Análise e Síntese de Algoritmos

MSTs. Prim.
Conjuntos Disjuntos. Kruskal.

CLRS Cap. 21 e 23

Instituto Superior Técnico

2022/2023

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Árvores abrangentes [CLRS, Cap.23]
  - Fluxos máximos [CLRS, Cap.26]
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Complexidade Computacional

Resumo

Motivação

Problema

Suponha que pretende instalar uma nova rede de fornecimento de para um serviço (TV por cabo, gás natural, ...) numa urbanização.

Para estabelecer a rede é necessário fazer obras na via pública para instalar a infraestrutura (colocação de cabos de fibra óptica ou novas condutas de gás).

Motivação

Problema
Suponha que pretende instalar uma nova rede de fornecimento de para um serviço (TV por cabo, gás natural, ...) numa urbanização.

Para estabelecer a rede é necessário fazer obras na via pública para instalar a infraestrutura (colocação de cabos de fibra óptica ou novas condutas de gás).

Objectivo
O objectivo é fornecer o serviço a todas as casas da urbanização através de uma rede. No entanto, cada possível ligação na urbanização tem um custo e pretende-se minimizar o custo total da instalação.

Motivação

Solução

- Cada casa da urbanização é modelada como um vértice num grafo
- Cada possível ligação entre casas corresponde a um arco pesado cujo peso indica o custo da ligação
- A solução do problema corresponde à árvore abrangente de menor custo – *Minimum Spanning Tree (MST)* – do grafo

Árvores Abrangentes de Menor Custo

Árvores Abrangentes

- Um grafo não dirigido $G = (V, E)$, diz-se ligado se para qualquer par de vértices existe um caminho que liga os dois vértices
- Dado grafo não dirigido $G = (V, E)$, ligado, uma árvore abrangente é um sub-conjunto acíclico $T \subseteq E$, que liga todos os vértices
- O tamanho da árvore é $|T| = |V| - 1$

Árvores Abrangentes de Menor Custo

Árvores Abrangentes de Menor Custo

Dado grafo $G = (V, E)$, ligado, não dirigido, com uma função de pesos $w : E \rightarrow \mathbb{R}$, identificar uma árvore abrangente T, tal que a soma dos pesos dos arcos de T é minimizada

$$\min\ w(T) = \sum_{(u,v)\in T} w(u, v)$$

Algoritmo (greedy) genérico

Abordagem Greedy

- Manter conjunto A que é um sub-conjunto de uma MST T
- A cada passo do algoritmo identificar arco (u, v) que pode ser adicionado a A sem violar a invariante
- $A \cup \{(u, v)\}$ é sub-conjunto de uma MST T
  - (u, v) é declarado um arco seguro para A

Invariante

Antes de cada iteração, A é um sub-conjunto de uma MST T

Algoritmo (greedy) genérico

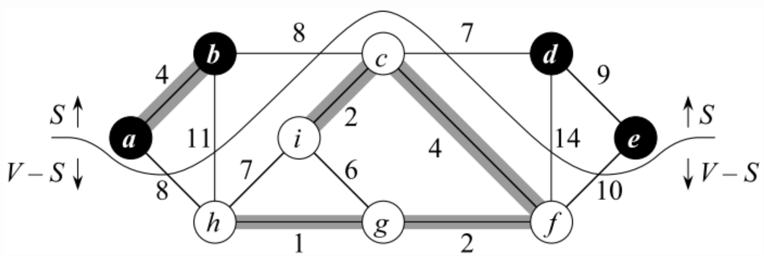
MST-Genérico(G,w)

$A = \emptyset$
while A não forma árvore abrangente **do**
  identificar arco seguro (u, v) para A
  $A = A \cup \{(u, v)\}$
end while
return A

Algoritmo (greedy) genérico

Definições

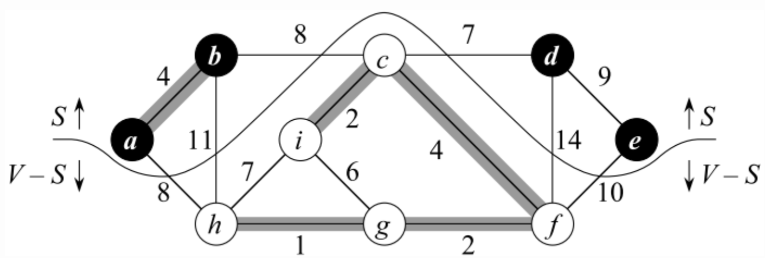
- Um corte $(S, V - S)$ de um grafo não dirigido $G = (V, E)$ é uma partição de V
- Um arco $(u, v) \in E$ cruza o corte $(S, V - S)$ se um dos extremos está em S e o outro está em $V - S$



Algoritmo (greedy) genérico

Definições

- Um corte respeita um conjunto de arcos A se nenhum arco de A cruza o corte
- Um arco diz-se um arco leve que cruza um corte se o seu peso é o menor de todos os arcos que cruzam o corte



Algoritmo (greedy) genérico

Critérios de Otimalidade

- Seja $G = (V, E)$ um grafo não dirigido, ligado, com função de pesos w
- Seja A um sub-conjunto de E incluído numa MST T $(A \subseteq T \subseteq E)$
- Seja $(S, V - S)$ qualquer corte de G que respeita A
- Seja (u, v) um arco leve que cruza $(S, V - S)$

$\Rightarrow$ Então (u, v) é um arco seguro para A

Prova

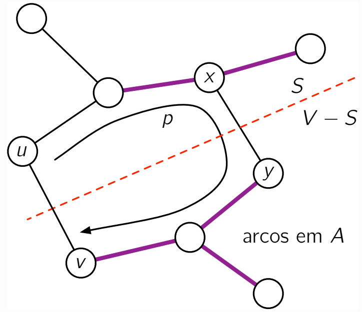
- MST T, com $A \subseteq T$, e arco leve $(u, v) \notin T$
- Objectivo: Construir outra MST T' que inclui $A \cup \{(u, v)\}$
- (u, v) é um arco seguro para A

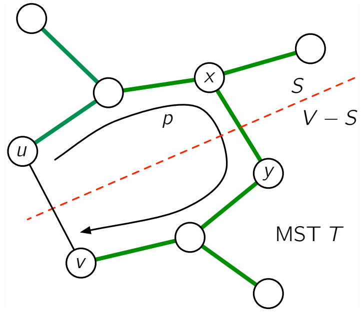
Algoritmo (greedy) genérico

Critérios de Otimalidade (cont.)

Prova

- O arco (u, v) forma ciclo com arcos do caminho p, definido em T, que liga u a v
- Dado u e v estarem nos lados opostos do corte $(S, V - S)$, então existe pelo menos um arco (x, y) do caminho p em T que cruza o corte





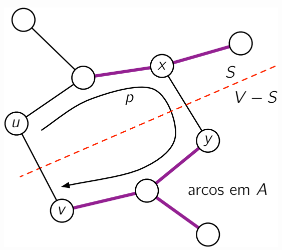
Algoritmo (greedy) genérico

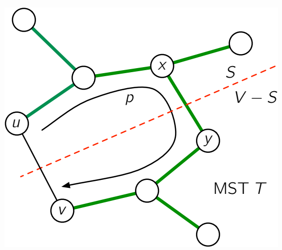
Critérios de Otimalidade (cont.)

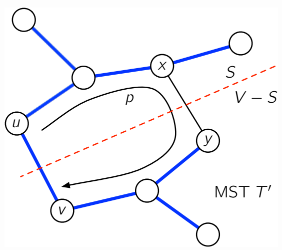
Prova

Arco (x, y)

- $(x, y) \notin A$, porque corte $(S, V - S)$ respeita A
- Remoção de (x, y) divide T em dois componentes
- Inclusão de (u, v) permite formar $T' = T \setminus \{(x, y)\} \cup \{(u, v)\}$
- Dado que (u, v) é um arco leve que cruza o corte $(S, V - S)$, e porque (x, y) também cruza o corte: $w(u, v) \leq w(x, y)$







Algoritmo (greedy) genérico

Critérios de Otimalidade (cont.)

Prova

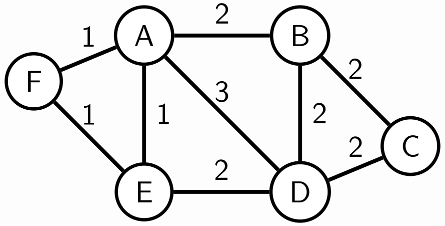
- $w(T') = w(T) - w(x, y) + w(u, v)$
- $w(T') \leq w(T)$ porque $w(u, v) \leq w(x, y)$
- Mas T é MST, pelo que $w(T) \leq w(T')$, por definição de MST
- Logo, $w(T') = w(T)$, e T' também é MST

(u, v) é seguro para A:

- Verifica-se $A \subseteq T'$, dado que por construção $A \subseteq T$, e $(x, y) \notin A$
- Assim, verifica-se também $A \cup (u, v) \subseteq T'$
- T' é MST, pelo que (u, v) é seguro para A

Algoritmo (greedy) genérico

Exercício: Quantas MST's existem?



Algoritmo de Prim

Algoritmo de Prim

- Algoritmo greedy
- MST construída a partir de um vértice raíz r
- Algoritmo mantém sempre uma árvore A $(A \subseteq T)$
- Árvore A é extendida a partir do vértice r
- A cada passo é escolhido um arco leve, seguro para A
- Utilização de fila de prioridade Q

Notação

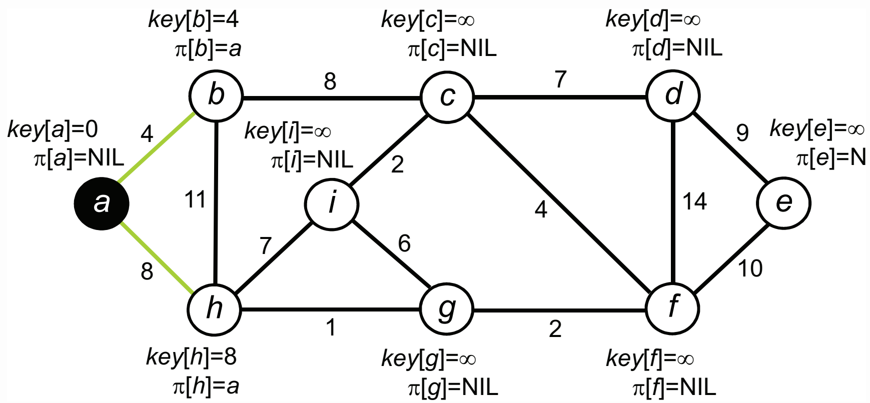
- $key[v]$: menor peso de qualquer arco que ligue v a um vértice na árvore
- $\pi[v]$: antecessor de v na árvore

Algoritmo de Prim

MST-Prim(G,w,r)

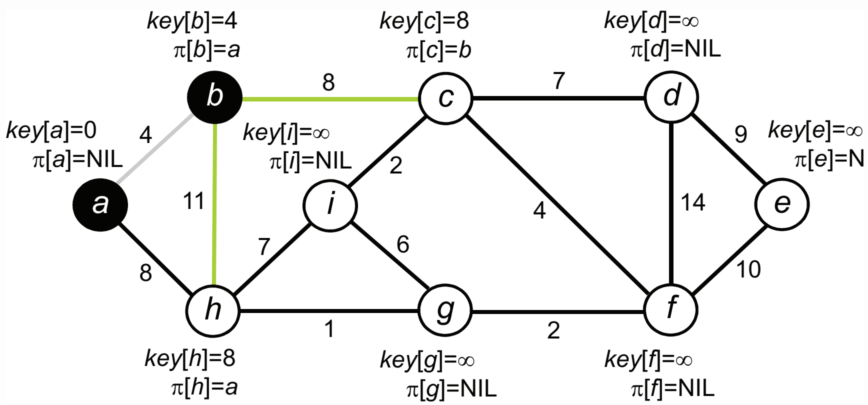
for each $u \in G.V$ **do**
  $key[u] = \infty$ {Inicialização}
  $\pi[u] = NIL$
end for
$key[r] = 0$
$Q = G.V$ {Fila de prioridade}
while $Q \neq \emptyset$ **do**
  $u =$ Extract-Min(Q)
  for each $v \in Adj[u]$ **do**
    if $v \in Q$ **and** $w(u,v) < key[v]$ **then**
      $\pi[v] = u$
      $key[v] = w(u,v)$ {Actualização de Q}
    end if
  end for
end while

Algoritmo de Prim



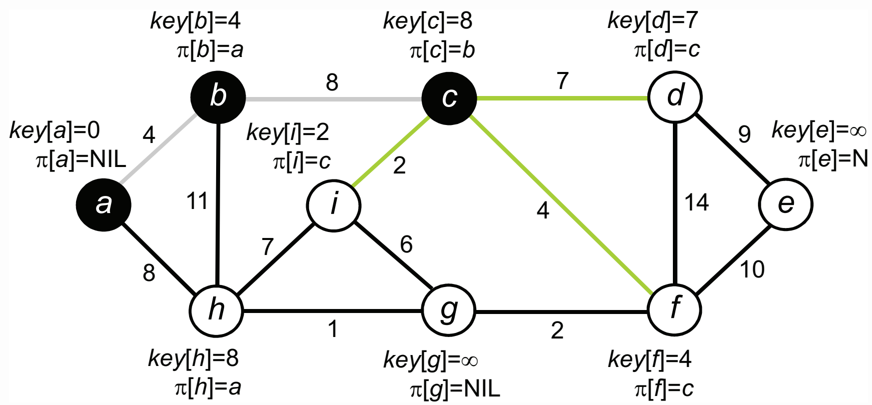
$Q : b, h, c, d, e, f, g, i$

Algoritmo de Prim



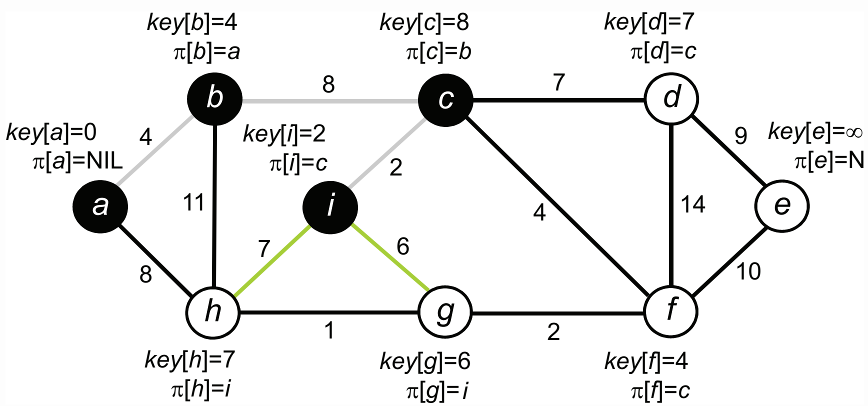
$Q : c, h, d, e, f, g, i$

Algoritmo de Prim



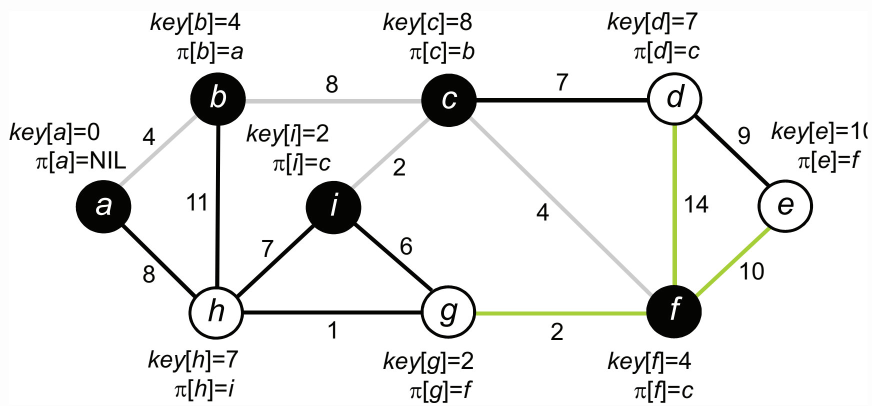
$Q : i, f, d, h, e, g$

Algoritmo de Prim



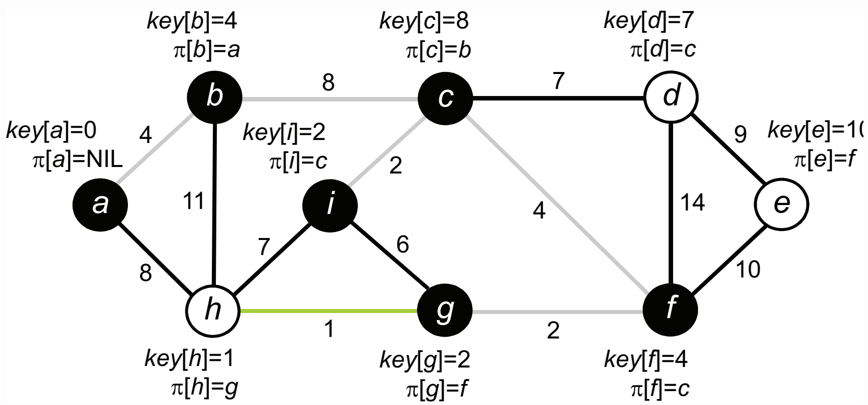
$Q : f, g, d, h, e$

Algoritmo de Prim



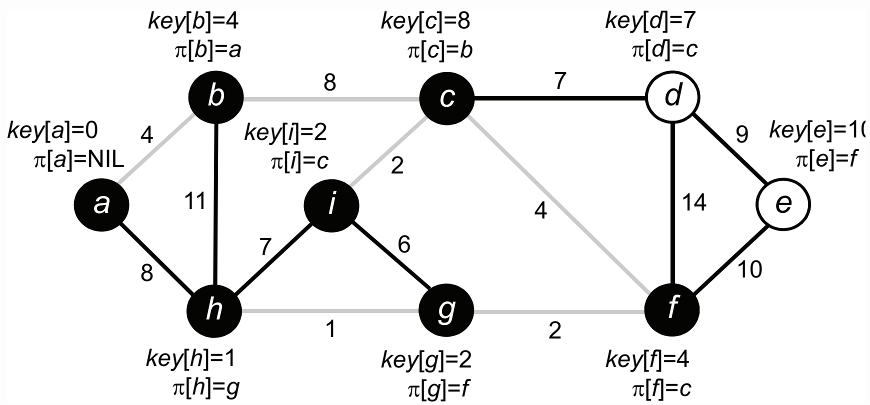
$Q : g, d, h, e$

Algoritmo de Prim



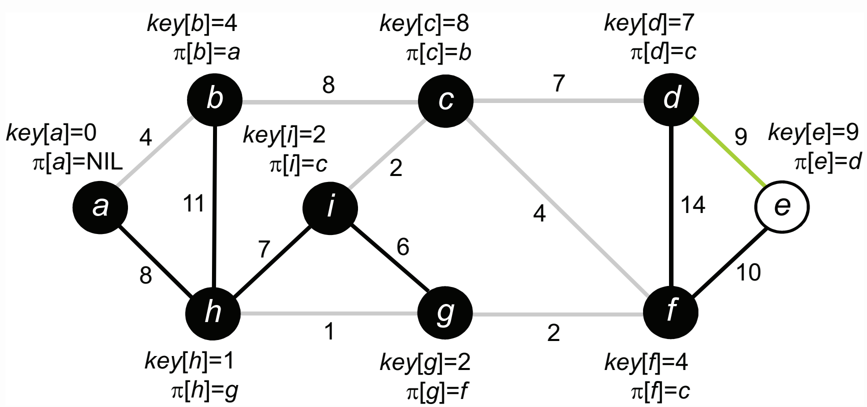
$Q : h, d, e$

Algoritmo de Prim



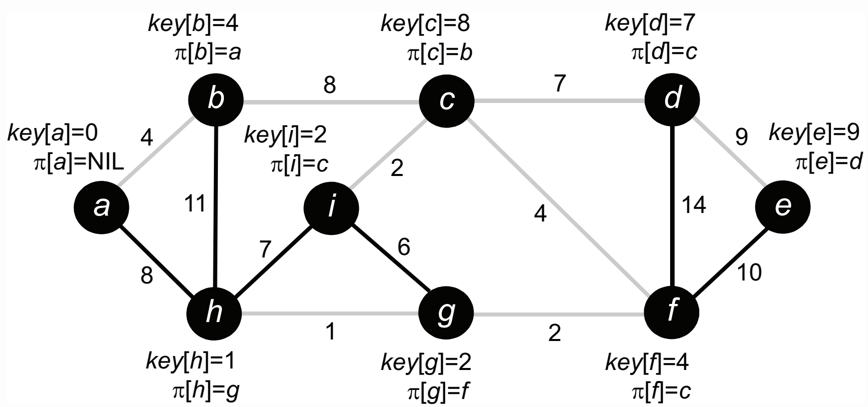
$Q : d, e$

Algoritmo de Prim



$Q : e$

Algoritmo de Prim



$Q : \emptyset$

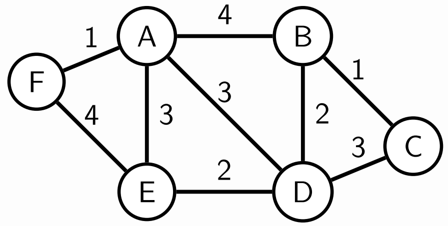
Algoritmo de Prim

Complexidade

- Fila de prioridade baseada em amontoados (heap)
- Extração de vértice da fila Q, implica actualização de Q
  - Cada vértice é extraído apenas 1 vez: $\Theta(V)$
  - Actualização de Q: $O(\lg V)$
  - Logo: $O(V \lg V)$
- Para cada arco (i.e. $\Theta(E)$) existe no pior-caso uma actualização de Q em $O(\lg V)$
- Complexidade algoritmo Prim: $O(V \lg V + E\, lg\, V)$
- Logo, temos $O(E \ \lg \ V)$ porque grafo é ligado
  - Grafo ligado: $|E| \geq |V| - 1$

Algoritmo de Prim

Exercício: Calcule a MST usando o algoritmo de Prim



Conjuntos Disjuntos

Definições

Conjuntos $\{S_1, ..., S_n\}$ sem elementos em comum, ou seja, a interseção entre quaisquer dois conjuntos é o conjunto vazio $S_i \cap S_j = \emptyset, i \neq j$.

- Cada conjunto caracterizado por um representante, que é um elemento do conjunto

Estrutura de Dados para Conjuntos Disjuntos

Permite manter uma colecção de conjuntos disjuntos dinâmicos

- Consulta à estrutura de dados <u>não altera</u> o representante

Conjuntos Disjuntos

Operações

Cada elemento da estrutura é representado por um elemento x

- Make-Set(x)
  - Cria novo conjunto que apenas inclui elemento x (representante)
  - x aponta para o único elemento do conjunto, o representante do conjunto

Conjuntos Disjuntos

Operações

Cada elemento da estrutura é representado por um elemento x

- Make-Set(x)
  - Cria novo conjunto que apenas inclui elemento x (representante)
  - x aponta para o único elemento do conjunto, o representante do conjunto
- Union(x, y)
  - Realiza a união dos conjuntos que contém x e y, respectivamente S_x e S_y
    - Novo conjunto criado: $S_x \cup S_y$
    - S_x e S_y eliminados (conjuntos disjuntos)
    - Novo representante será o representante de S_x ou S_y

Conjuntos Disjuntos

Operações

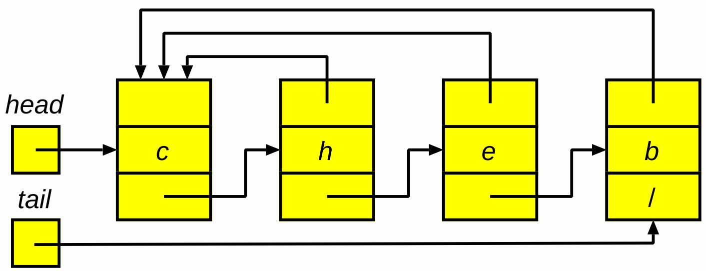
Cada elemento da estrutura é representado por um elemento x

- Make-Set(x)
  - Cria novo conjunto que apenas inclui elemento x (representante)
  - x aponta para o único elemento do conjunto, o representante do conjunto
- Union(x, y)
  - Realiza a união dos conjuntos que contém x e y, respectivamente S_x e S_y
    - Novo conjunto criado: $S_x \cup S_y$
    - S_x e S_y eliminados (conjuntos disjuntos)
    - Novo representante será o representante de S_x ou S_y
- Find-Set(x)
  - Retorna apontador para representante do conjunto que contém x

Utilização de Lista Ligada

Organização

- Elementos de cada conjunto em lista (simplesmente) ligada
- Primeiro elemento é o representante do conjunto
- Todos os elementos incluem apontador para o representante do conjunto



Utilização de Lista Ligada

Tempos de Execução

- Make-Set(x)
  - Criar nova lista com elemento x:

Utilização de Lista Ligada

Tempos de Execução

- Make-Set(x)
  - Criar nova lista com elemento x: $O(1)$
- Find-Set(x)
  - Devolver ponteiro para representante:

Utilização de Lista Ligada

Tempos de Execução

- Make-Set(x)
  - Criar nova lista com elemento x: $O(1)$
- Find-Set(x)
  - Devolver ponteiro para representante: $O(1)$
- Union(x, y):
  - Colocar elementos de x no fim da lista de y
  - Actualizar ponteiros de elementos de x para representante

Utilização de Lista Ligada

Tempos de Execução

- Make-Set(x)
  - Criar nova lista com elemento x: $O(1)$
- Find-Set(x)
  - Devolver ponteiro para representante: $O(1)$
- Union(x, y):
  - Colocar elementos de x no fim da lista de y
  - Actualizar ponteiros de elementos de x para representante
- Operações sobre n elementos $x_1, x_2, \ldots, x_n$
  - n operações Make-Set(x_i)
    - $\Theta(n)$
  - $n-1$ operações Union(x_{i-1}, x_i), para $i = 2, \ldots, n$
    - Cada operação Union(x_{i-1}, x_i) actualiza $i-1$ elementos
    - Custo das $n-1$ operações: $\sum_{i=1}^{n-1} i = \Theta(n^2)$
  - Número total de operações é $m = 2n - 1$
    Em média, cada operação requer tempo $\Theta(n)$

Utilização de Lista Ligada

Heurística: União Pesada (union by size)

- A cada conjunto associar o número de elementos
- Para cada operação Union, juntar lista com menor número de elementos à lista com maior número de elementos
  - Necessário actualizar menor número de ponteiros para representante
- Custo total de m operações é melhorado

Utilização de Lista Ligada

Heurística: União Pesada (union by size)

- A cada conjunto associar o número de elementos
- Para cada operação Union, juntar lista com menor número de elementos à lista com maior número de elementos
  - Necessário actualizar menor número de ponteiros para representante
- Custo total de m operações é melhorado
- Sequência de m operações de Make-Set, Union e Find-Set (que incluem n operações Union) é: $O(m + n \lg n)$

Utilização de Lista Ligada

Tempos de Execução (com Heurística) (Prova Teorema 21.1 CLRS)

- Sempre que o ponteiro para o representante x é actualizado, x encontra-se no conjunto com menor número de elementos
  - Da 1ª vez, conjunto resultante com pelo menos 2 elementos
  - Da 2ª vez, conjunto resultante com pelo menos 4 elementos
  - ...
  - Após representante de x ter sido actualizado $\lceil \lg k \rceil$ vezes, conjunto resultante tem pelo menos k elementos
- Maior conjunto tem n elementos
  - Cada ponteiro actualizado não mais do que $\lceil \lg n \rceil$ vezes
- Tempo total para actualizar n elementos é $O(n \lg n)$
- Make-Set e Find-Set têm tempos de execução $O(1)$ e há $O(m)$ destas operações
- Tempo total para m operações (com n Union) é $O(m + n \lg n)$

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_1, x_{13})
Union(x_6, x_9)
Find-Set(x_7)
Union(x_3, x_{11})
Find-Set(x_{14})

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_1, x_{13})
Union(x_6, x_9)
Find-Set(x_7)
Union(x_3, x_{11})
Find-Set(x_{14})

$\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},$
$\{9\},\{10\},\{11\},\{12\},\{13\},\{14\},\{15\},\{16\}$

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_1, x_{13})
Union(x_6, x_9)
Find-Set(x_7)
Union(x_3, x_{11})
Find-Set(x_{14})

{1},{2},{3},{4},{5},{6},{7},{8},
{9},{10},{11},{12},{13},{14},{15},{16}

{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_1, x_{13})
Union(x_6, x_9)
Find-Set(x_7)
Union(x_3, x_{11})
Find-Set(x_{14})

{1},{2},{3},{4},{5},{6},{7},{8},
{9},{10},{11},{12},{13},{14},{15},{16}

{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_1, x_{13})
Union(x_6, x_9)
Find-Set(x_7)
Union(x_3, x_{11})
Find-Set(x_{14})

{1},{2},{3},{4},{5},{6},{7},{8},
{9},{10},{11},{12},{13},{14},{15},{16}

{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}

{1,2,3,4, 13,14,15,16}, {5,6,7,8}, {9,10,11,12}

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_1, x_{13})
Union(x_6, x_9)
Find-Set(x_7)
Union(x_3, x_{11})
Find-Set(x_{14})

{1},{2},{3},{4},{5},{6},{7},{8},
{9},{10},{11},{12},{13},{14},{15},{16}

{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}

{1,2,3,4, 13,14,15,16}, {5,6,7,8}, {9,10,11,12}

{1,2,3,4, 13,14,15,16}, {5,6,7,8, 9,10,11,12}

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
Make-Set(x_i)
end for
	{1},{2},{3},{4},{5},{6},{7},{8},
	{9},{10},{11},{12},{13},{14},{15},{16}

for $i = 1$ **to** 15 **by** 2 **do**
Union(x_i, x_{i+1})
end for
	{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

for $i = 1$ **to** 13 **by** 4 **do**
Union(x_i, x_{i+2})
end for
	{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}

Union(x_1, x_{13})	{1,2,3,4, 13,14,15,16}, {5,6,7,8}, {9,10,11,12}
Union(x_6, x_9)	{1,2,3,4, 13,14,15,16}, {5,6,7,8, 9,10,11,12}
Find-Set(x_7)	5
Union(x_3, x_{11})	
Find-Set(x_{14})	

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
Make-Set(x_i)
end for
	{1},{2},{3},{4},{5},{6},{7},{8},
	{9},{10},{11},{12},{13},{14},{15},{16}

for $i = 1$ **to** 15 **by** 2 **do**
Union(x_i, x_{i+1})
end for
	{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

for $i = 1$ **to** 13 **by** 4 **do**
Union(x_i, x_{i+2})
end for
	{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}

Union(x_1, x_{13})	{1,2,3,4, 13,14,15,16}, {5,6,7,8}, {9,10,11,12}
Union(x_6, x_9)	{1,2,3,4, 13,14,15,16}, {5,6,7,8, 9,10,11,12}
Find-Set(x_7)	5
Union(x_3, x_{11})	{1,2,3,4, 13,14,15,16, 5,6,7,8, 9,10,11,12}
Find-Set(x_{14})	

Utilização de Lista Ligada

Exemplo (usar heurística União Pesada)

for $i = 1$ **to** 16 **do**
Make-Set(x_i)
end for
	{1},{2},{3},{4},{5},{6},{7},{8},
	{9},{10},{11},{12},{13},{14},{15},{16}

for $i = 1$ **to** 15 **by** 2 **do**
Union(x_i, x_{i+1})
end for
	{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}

for $i = 1$ **to** 13 **by** 4 **do**
Union(x_i, x_{i+2})
end for
	{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,16}

Union(x_1, x_{13})	{1,2,3,4, 13,14,15,16}, {5,6,7,8}, {9,10,11,12}
Union(x_6, x_9)	{1,2,3,4, 13,14,15,16}, {5,6,7,8, 9,10,11,12}
Find-Set(x_7)	5
Union(x_3, x_{11})	{1,2,3,4, 13,14,15,16, 5,6,7,8, 9,10,11,12}
Find-Set(x_{14})	1

Utilização de Árvores

Organização

- Cada conjunto representado por uma árvore
- Cada elemento aponta apenas para antecessor na árvore
- Representante da árvore é a raíz
- Antecessor da raíz é a própria raíz

Utilização de Árvores

Organização
- Cada conjunto representado por uma árvore
- Cada elemento aponta apenas para antecessor na árvore
- Representante da árvore é a raíz
- Antecessor da raíz é a própria raíz

Operações
- Find-Set: Percorrer antecessores até raíz ser encontrada $O(n)$
- Union: raíz de uma árvore aponta para raíz da outra árvore $O(1)$

Utilização de Árvores

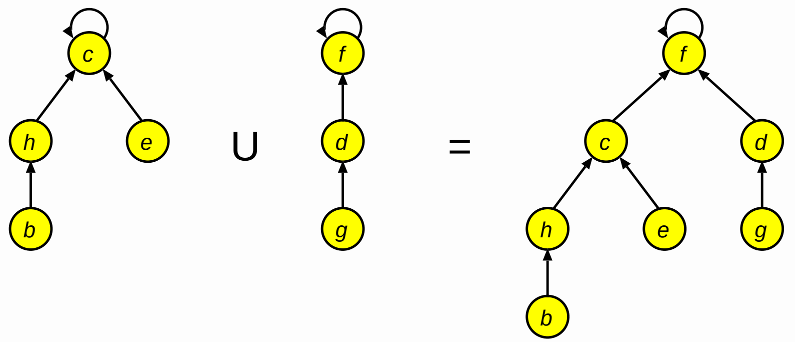
Organização
- Cada conjunto representado por uma árvore
- Cada elemento aponta apenas para antecessor na árvore
- Representante da árvore é a raíz
- Antecessor da raíz é a própria raíz

Operações
- Find-Set: Percorrer antecessores até raíz ser encontrada $O(n)$
- Union: raíz de uma árvore aponta para raíz da outra árvore $O(1)$

Complexidade
- Sequência de $O(m)$ operações é $O(m\,n)$
- Pior caso ocorre quando as árvores que são apenas listas dos n elementos

Utilização de Árvores

Exemplo



Utilização de Árvores

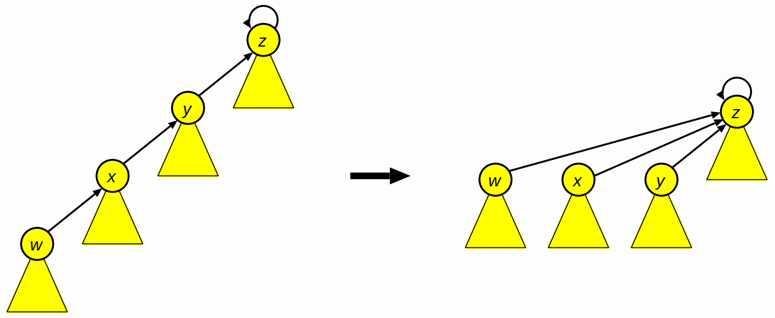
Heurística: União por Categoria (union by rank)

Numa união de dois conjuntos, colocar árvore com menos elementos a apontar para árvore com mais elementos

- Utilizar estimativa da altura de cada sub-árvore
- Categoria (*rank*): aproxima logaritmo do tamanho da sub-árvore e é um limite superior na altura da sub-árvore
- Numa união, raíz da árvore com menor *rank* aponta para raíz da árvore com maior *rank*

Heurística: Compressão de Caminhos (path compression)

Em cada operação Find-Set coloca cada nó visitado a apontar directamente para a raíz da árvore (representante do conjunto)



Make-Set(x)
  $p[x] = x$
  $rank[x] = 0$

Find-Set(x)
  if $x \neq p[x]$ **then**
    $p[x] =$ Find-Set$(p[x])$
  end if
  return $p[x]$

Union(x, y)
  Link(Find-Set(x), Find-Set(y))

Link(x, y)
  if $rank[x] > rank[y]$ **then**
    $p[y] = x$
  else
    $p[x] = y$
    if $rank[x] == rank[y]$ **then**
      $rank[y] = rank[y] + 1$
    end if
  end if

Complexidade

Execução de m operações sobre n elementos: $O(m\ \alpha(n))$

- $\alpha(n) \leq 4$ para todos os efeitos práticos

Prova
Secção 21.4 do livro CLRS

Exemplo (usar heurística União por categoria & compressão caminhos)

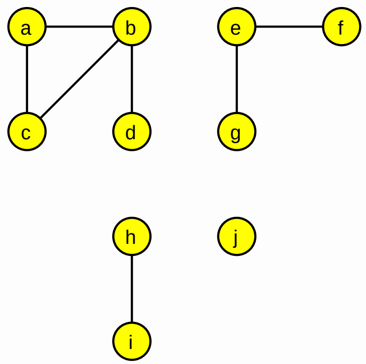
for $i = 1$ **to** 16 **do**
  Make-Set(x_i)
end for
for $i = 1$ **to** 15 **by** 2 **do**
  Union(x_i, x_{i+1})
end for
for $i = 1$ **to** 13 **by** 4 **do**
  Union(x_i, x_{i+2})
end for
Union(x_{11}, x_{13})
Union(x_6, x_{16})
Find-Set(x_{12})

Conjuntos Disjuntos: Aplicações

Problema

Identificar os componentes ligados de grafo não dirigido $G = (V, E)$

Connected-Components(G)
  for each $v \in G.V$ **do**
    Make-Set(v)
  end for
  for each $(u, v) \in G.E$ **do**
    if Find-Set(u) $\neq$ Find-Set(v) **then**
      Union(u, v)
    end if
  end for



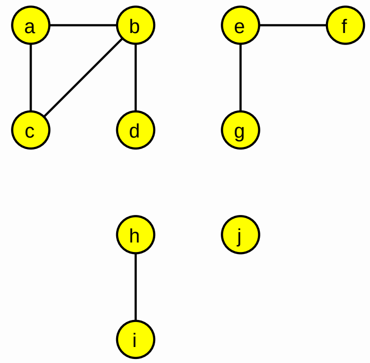
Same-Component(u, v)
  return Find-Set(u) == Find-Set(v)

Conjuntos Disjuntos: Aplicações

Problema

Identificar ciclos num grafo não dirigido $G = (V, E)$

Cycle-detection(G)
  for each $v \in G.V$ **do**
    Make-Set(v)
  end for
  for each $(u, v) \in G.E$ **do**
    if Find-Set(u) $\neq$ Find-Set(v) **then**
      Union(u, v)
    else
      return "Cycle found"
    end if
  end for



Conjuntos Disjuntos: Aplicações

Problema

Considere que está na equipa de projecto de uma rede de distribuição entre um conjunto de cidades. Foram efectuados estudos que calcularam o custo $c(u, v)$ associado a cada ligação possível da nova rede. Pretende-se saber qual o menor custo total de uma rede que interligue todas as cidades

Conjuntos Disjuntos: Aplicações

Problema

Considere que está na equipa de projecto de uma rede de distribuição entre um conjunto de cidades. Foram efectuados estudos que calcularam o custo $c(u, v)$ associado a cada ligação possível da nova rede. Pretende-se saber qual o menor custo total de uma rede que interligue todas as cidades

Solução

- Representar a rede como um grafo não-dirigido e pesado
- Função de pesos é definida pelo custo entre as possíveis ligações
- Rede de menor custo será a Árvore Abrangente de Menor Custo (MST) do grafo

Algoritmo de Kruskal

Organização

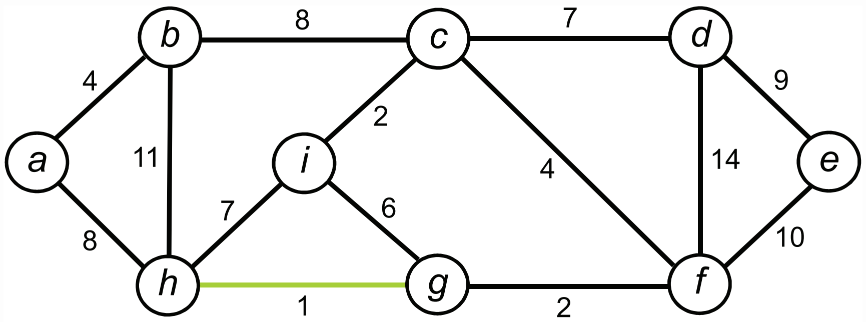
- Algoritmo greedy
- Mantém floresta (de árvores) A
- Utilização de uma estrutura de dados para representar conjuntos disjuntos
- Cada conjunto representa uma sub-árvore de uma MST
- Em cada passo é escolhido um arco leve, seguro para A

Algoritmo de Kruskal

MST-Kruskal(G,w)

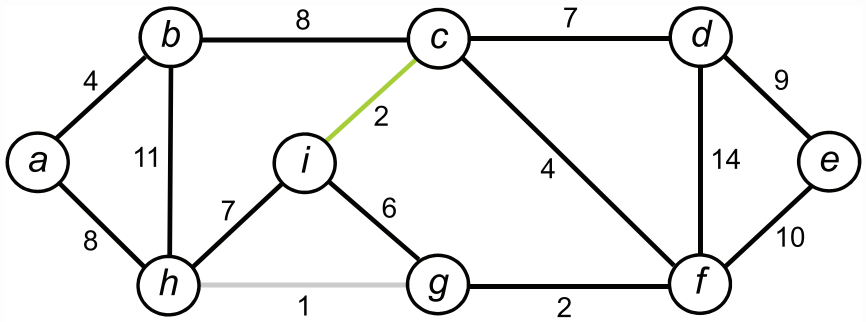
```
A = ∅
for each v ∈ G.V do
   Make-Set(v)                          Cria conjunto para cada v
end for
sortedEdges = sortNonDecreasing(G.E)
for each (u, v) ∈ sortedEdges do
   if Find-Set(u) ≠ Find-Set(v) then
      A = A ∪ {(u, v)}                  (u, v) é arco leve, seguro para A
      Union(u, v)
   end if
end for
return A
```


Algoritmo de Kruskal



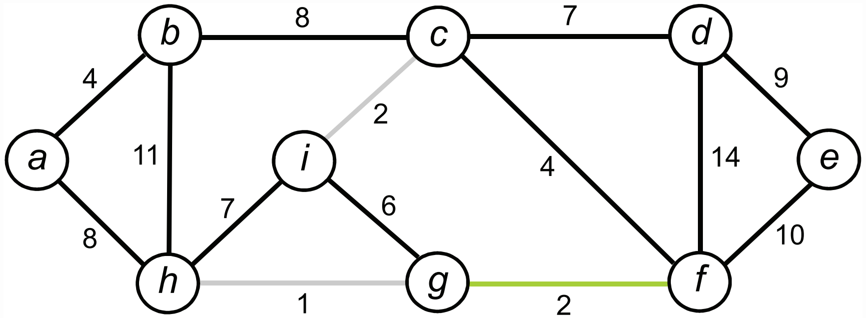
A = { (h,g) }

Algoritmo de Kruskal



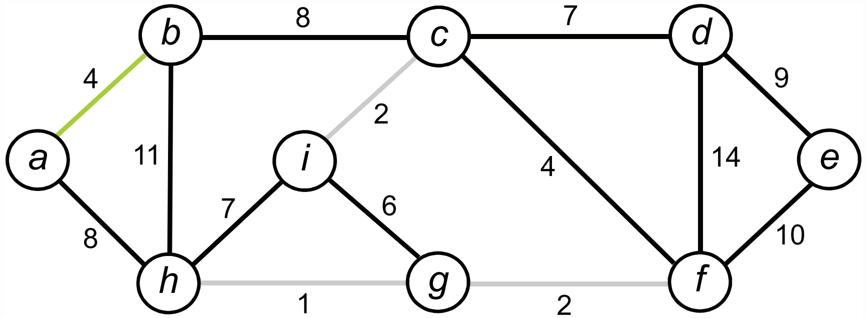
A = { (h,g), (i,c) }

Algoritmo de Kruskal



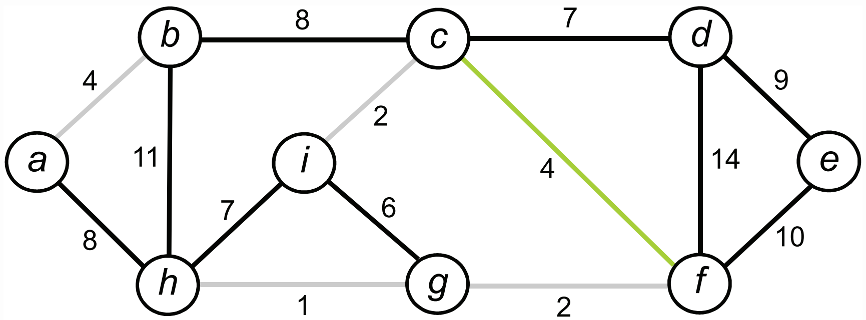
$A = \{ (h,g), (i,c), (g,f) \}$

Algoritmo de Kruskal



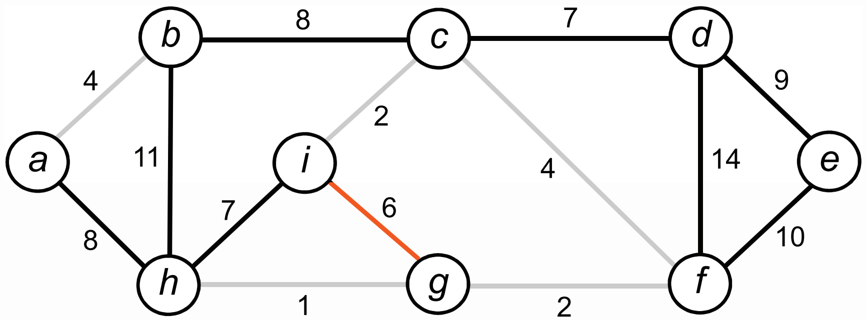
$A = \{ (h,g), (i,c), (g,f), (a,b) \}$

Algoritmo de Kruskal



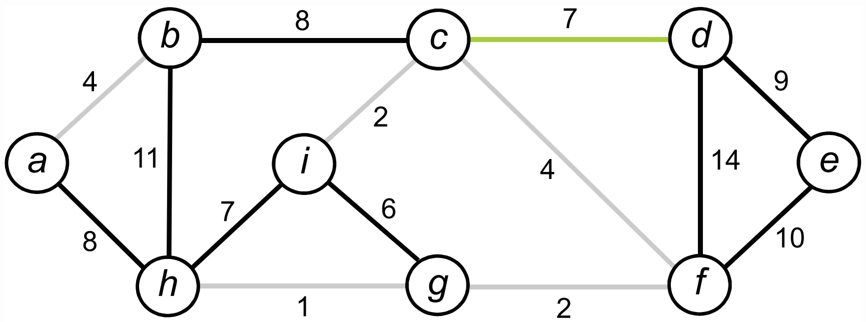
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f) \}$

Algoritmo de Kruskal



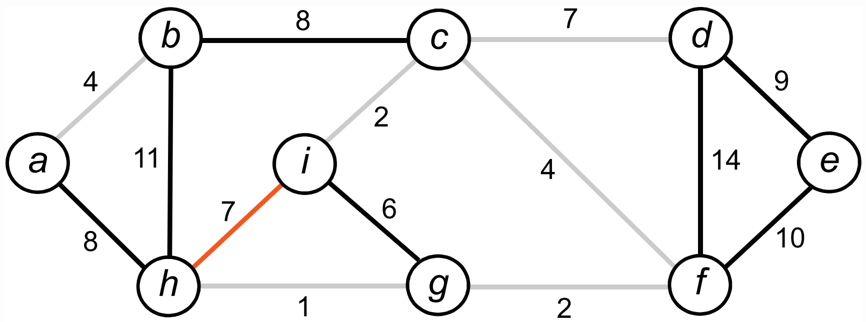
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f) \}$

Algoritmo de Kruskal



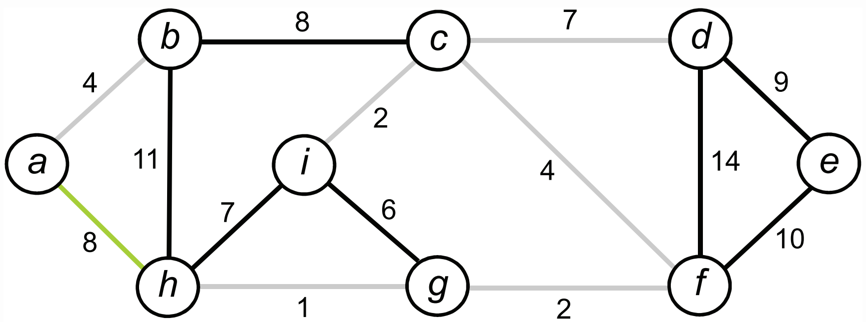
A = { (*h*,*g*), (*i*,*c*), (*g*,*f*), (*a*,*b*), (*c*,*f*), (*c*,*d*) }

Algoritmo de Kruskal



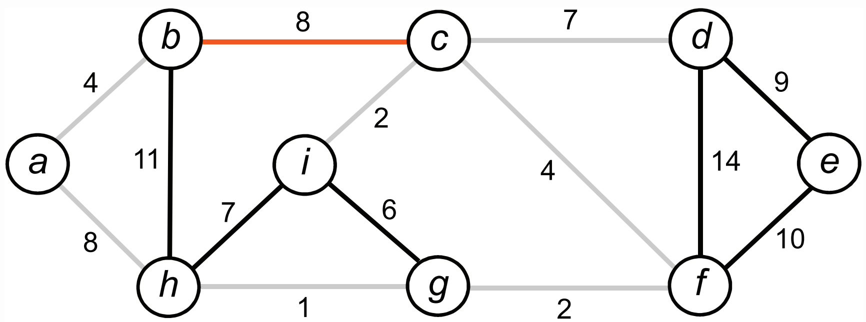
A = { (*h*,*g*), (*i*,*c*), (*g*,*f*), (*a*,*b*), (*c*,*f*), (*c*,*d*) }

Algoritmo de Kruskal



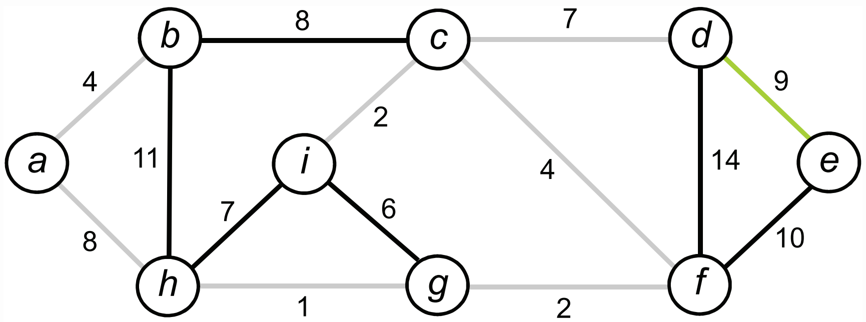
A = { (*h*,*g*), (*i*,*c*), (*g*,*f*), (*a*,*b*), (*c*,*f*), (*c*,*d*), (*a*,*h*) }

Algoritmo de Kruskal



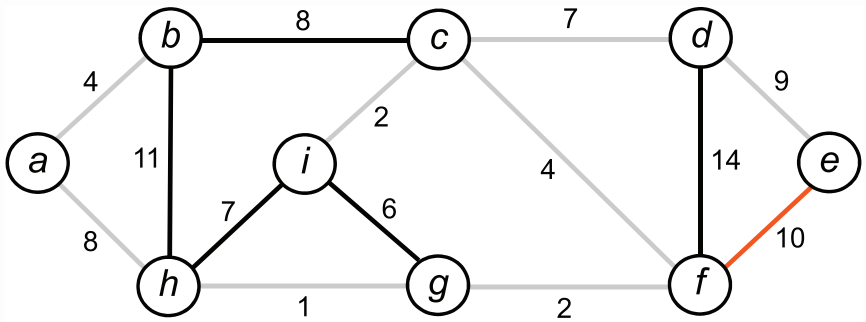
A = { (*h*,*g*), (*i*,*c*), (*g*,*f*), (*a*,*b*), (*c*,*f*), (*c*,*d*), (*a*,*h*) }

Algoritmo de Kruskal



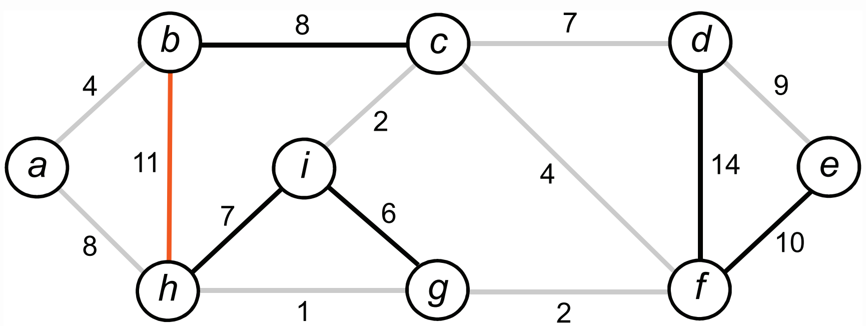
$A = \{\ (h,g),\ (i,c),\ (g,f),\ (a,b),\ (c,f),\ (c,d),\ (a,h),\ (d,e)\ \}$

Algoritmo de Kruskal



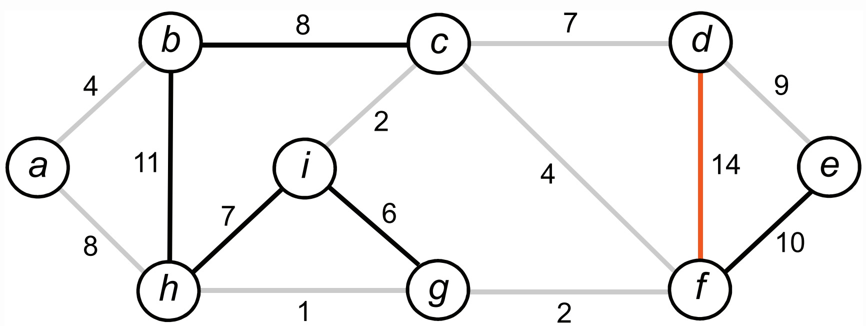
$A = \{\ (h,g),\ (i,c),\ (g,f),\ (a,b),\ (c,f),\ (c,d),\ (a,h),\ (d,e)\ \}$

Algoritmo de Kruskal



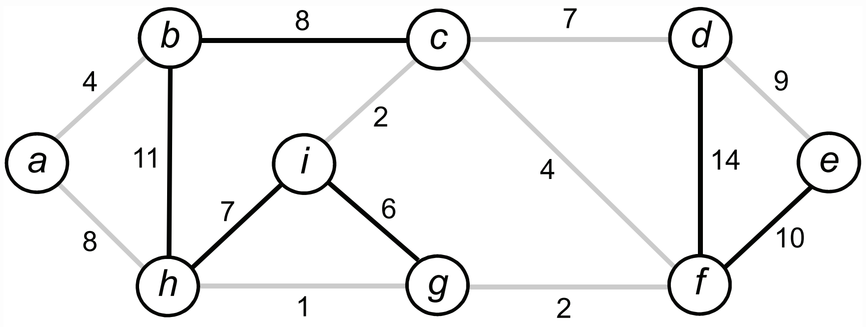
$A = \{\ (h,g),\ (i,c),\ (g,f),\ (a,b),\ (c,f),\ (c,d),\ (a,h),\ (d,e)\ \}$

Algoritmo de Kruskal



$A = \{\ (h,g),\ (i,c),\ (g,f),\ (a,b),\ (c,f),\ (c,d),\ (a,h),\ (d,e)\ \}$

Algoritmo de Kruskal



$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d), (a,h), (d,e) \}$

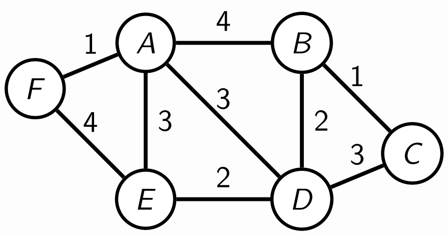
Algoritmo de Kruskal

Complexidade

- Depende da implementação das operações sobre conjuntos disjuntos
- Inicialização: $O(E \lg E)$ devido à ordenação dos arcos
- Operações sobre os conjuntos disjuntos
  - $O(V)$ operações de Make-Set
  - $O(E)$ operações de Find-Set e Union
  - Com estruturas de dados adequadas (árvores com compressão de caminhos e união por categorias) para conjuntos disjuntos é possível estabelecer que $O((V+E)\ \alpha(V))$
  - Como $|E| \geq |V| - 1$ porque o grafo é ligado, então temos $O(E\ \alpha(V))$
- Logo, é possível assegurar $O(E\ lg\ E)$ (maior termo)
  - Dado que $|E| < |V|^2$, obtém-se também $O(E \lg V)$
    - $log|E| < log|V|^2 \Leftrightarrow log|E| < 2log|V|$

Algoritmo de Kruskal

Exercício: Calcule a MST usando o algoritmo de Kruskal



	A	B	C	D	E	F
$rank[v]$						
$\pi[v]$						

Peso das MSTs:	
Número de MSTs:	