ALGORITMOS E ESTRUTURAS DE DADOS II

Grafos – Árvore Geradora Mínima

Prof. Rafael Fernandes Lopes

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

rafaelf@ifma.edu.br

Introdução

- Existem problemas que requerem modelar a interconexão entre diferentes vértices de um grafo como uma árvore, considerando o mínimo custo entre eles
  - Ex 1:
    - No projeto de circuitos eletrônicos, muitas vezes é necessário ligar os pinos de muitos componentes eletronicamente equivalentes, conectando-os com fios. Para interconectar *n* pinos pode-se utilizar um arranjo de n – 1 fios, cada um conectando dois pinos. Para todos os arranjos de fios, a utilização da menor quantidade possível de fios é desejável
  - Ex 2:
    - *Uma companhia telefônica deseja criar uma rede interligando um conjunto de cidades. O custo para unir duas cidades por meio de cabos é conhecido mas, como toda boa companhia, esta deseja minimizar seus gastos, fazendo as ligações mais baratas possíveis sem deixar de cobrir nenhuma das cidades.*

Modelo

- G = (V, E) é um grafo em que V e E correspondem, respectivamente, aos conjuntos de vértices e arestas
  - No exemplo do circuito eletrônico, os pinos podem ser representados por vértices e os fios por arestas

- Para cada aresta $(u,v) \in E$ temos o peso $w(u,v)$ especificando o custo da conexão

Problema

- Os problemas podem ser resolvidos por meio da obtenção da Árvore de Geradora Mínima (ou Árvore de Espalhamento Mínimo, *Minimum Spanning Tree* - MST)

- O problema da MST pode ser descrito da seguinte maneira:
  - Encontre T ⊆ E que conecte todos os vértices, onde G[T] é acíclico, e tal que $\sum_{(u,v)\in T} w(u,v)$ seja minimizado

  - Uma vez que G[T] é um subgrafo acíclico e conecta todos os vértices, G[T] deve ser uma árvore de espalhamento

Exemplo

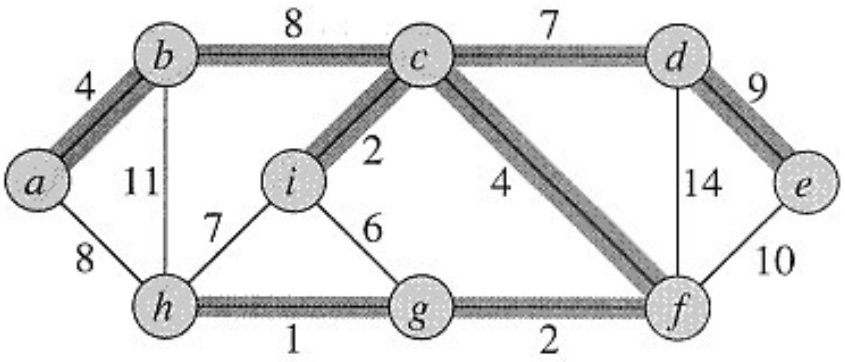


Figure 23.1 A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

Árvore Geradora Mínima

- Assumindo que se queira encontrar uma MST de um grafo G = (V, E), conectado e não direcionado, com uma função peso w : E $\rightarrow \Re$
  - Para tanto, existem dois principais algoritmos para resolver o problema (ambos utilizam estratégias gulosas): os algoritmos de Kruskal e Prim
  - Os algoritmos diferem basicamente em como aplicar as técnicas

- O algoritmo de Kruskal executa em tempo O(|E| lg |E|) usando algoritmo de ordenação merge-sort e a estrutura de dados union-find
- O algoritmo de Prim executa em tempo O(|E| + |V| lg |V|) se for utilizado um heap Fibonacci

Árvore Geradora Mínima

- O que significa que os algoritmos são "gulosos"?
  - A cada passo, uma de muitas possíveis opções deve ser escolhida
  - A estratégia faz a escolha que dá o maior ganho imediato
    - Essa estratégia não garante, em geral, encontrar a solução ótima
    - No entanto, para o problema da MST, a estratégia gulosa produz a solução ótima! (matróide)

Árvore Geradora Mínima

- Os algoritmos gulosos (Kruskal e Prim) seguem uma estrutura genérica, a qual cresce a MST uma aresta por vez

- O algoritmo genérico manipula um subconjunto A que é sempre um subconjunto da MST
  - A cada passo, uma aresta (u, v) é determinada e adicionada a A sem violar o invariante, isto é, $A \cup \{(u, v)\}$ é também um subconjunto da MST

- Uma aresta *e* é dita segura (“safe edge”) se *e* pode ser adicionada a A sem afetar o invariante

Algoritmo Guloso Genérico para cálculo da MST

```
GENERIC-MST(G, w)
1  A ← ∅
2  while A does not form a spanning tree
3       do find an edge (u, v) that is safe for A
4            A ← A ∪ {(u, v)}
5  return A
```

Algoritmo Guloso Genérico para cálculo da MST

- Grande questão relativa ao algoritmo genérico: como encontrar uma aresta segura?

- Observações:
  - À medida que o algoritmo progride, o conjunto A é sempre acíclico
  - Em qualquer estágio do algoritmo, o grafo $G_A = (V, A)$ é uma floresta e cada componente conexo de G_A é uma árvore
    - No início do algoritmo, cada árvore contém apenas um vértice
  - Qualquer aresta segura (u, v) para A conecta apenas componentes distintos de G_A, uma vez que $A \cup \{(u, v)\}$ deve ser acíclico

Árvore Geradora Mínima: Algoritmos

- São descritos na sequência dois algoritmos para a obtenção da MST: Kruskal e Prim
  - Cada um deles utiliza uma regra específica para determinar as arestas seguras da linha 3 do algoritmo genérico
  - Algoritmo de Kruskal:
    - O conjunto A é uma floresta. A aresta segura adicionada a A é sempre a aresta com menos peso no grafo que conecta dois componentes conexos de G_A distintos
  - Algoritmo de Prim:
    - O conjunto A forma uma única árvore. A aresta segura adicionada a A é sempre a aresta com menos peso que conecta a árvore a um vértice que não está na árvore

Algoritmo de Kruskal

- Este algoritmo encontra a aresta segura (e com menos peso) procurando em todas as arestas que conectam quaisquer duas árvores na floresta A
  - A cada passo o algoritmo de Kruskal adiciona à floresta a aresta com menor peso possível
- O algoritmo de Kruskal utiliza conjuntos disjuntos, em cada conjunto contém os vértices de uma árvore na floresta atual
  - A operação FIND-SET(u) retorna um elemento representativo do conjunto que contém u
  - Assim, para determinar se dois vértices u e v pertencem à mesma árvore, basta testar se FIND-SET(u) é igual a FIND-SET(v)
  - A combinação de árvores é executada com o procedimento UNION

Algoritmo de Kruskal

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 UNION(u, v)
9  return A
```

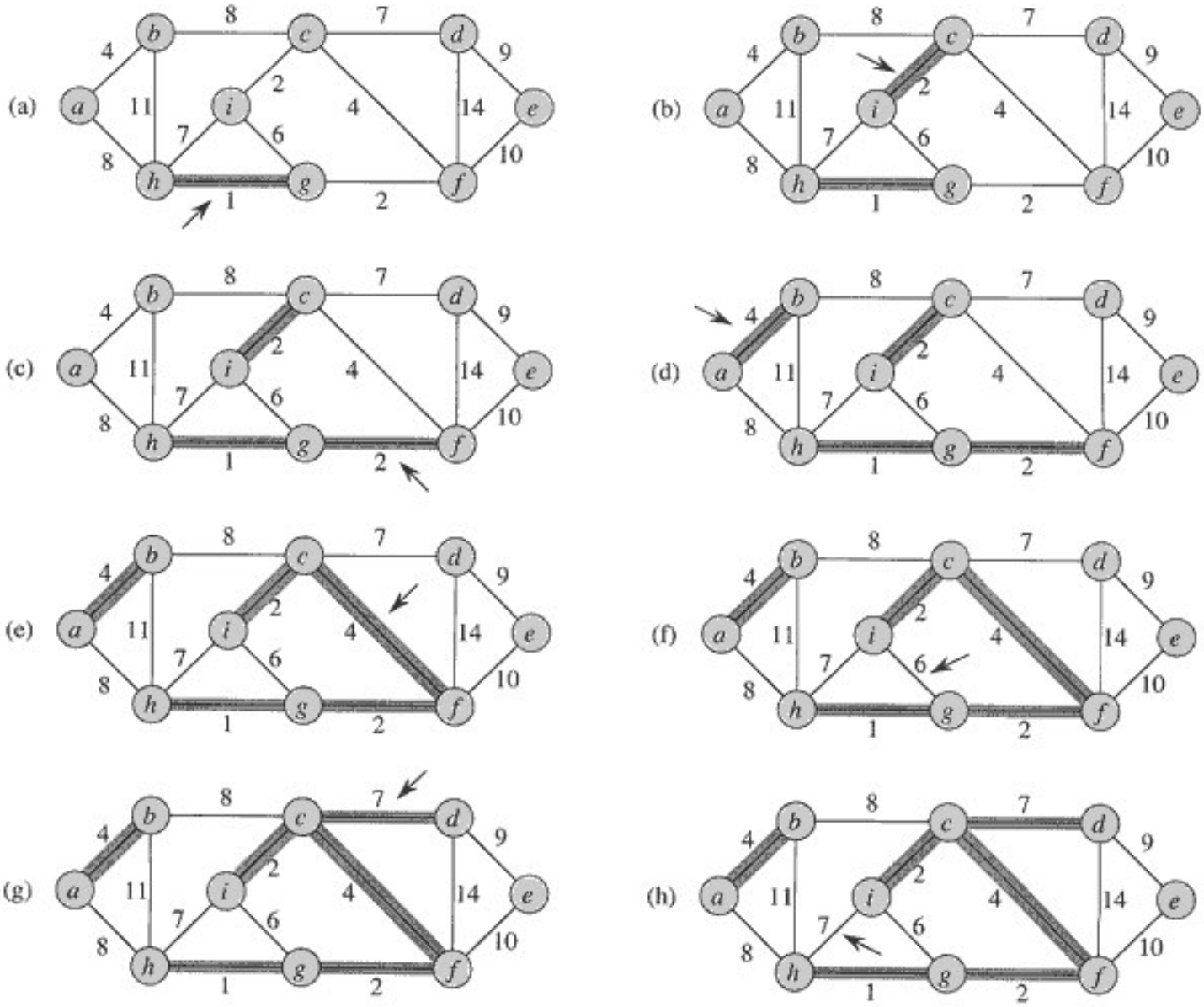
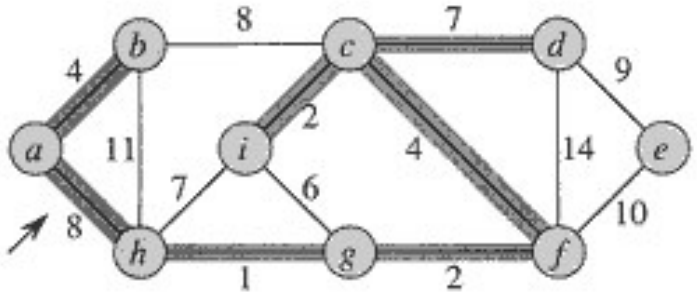
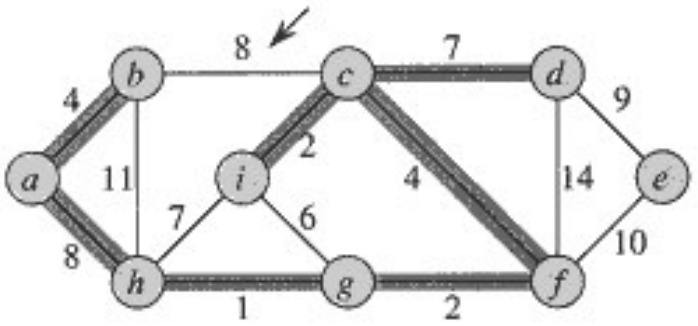


Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.

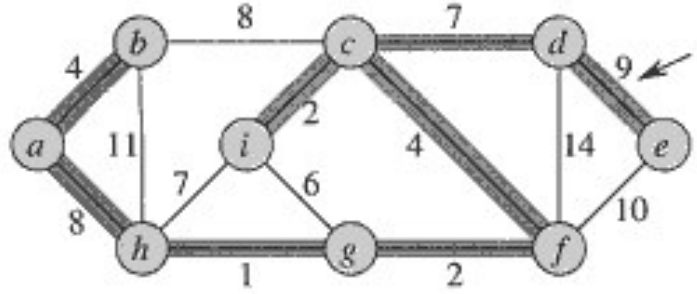
(i)


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

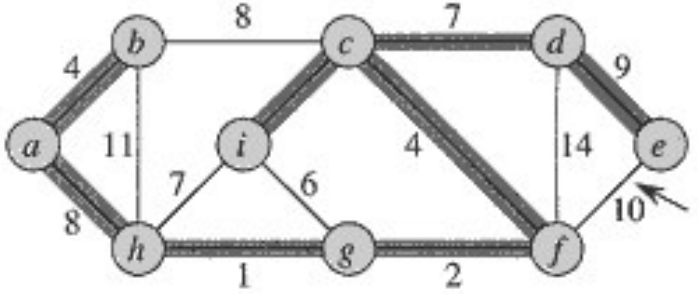
(j)


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

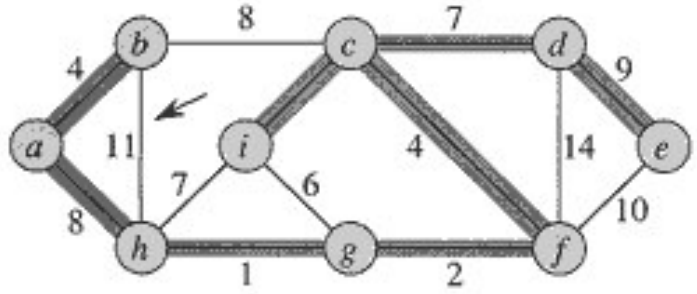
(k)


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

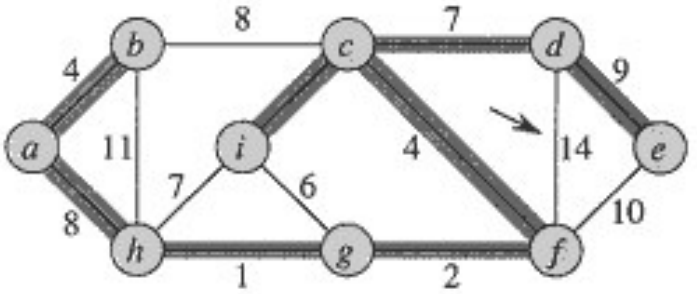
(l)


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

(m)


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

(n)


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

Algoritmo de Kruskal: Análise

- O tempo de execução do algoritmo de Kruskal sobre um grafo G = (V, E) depende da implementação da estrutura de dados conjuntos-disjuntos
  - Inicialização consome O(|V|)
  - Ordenação consome O(|E| lg |E|)
  - Se for utilizada a estrutura de dados UNION-FIND com a heurística “compressão-de-caminho” (Cormen, capítulo 21), cada operação sobre o conjunto-disjunto é da ordem O(lg |E|)

- Portanto, o tempo computacional do algoritmo de Kruskal é O(|E| lg |E|)

Algoritmo de Prim

- Assim como o algoritmo de Kruskal, Prim é um caso especial do algoritmo genérico de MST
  - Este algoritmo opera de forma similar ao algoritmo de Dijkstra para encontrar os menores caminhos em um grafo
- Este algoritmo tem a propriedade de que as arestas no conjunto A sempre formam uma única árvore
  - Esta árvore inicia-se (raiz) a partir de um vértice arbitrário e cresce até que a árvore estenda-se a todos os vértices em V

Algoritmo de Prim

- Durante a execução, os vértices que não estão em G_A residem em uma fila de prioridade Q
  - Para cada vértice v, key[v] é o peso da aresta mais leve (menor peso) conectando v a algum vértice de $V[G_A]$
  - key[v] = ∞ se tal aresta não existe
  - O parâmetro π[v] corresponde ao pai de v na MST
- Quando o algoritmo termina, a fila de prioridades Q está vazia, e a MST A para G é dada por

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}$$

Algoritmo de Prim

```
MST-PRIM(G, w, r)
 1  for each u ∈ V[G]
 2       do key[u] ← ∞
 3          π[u] ← NIL
 4  key[r] ← 0
 5  Q ← V[G]
 6  while Q ≠ ∅
 7       do u ← EXTRACT-MIN(Q)
 8          for each v ∈ Adj[u]
 9               do if v ∈ Q and w(u, v) < key[v]
10                    then π[v] ← u
11                         key[v] ← w(u, v)
```

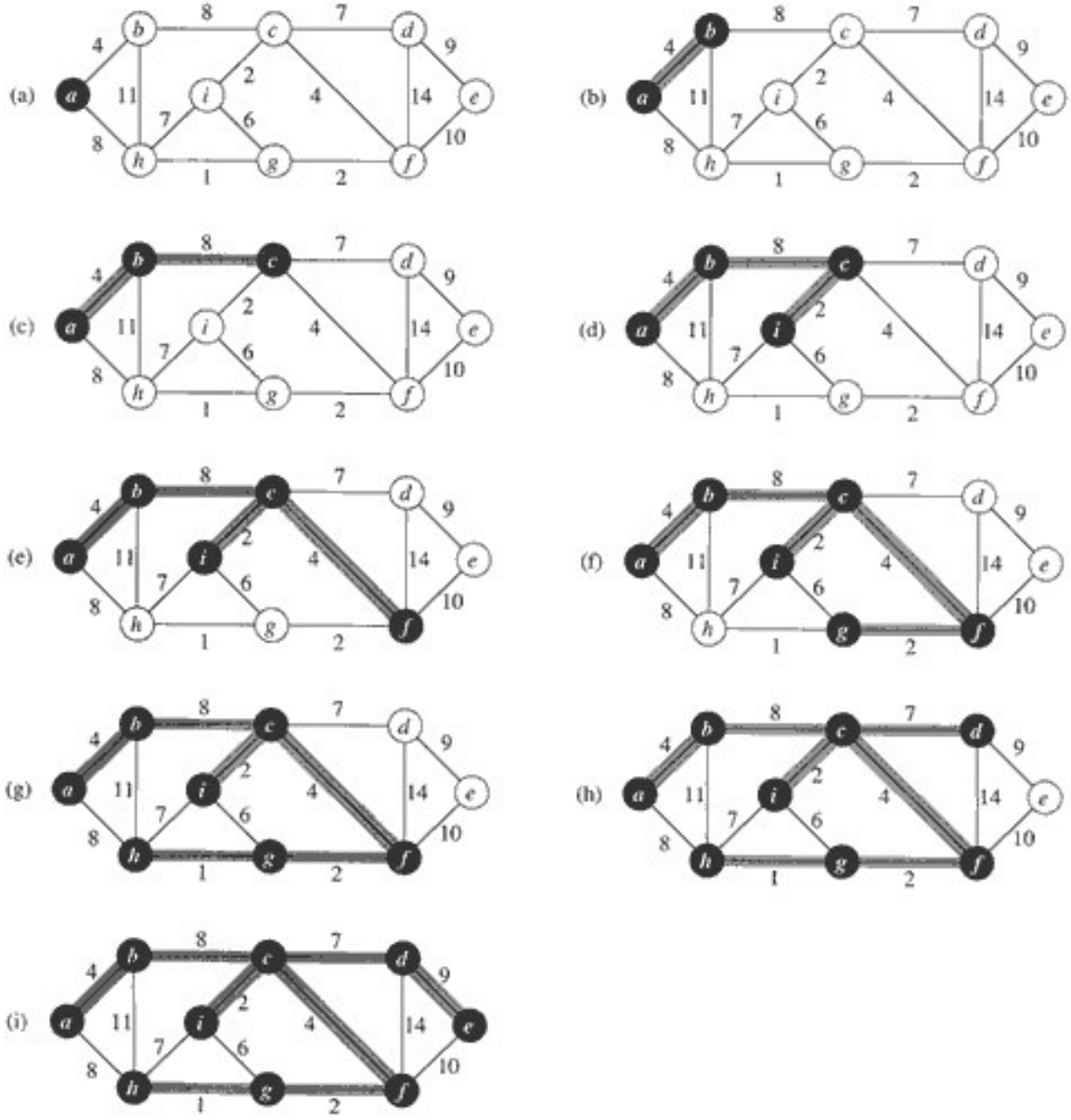


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.

Algoritmo de Prim: Análise

- O desempenho do algoritmo de Prim depende de como implementamos a fila de prioridades Q
- Se for utilizado um heap binário, então
- Os passos de inicialização de 1 a 5 são executados em tempo O(|V|)
- O laço while é executado |V| vezes
  - Uma vez que EXTRACT MIN custa O(lg |V|) as chamadas EXTRACT MIN vão custar O(|V| lg |V|)
- O laço for, 8-11, é executado |E| vezes, uma vez que o comprimento das listas de adjacências é no máximo 2|E|
  - A atribuição na linha 11 envolve operações no heap da ordem O(lg|V|)
  - Assim, o custo total de redução de chaves é O(|E| lg |V|)

Algoritmo de Prim: Análise

- Portanto, o algoritmo de Prim tem tempo de execução
  $O(|V| \lg |V| + |E| \lg |V|) \in O(|E| \lg |V|)$

- Usando heaps Fibonacci, o algoritmo de Prim pode ter complexidade reduzida para
  $O(|V| \lg |V| + |E|)$