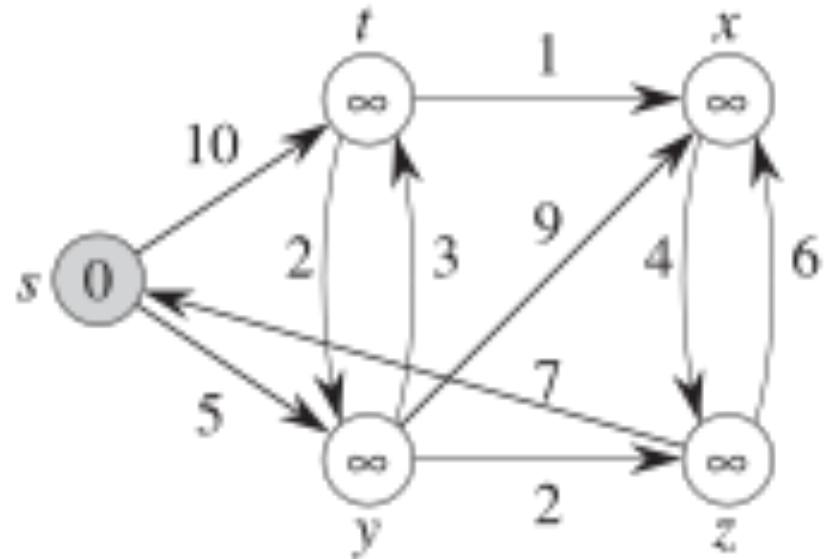
1. Inicialmente, assume-se para os pesos dos caminhos da fonte até todos os demais vértices um valor muito grande, por exemplo infinito
2. A árvore de caminhos  $S$  inicia vazia
3. Escolhe-se um vértice  $u$ , fora de  $S$ , que tenha a menor estimativa de caminho desde a fonte
  - a. Adicione  $u$  a  $S$
  - b. Para os vértices adjacentes a  $u$ , atualize suas respectivas estimativas de caminho
4. Repita 3 até que não haja mais vértices fora de  $S$

# Algoritmo de Dijkstra – Pseudocódigo

```
algoritmoDijkstra (grafo digrafo, vértice first)
    for todo vértice v
        currDist(v) =  $\infty$ ;
    currDist(first) = 0;
    toBeChecked = todos os vértices;
    while toBeChecked não é vazio
        v = um vértice em toBeChecked com currDist(v) mínimo;
        remova v de toBeChecked;
        for todo vértice u adjacente a v e em toBeChecked
            if currDist(u) > currDist(v) + peso(aresta(vu))
                currDist(u) = currDist(v) + peso(aresta(vu));
                antecessor(u) = v;
```

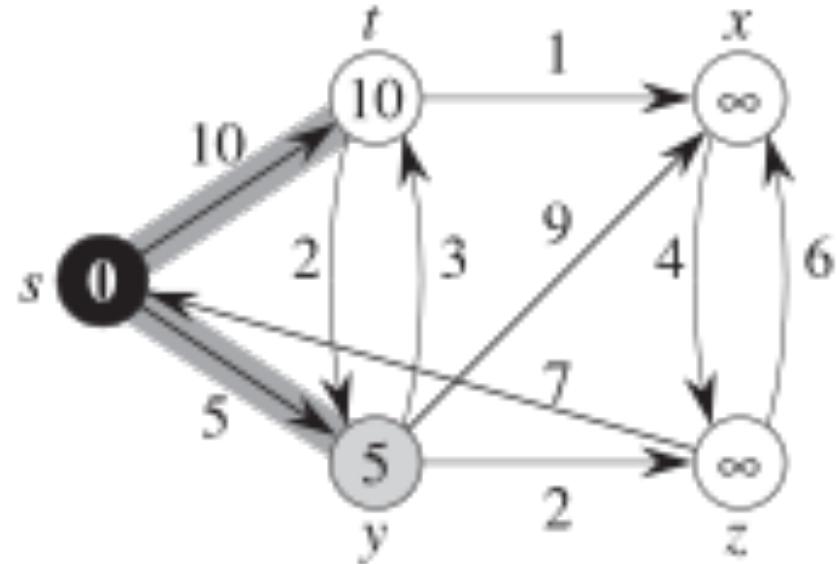
# Algoritmo de Dijkstra - Exemplo

- Inicia no vértice fonte,  $s$



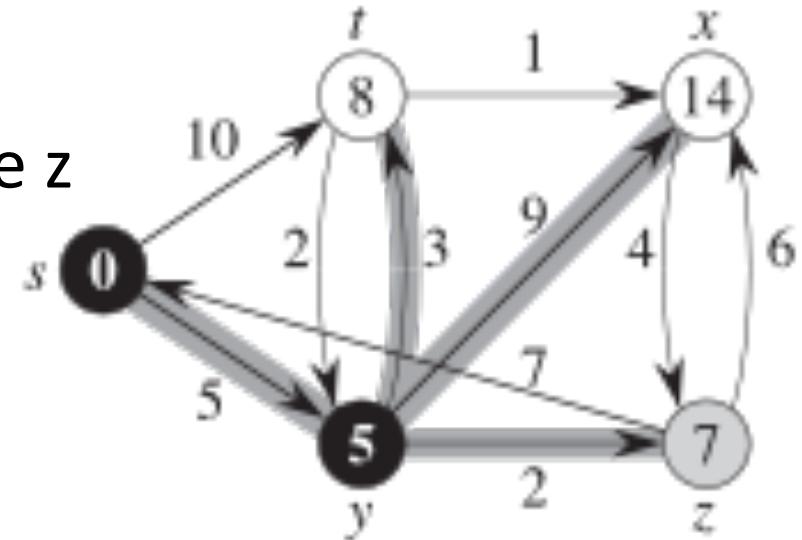
# Algoritmo de Dijkstra - Exemplo

- Atualiza distâncias dos vértices adjacentes – t e y.
- s é antecessor de t e y



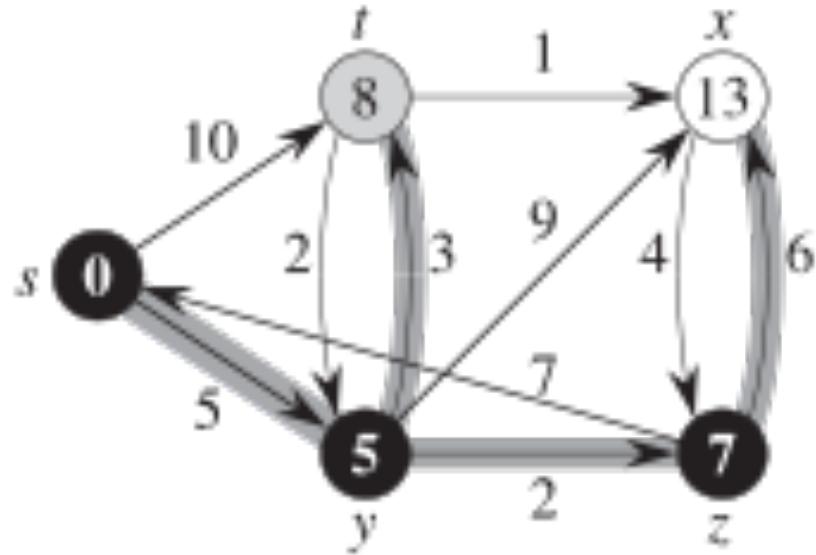
# Algoritmo de Dijkstra - Exemplo

- y é o vértice em análise
- Atualiza distâncias dos vértices adjacentes – t, x e z
- y é antecessor de t, x e z



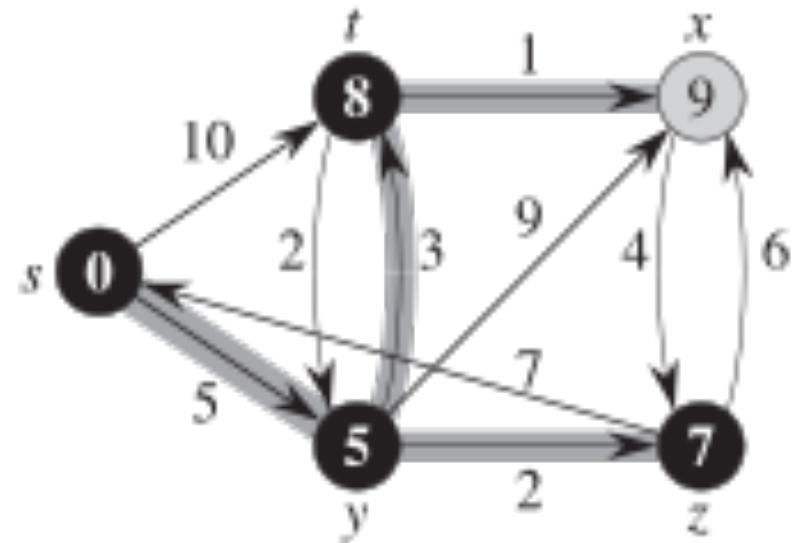
# Algoritmo de Dijkstra - Exemplo

- z é o vértice em análise
- Atualiza distâncias dos vértices adjacentes – x
- z é antecessor de x



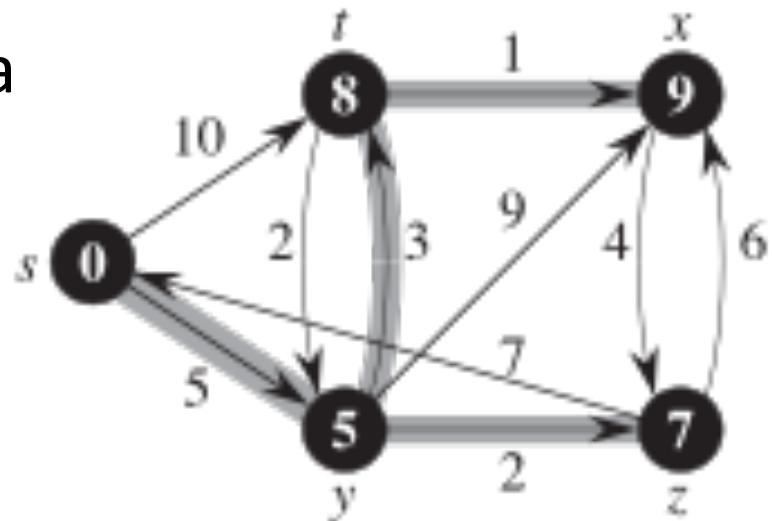
# Algoritmo de Dijkstra - Exemplo

- $t$  é o vértice em análise
- Atualiza distâncias dos vértices adjacentes –  $x$
- $t$  é antecessor de  $x$



# Algoritmo de Dijkstra - Exemplo

- Fim do algoritmo
- Areias hachuradas indicam a árvore de caminhos mínimos a partir do vértice fonte s
  - Registrada através do antecessor de cada vértice



Explore mais exemplos em <https://visualgo.net/sssp>

# Complexidade - Algoritmo de Dijkstra

`algoritmoDijkstra(grafo digrafo, vértice first)`

for todo vértice  $v$  | $V$ | vezes

$currDist(v) = \infty;$

*currDist(first) = 0;*

`toBeChecked = todos os vértices;`

while toBeChecked *não* é vazio

|V| vezes

$v = \text{um vértice em } \text{toBeChecked} \text{ com } \text{currDist}(v) \text{ mínimo:}$

*remova v de toBeChecked;*

for todo vértice  $v$  adjacente a  $v$  e em  $\text{toBeChecked}$

## grau(v) vezes

if  $currDist(u) > currDist(v) + peso(aresta(vu))$

$$currDist(u) = currDist(v) + peso(aresta(vu)) ;$$

*antecessor*(u) = v;

Custo computacional de  $O(|V|^2)$ ; pode ser melhorado para  $O((|V| + |E|) \log |V|)$  usando heap

# Árvore geradora mínima

# Árvore geradora mínima (Minimum Spanning Tree)

Uma companhia aérea conecta 7 cidades com múltiplos vôos

Devido à crise, ela precisa cortar vôos mas que, ainda assim, seja possível chegar a uma cidade a partir de qualquer outra, mesmo que indiretamente

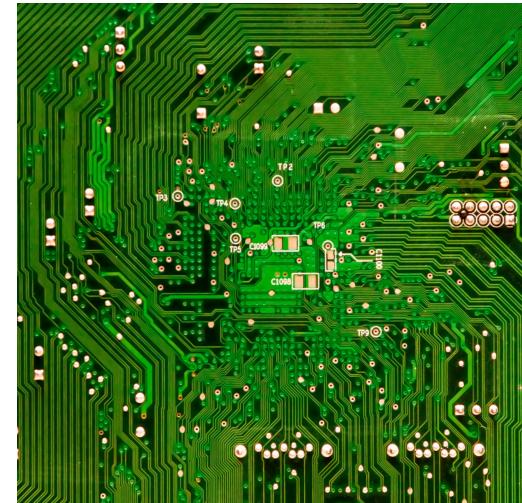
- Qual é o conjunto de vôos de menor custo total que deve ser mantido?



# Árvore geradora mínima

Outro cenário: em um circuito eletrônico precisa-se conectar  $n$  pinos com um arranjo de  $n - 1$  fios, cada um interligando dois pinos de cada vez.

- Entre todos os arranjos possíveis, qual usa o menor comprimento total de fios?



# Árvore geradora mínima

- Observe que o método de percurso em profundidade ou de percurso em largura constrói ao final uma árvore geradora, i.e. que alcança todos os vértices
- Porém, eles não consideram os pesos das arestas para definir seus caminhos

# Árvore geradora mínima – formalização

Considere um grafo  $G(V, E)$  **não-dirigido** ponderado, com pesos  $w: E \rightarrow \mathbb{R}$

Queremos encontrar um subconjunto sem ciclos  $T \subseteq E$  que conecte todos os vértices e cujo peso total

$$w(T) = \sum w(u, v)$$

seja **mínimo**.

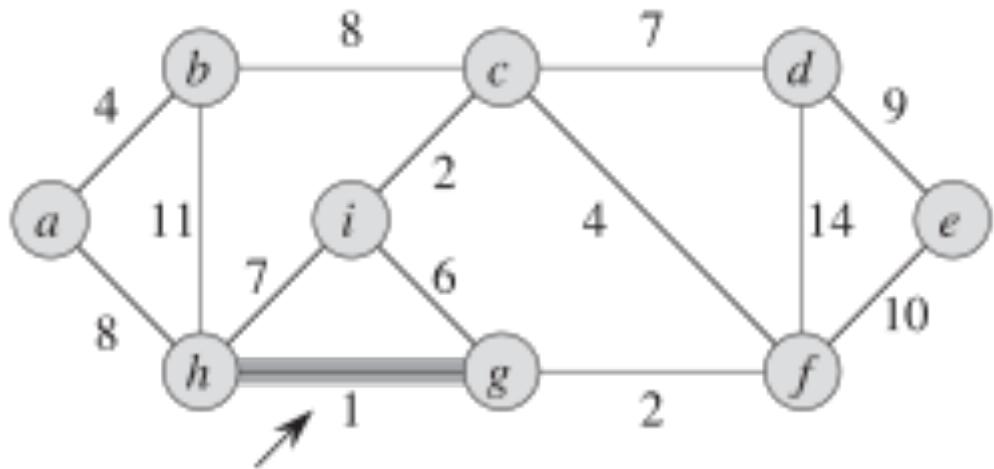
$T$  resulta na **árvore geradora mínima** (*minimum-spanning-tree*) do grafo  $G$

# Árvore geradora mínima – Algoritmo de Kruskal

1. Começamos com uma árvore/floresta vazia  $T$
2. Crie uma lista, em ordem crescente de peso, das arestas do grafo
3. Repita da primeira à última aresta da lista:
  - a. Se a aresta não forma um ciclo com as arestas já em  $T$   
Adicione a aresta a  $T$

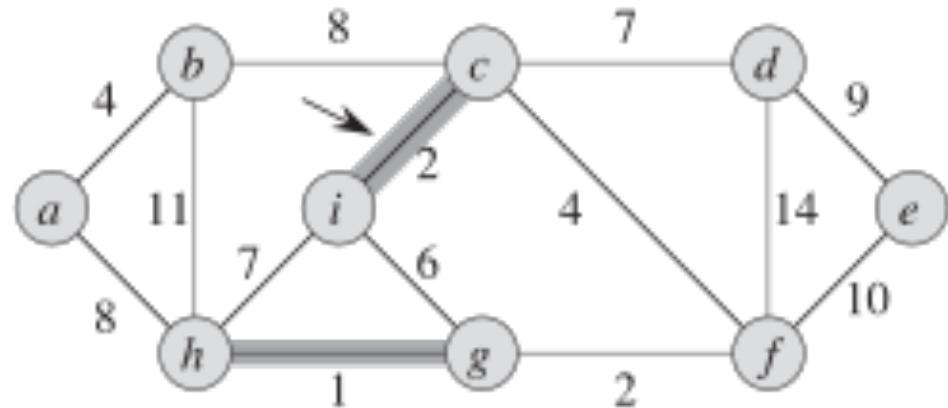
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (h,g)
- Não forma ciclo
- Adicione a T



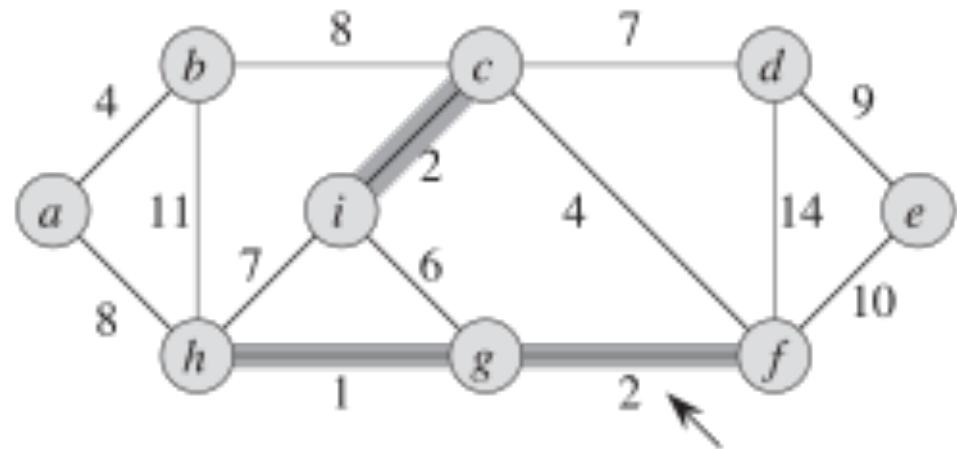
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (i,c)
- Não forma ciclo
- Adicione a T



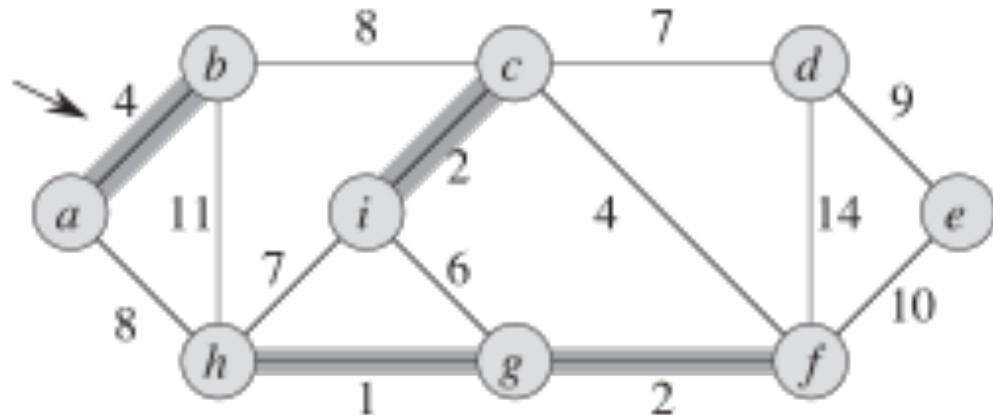
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (g,f)
- Não forma ciclo
- Adicione a T



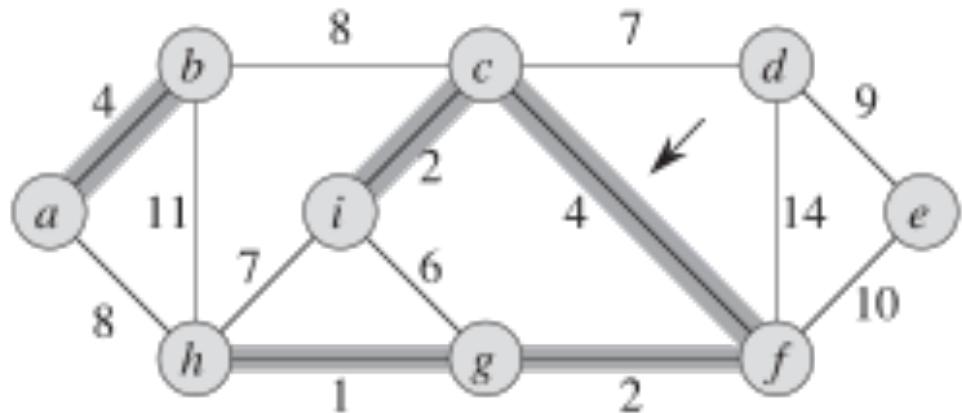
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (a,b)
- Não forma ciclo
- Adicione a T



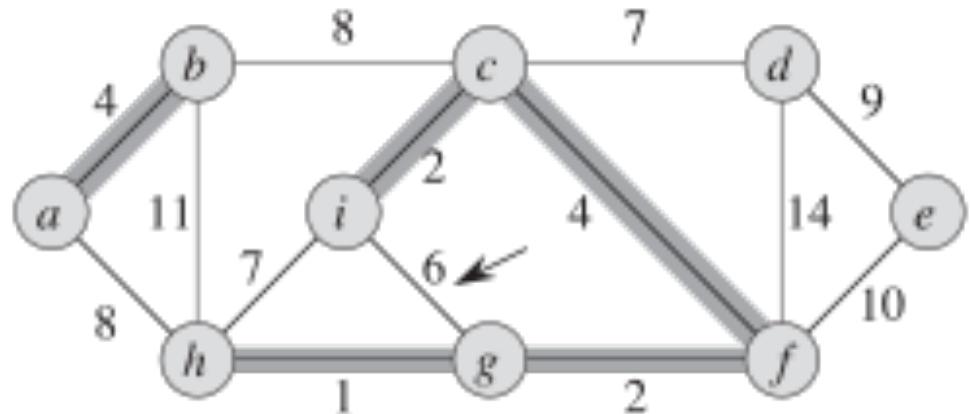
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (c,f)
- Não forma ciclo
- Adicione a T



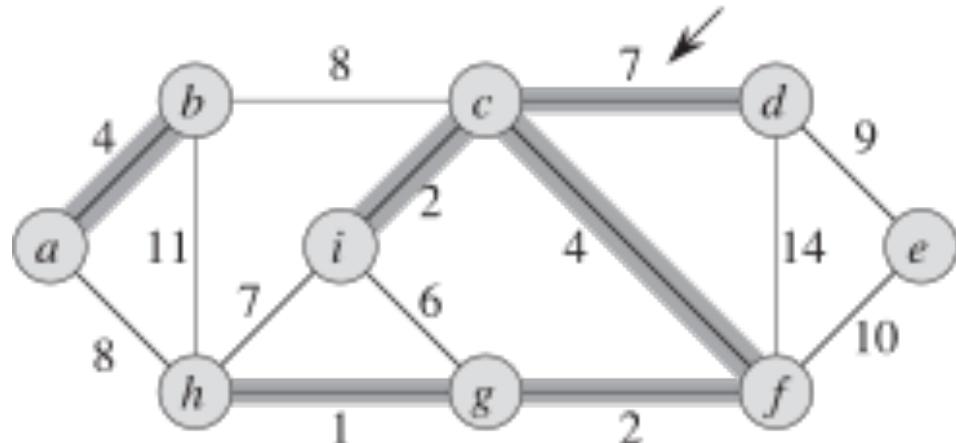
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (i,g)
- Forma ciclo
- Não adicione a T



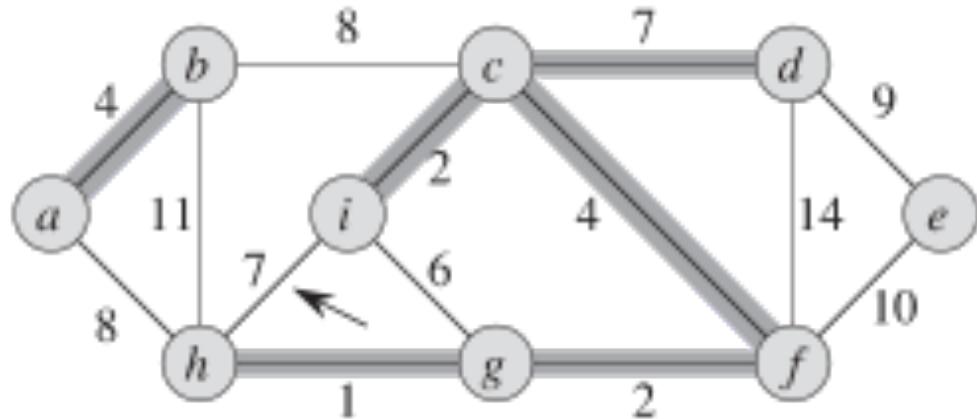
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (c,d)
- Não forma ciclo
- Adicione a T



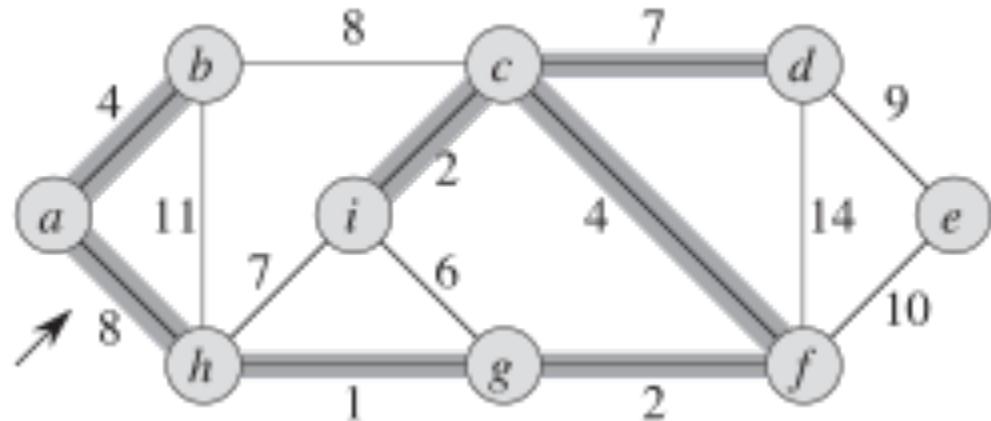
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (h,i)
- Forma ciclo
- Não adicione a T



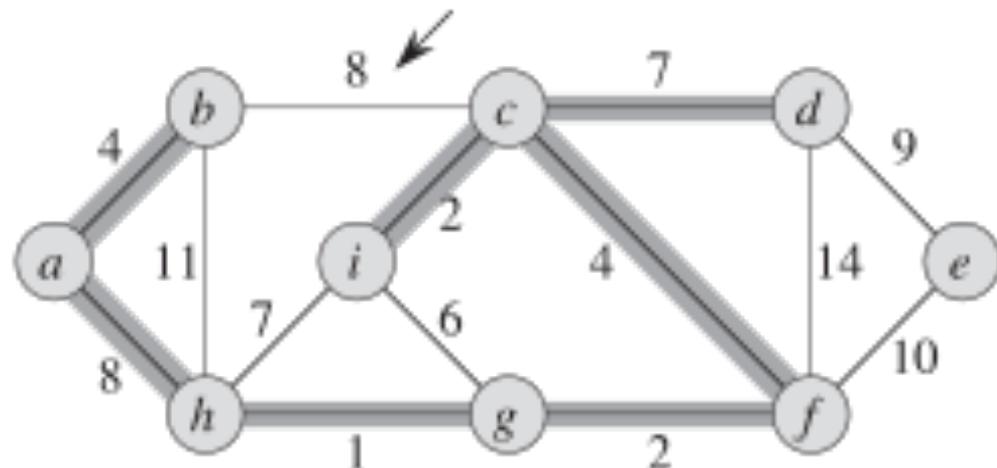
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (a,h)
- Não forma ciclo
- Adicione a T



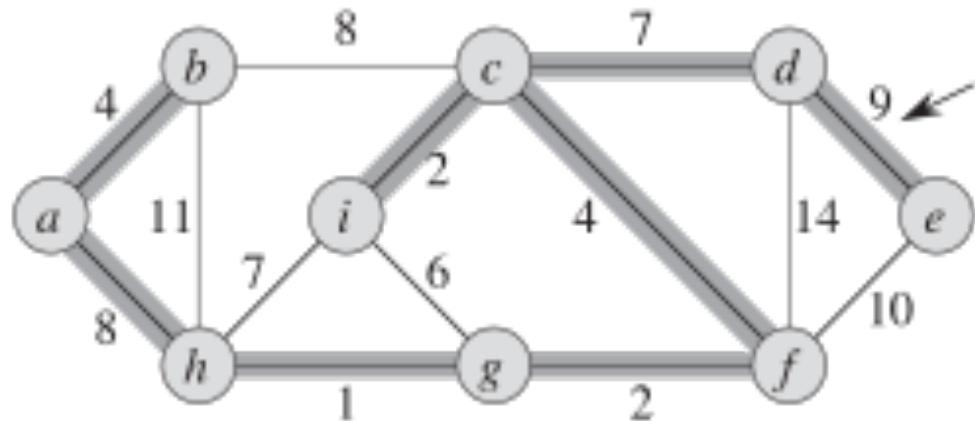
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (b,c)
- Forma ciclo
- Não adicione a T



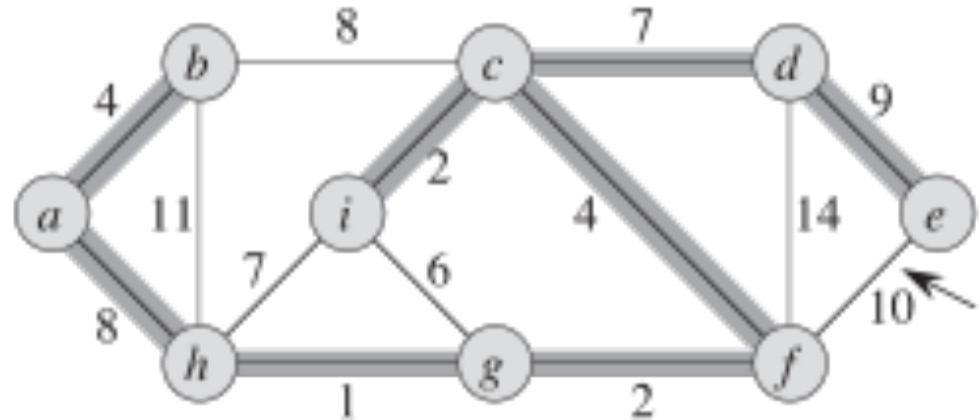
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (d,e)
- Não forma ciclo
- Adicione a T



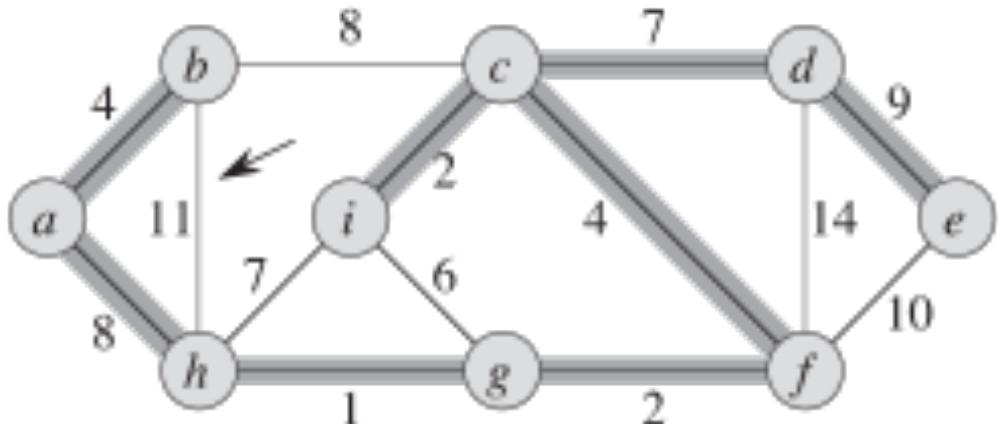
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (e,f)
- Forma ciclo
- Não adicione a T



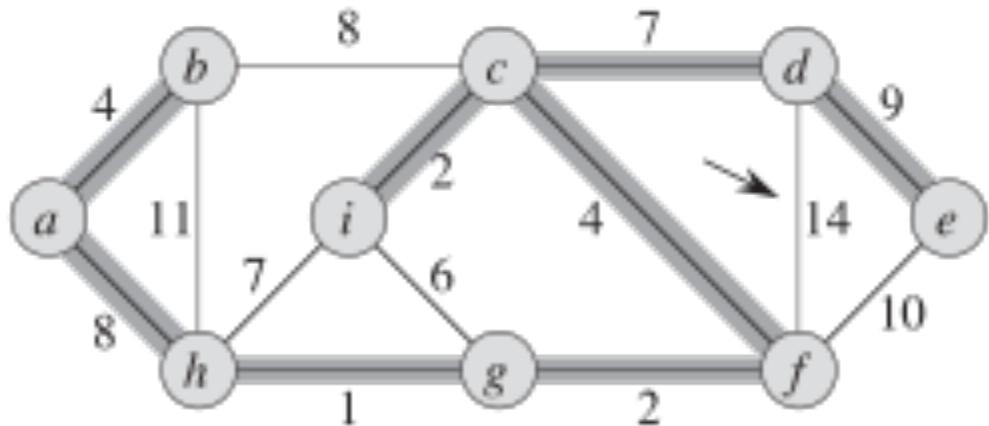
# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (b,h)
- Forma ciclo
- Não adicione a T



# Algoritmo de Kruskal - Exemplo

- Próxima menor aresta: (d,f)
- Forma ciclo
- Não adicione a T



Fim do algoritmo

# Árvore geradora mínima – Algoritmo de Kruskal

- Repare que é um algoritmo guloso, i.e. realiza a escolha de operação, numa dada iteração, que seja a melhor para aquele momento específico
- O ponto-chave do método é definir como detectar ciclos em um grafo

# Detecção de ciclos

- Uma forma de detectar se a adição de uma aresta  $(v, u)$  forma um ciclo em uma árvore é checar se os vértices  $v$  e  $u$  possuem a **mesma raiz de árvore**
- Podemos implementar uma operação que faz a detecção e, caso não haja ciclo, já acrescente a aresta à árvore

# Algoritmo de Kruskal – Pseudocódigo

algoritmoKruskal(*grafo não-dirigido* grafo)

árvore = *null*;

arestas = *sequência de todas as arestas ordenadas por peso*;

for *cada aresta* (*u, v*) *em* arestas *na ordem definida*

if *raiz(u) ≠ raiz(v)*

*adicone a aresta* (*u, v*) *a árvore*;

*retorne* árvore;

# Complexidade - Algoritmo de Kruskal

- Depende da
  - Implementação das operações de detecção da raiz de um vértice e inclusão da aresta, a qual equivale à operação de união de conjuntos
  - Ordenação dos pesos das arestas
- Chega-se à conclusão de que o algoritmo tem custo computacional  $O(|E| \log |V|)$

# Exercício de Assimilação de Conceitos

1. Estude, dentro da biblioteca grafo.h e grafo.cpp, a implementação do algoritmo de Dijkstra.
2. Crie um programa que
  - A. cria um grafo em memória a partir de uma lista de vértices e arestas de um arquivo texto (dica: use a biblioteca fstream),
  - B. calcula o caminho mínimo de um vértice de origem a um vértice de destino, informados pelo usuário a partir do teclado.

# Exercício de Assimilação de Conceitos

- Exemplo de arquivo texto de entrada:

```
v A
v B
v C
v D
v E
v F
e A B 40
e A C 20
e B C 10
e B D 50
e C D 80
e C E 100
e D E 20
e D F 60
e E F 20
```

