Análise e Síntese de Algoritmos

Árvores Abrangentes de Menor Custo [CLRS, Cap. 23]

2011/2012

Contexto

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

Resumo

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 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 $w : E \to R$, 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 subconjunto 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

Algoritmo

MST-Genérico(G,w)

1 $A = \emptyset$
2 **while** A não forma árvore abrangente identificar arco seguro (u, v) para A
3 **do** $A = A \cup \{(u, v)\}$
4 **return** A

Algoritmo (greedy) genérico

Critérios de Optimalidade

- 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 $V - S$
- 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 Optimalidade

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, seja $(S, V - S)$ qualquer corte de G que respeita A, e seja (u, v) um arco leve que cruza $(S, V - S)$. 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 Optimalidade

- 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

Arco (x,y)

- $(x,y) \notin A$, porque $(S, V-S)$ respeita A
- Remoção de (x,y) divide T em dois componentes
- Inclusão de (u,v) permite formar $T' = T - \{(x,y)\} \cup \{(u,v)\}$
- 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 Optimalidade

Conclusão

- $w(T') = w(T) - w(x,y) + w(u,v) \leq w(T)$ 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 de Kruskal

Algoritmo de Kruskal

- Algoritmo 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

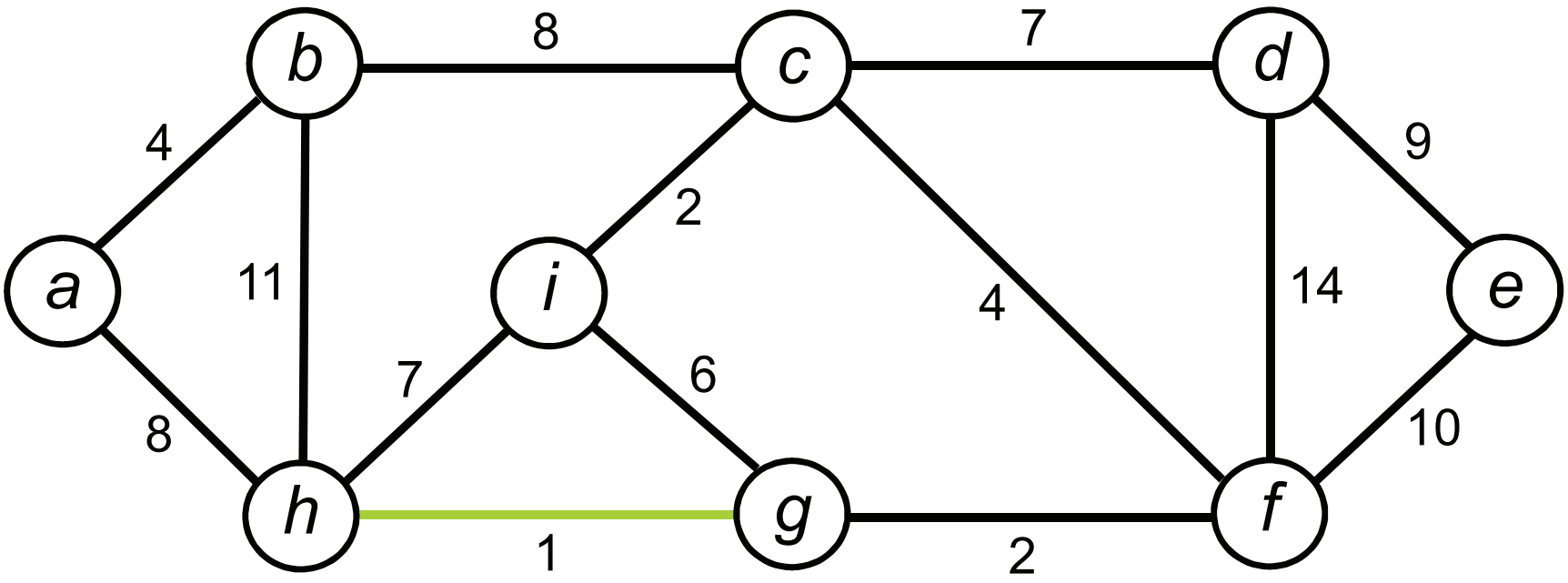
Algoritmo de Kruskal

Pseudo-Código

MST-Kruskal(G,w)

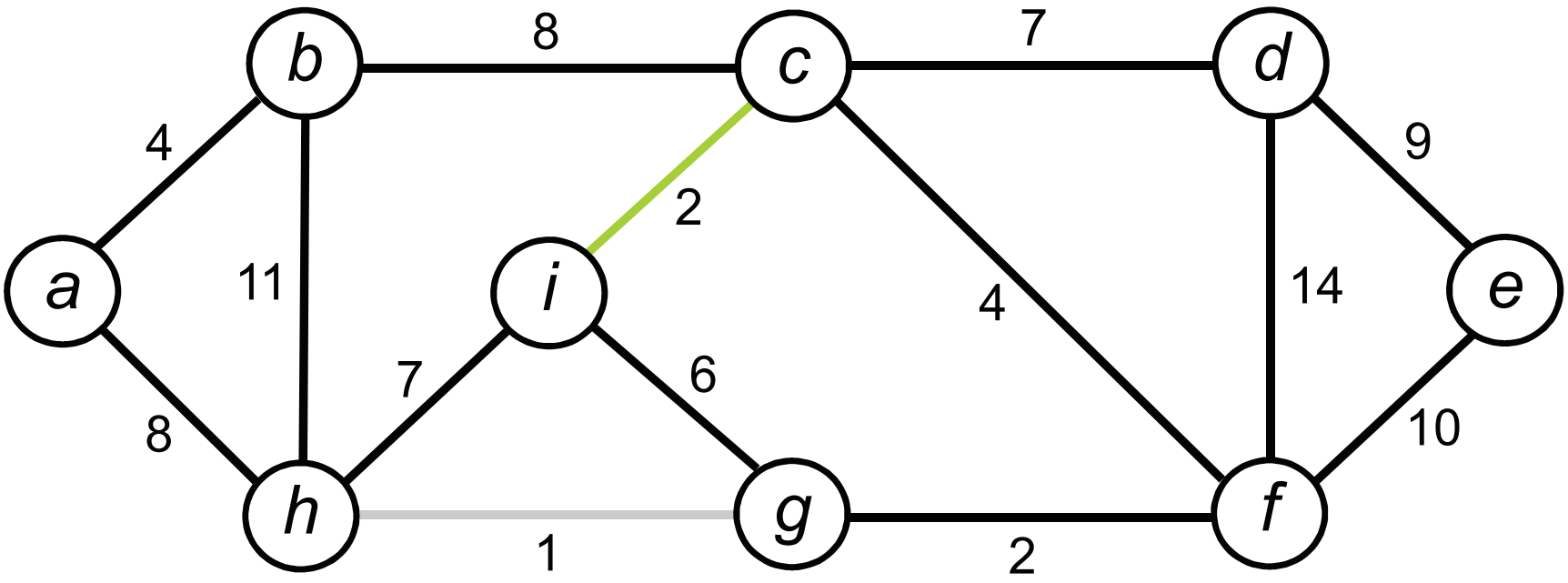
1 $A = \emptyset$
2 **for** each $v \in V[G]$ ▷ Cria conjunto para cada v
3 **do** Make-Set(v)
4 Ordenar arcos de $E[G]$ por ordem de peso não decrescente
5 **for** each $(u,v) \in E[G]$, por ordem não decrescente de $w(u,v)$
6 **do if** (Find-Set(u) $\neq$ Find-Set(v)) ▷ (u,v) é arco leve, seguro para A
7 **then** $A = A \cup \{(u,v)\}$
8 Union (u,v)
9 **return** A

Algoritmo de Kruskal



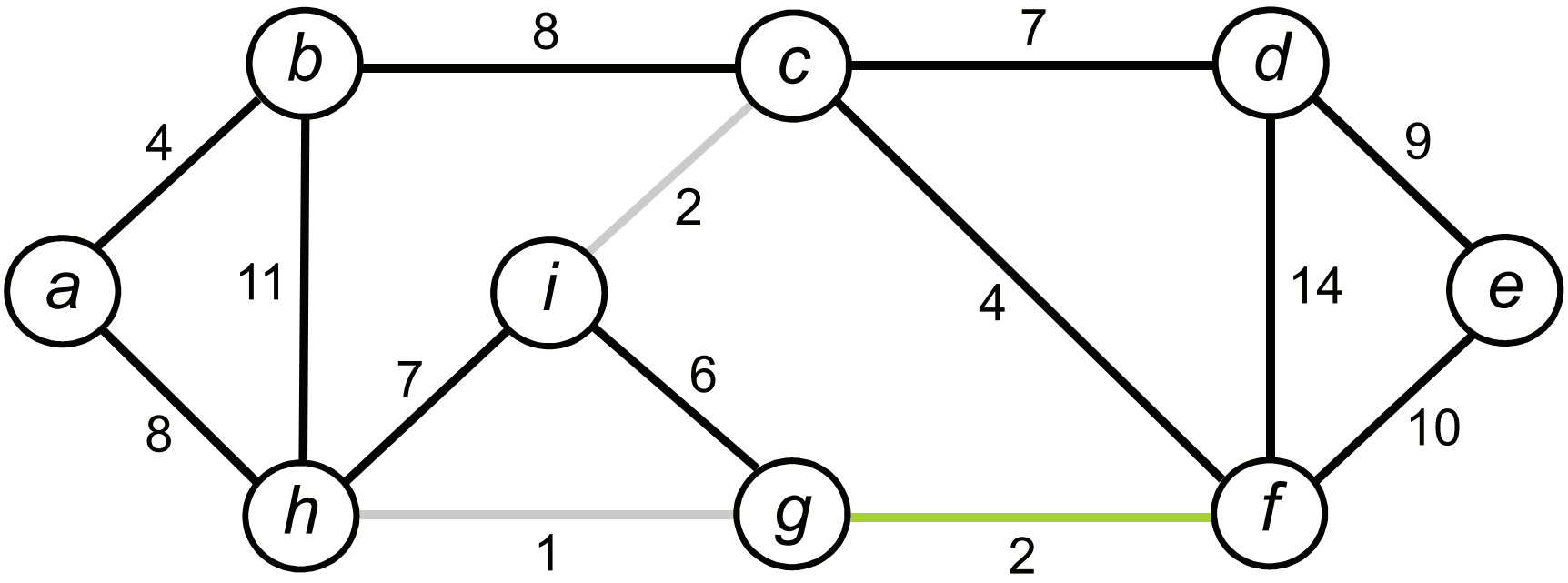
$A = \{ (h,g) \}$

Algoritmo de Kruskal



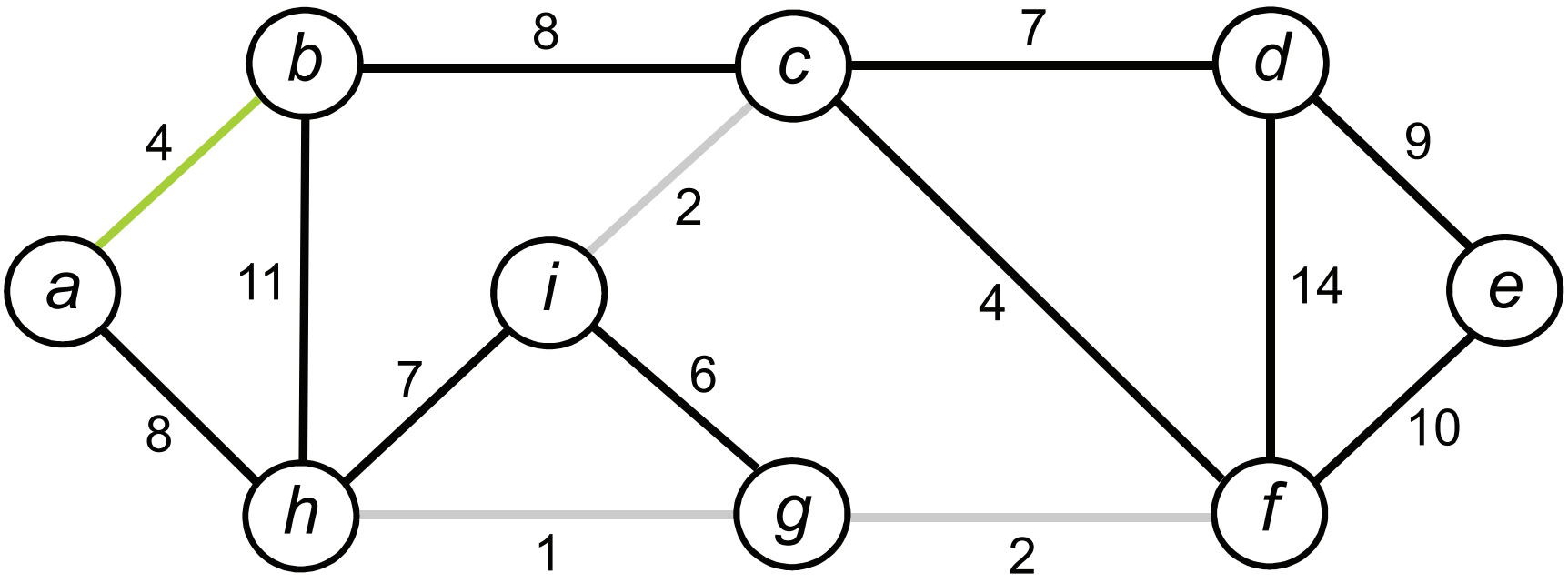
$A = \{ (h,g), (i,c) \}$

Algoritmo de Kruskal



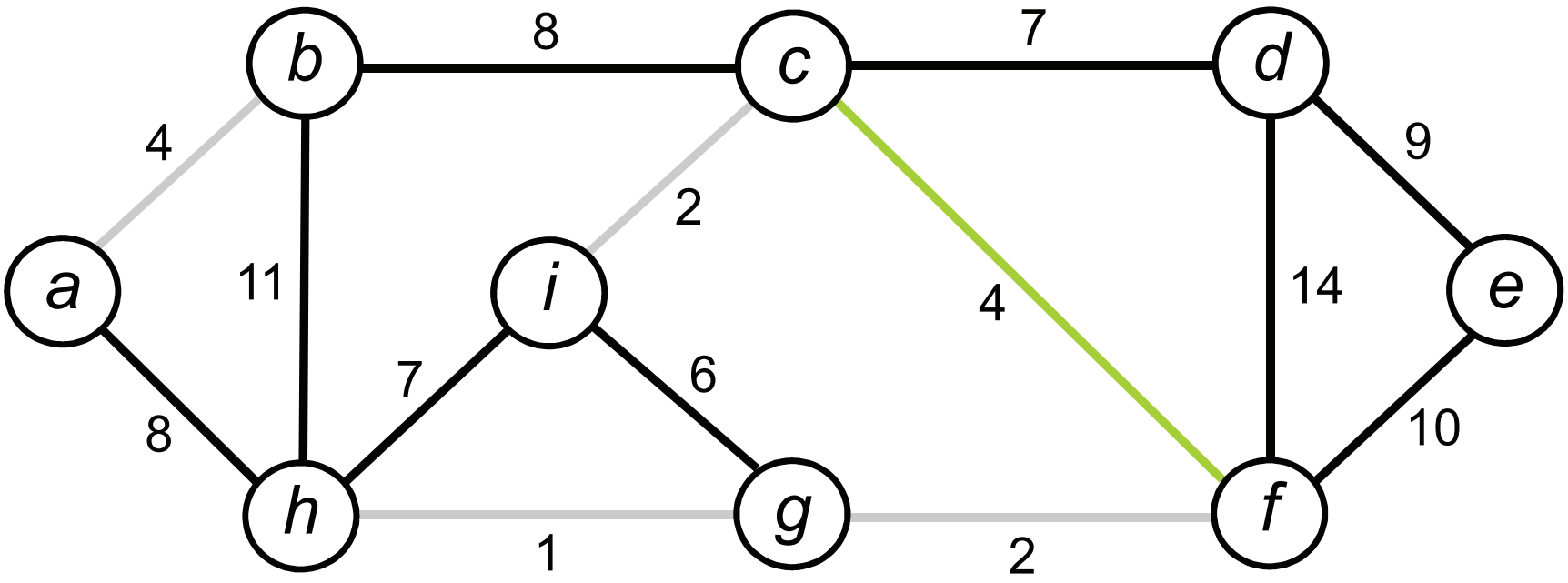
$A = \{ (h,g), (i,c), (g,f) \}$

Algoritmo de Kruskal



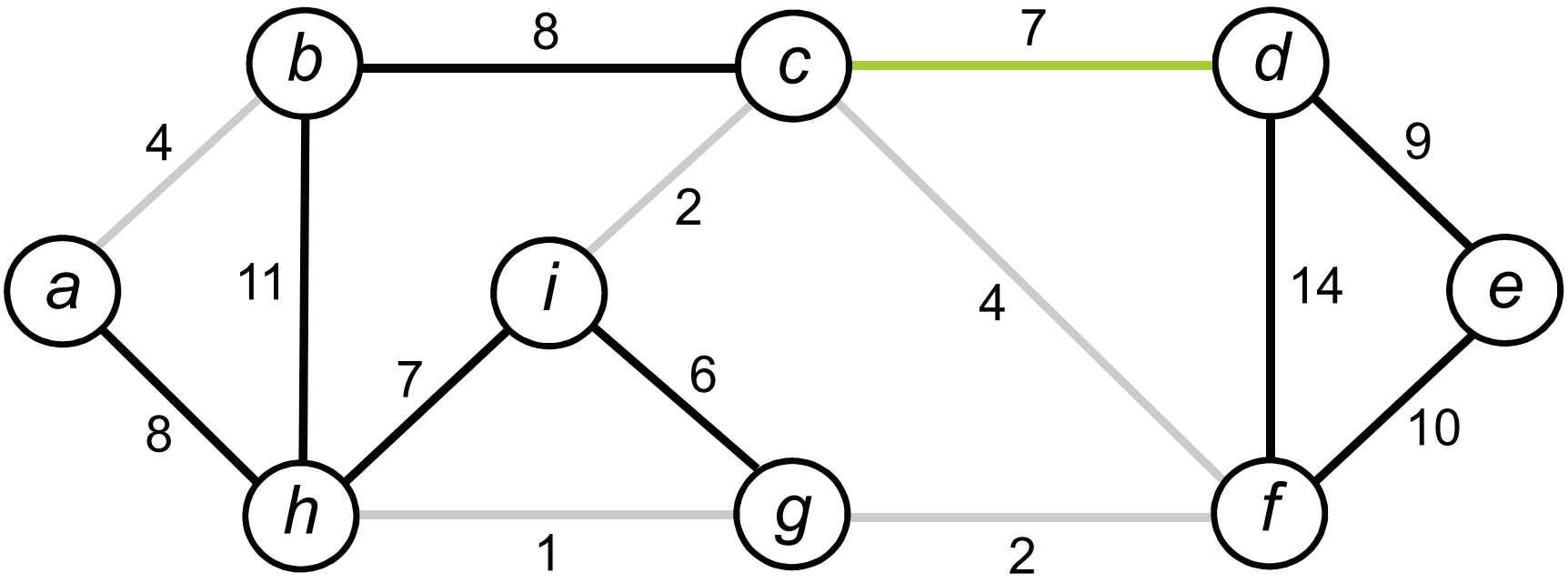
$A = \{ (h,g), (i,c), (g,f), (a,b) \}$

Algoritmo de Kruskal



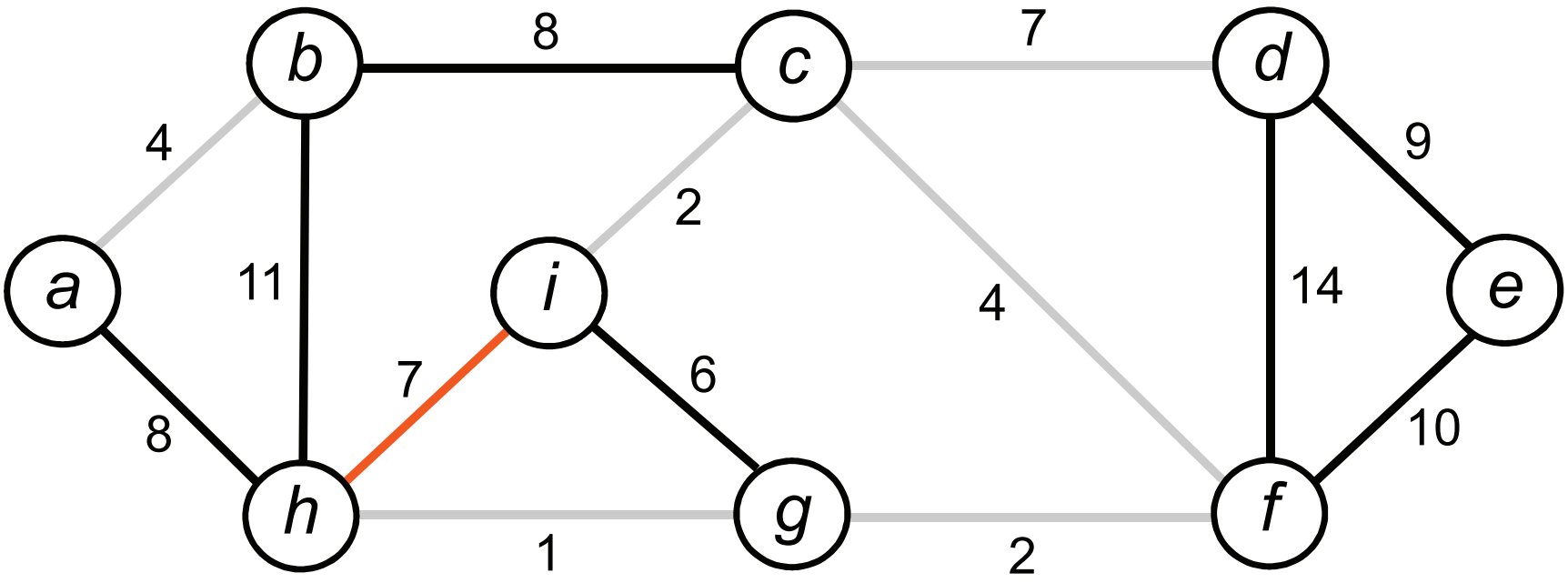
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f) \}$

Algoritmo de Kruskal



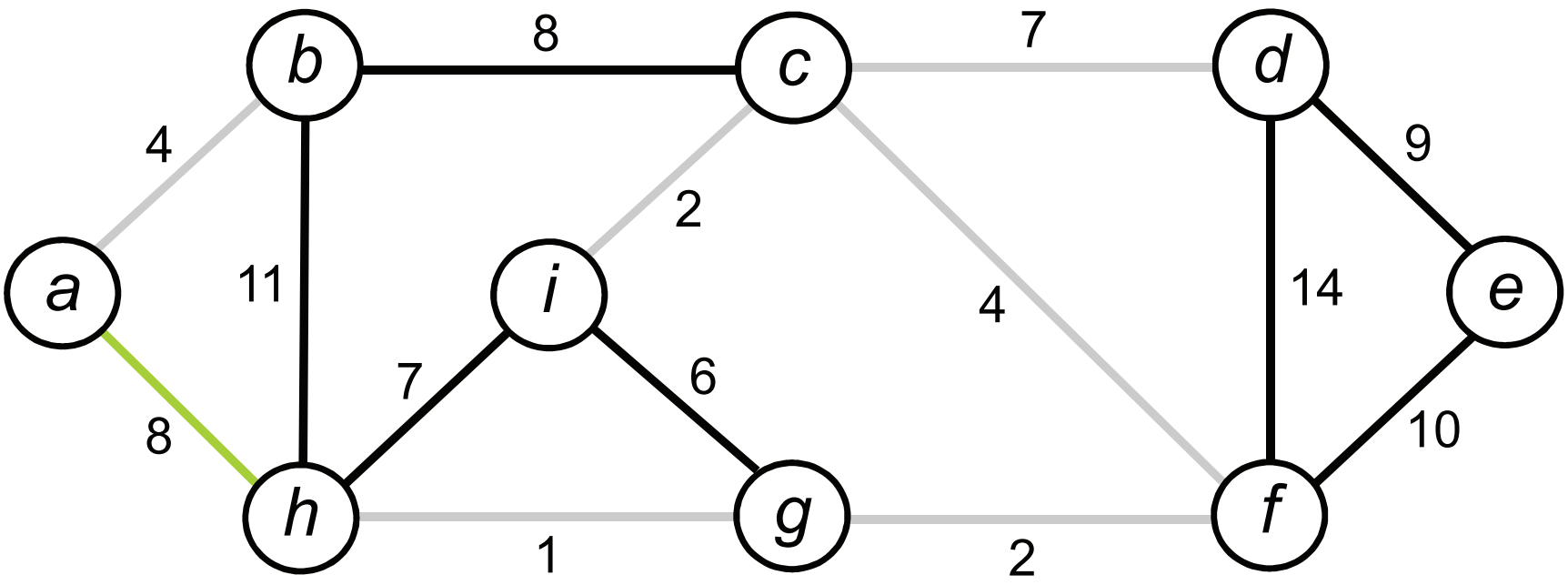
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d) \}$

Algoritmo de Kruskal



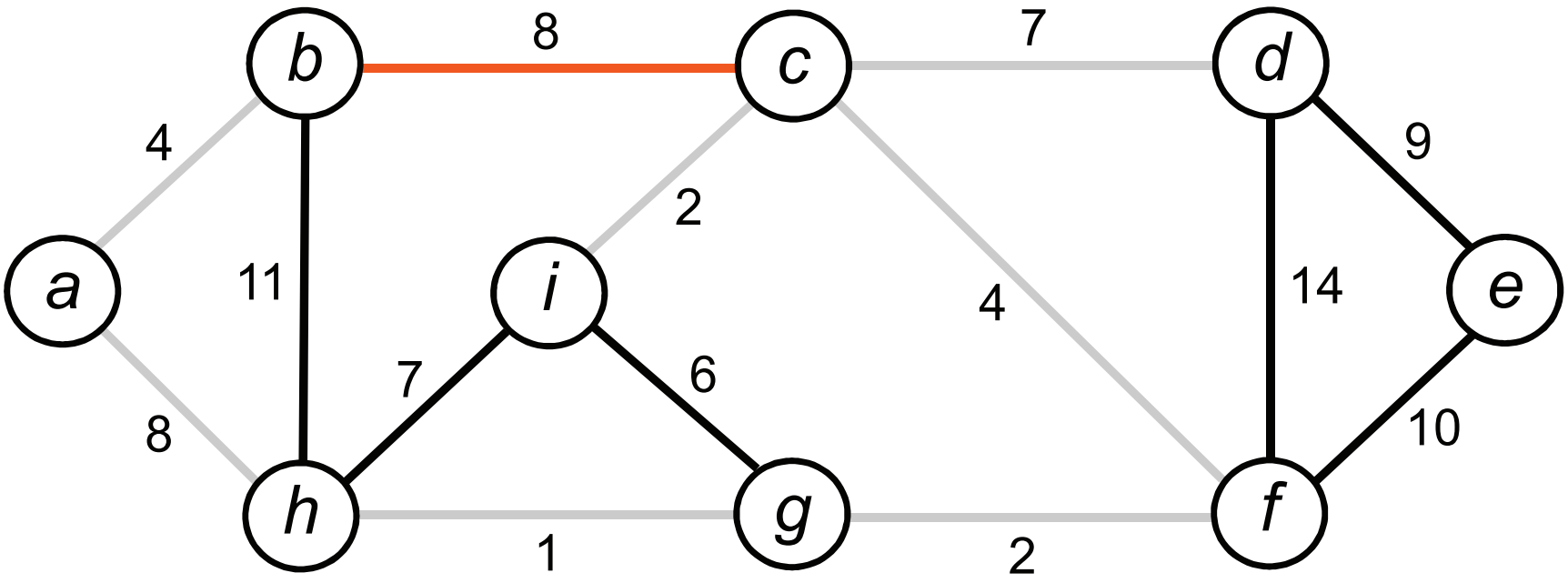
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d) \}$

Algoritmo de Kruskal



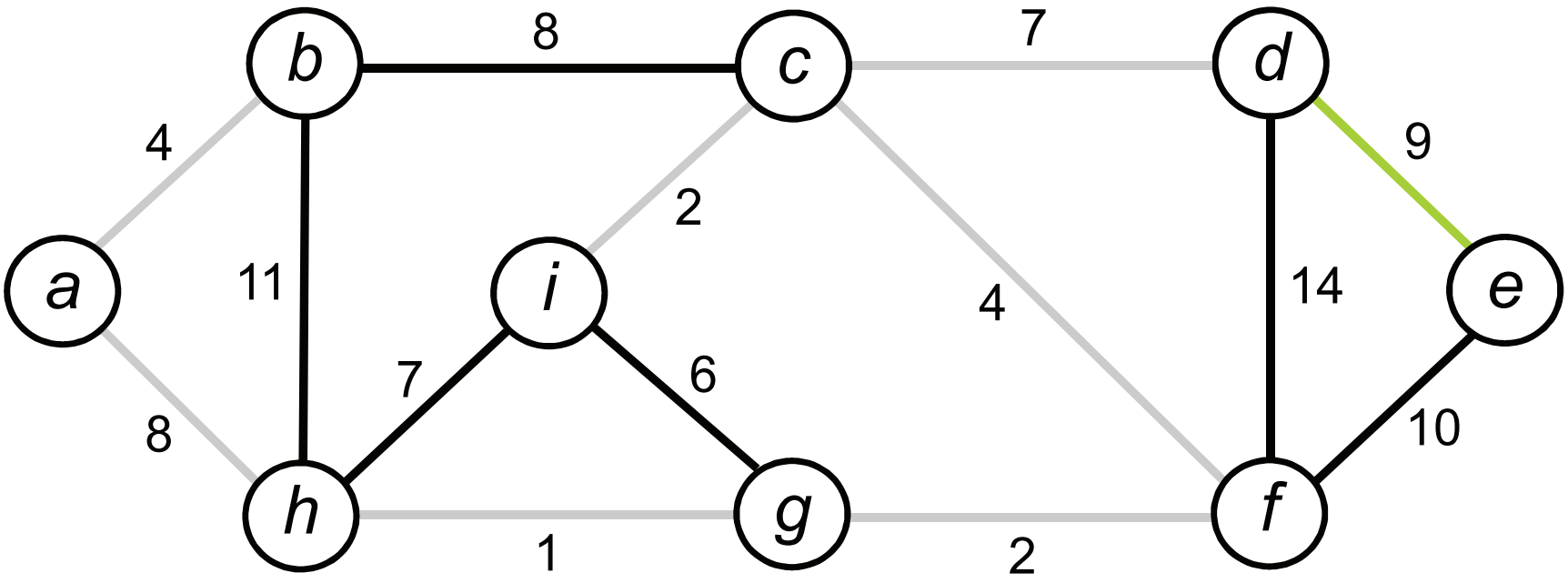
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d), (a,h) \}$

Algoritmo de Kruskal



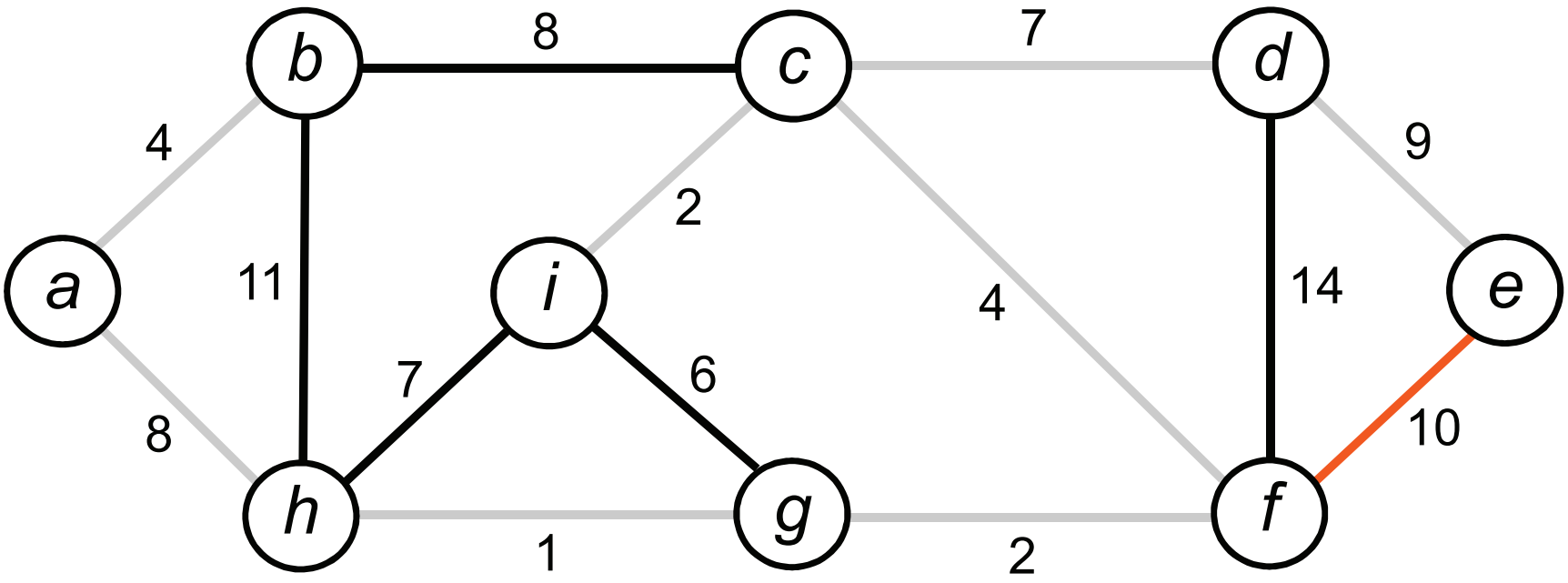
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d), (a,h) \}$

Algoritmo de Kruskal



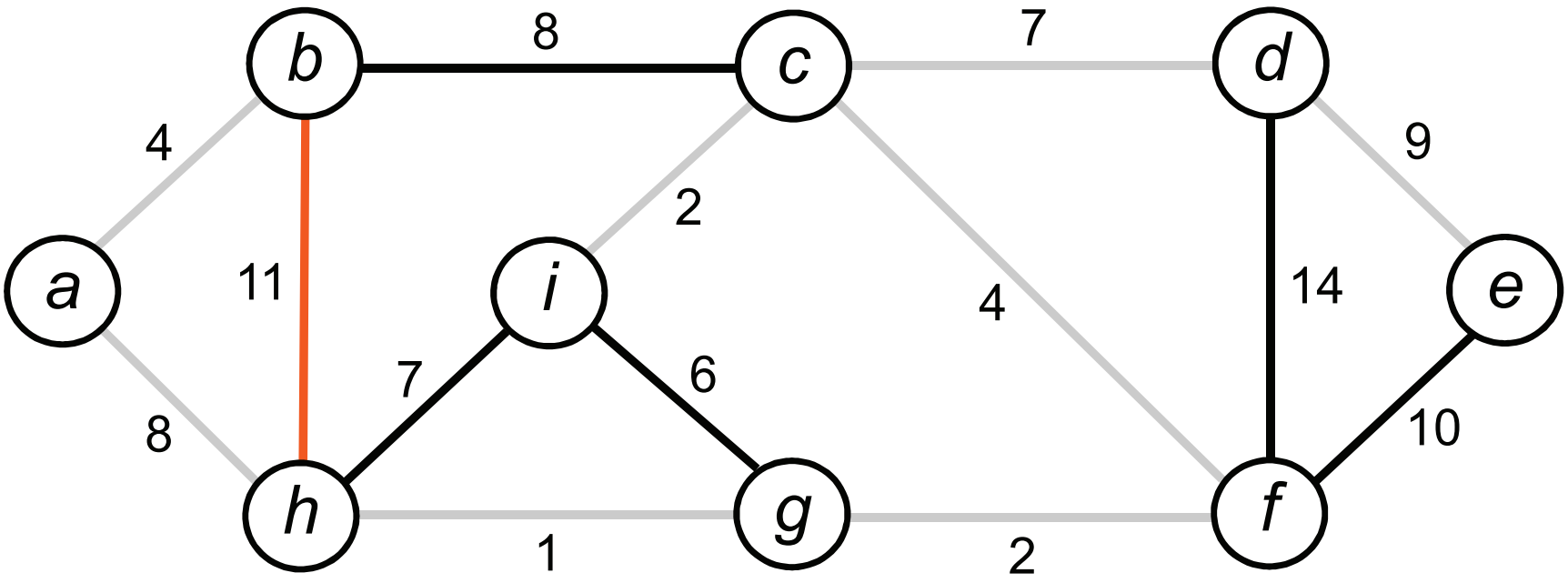
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d), (a,h), (d,e) \}$

Algoritmo de Kruskal



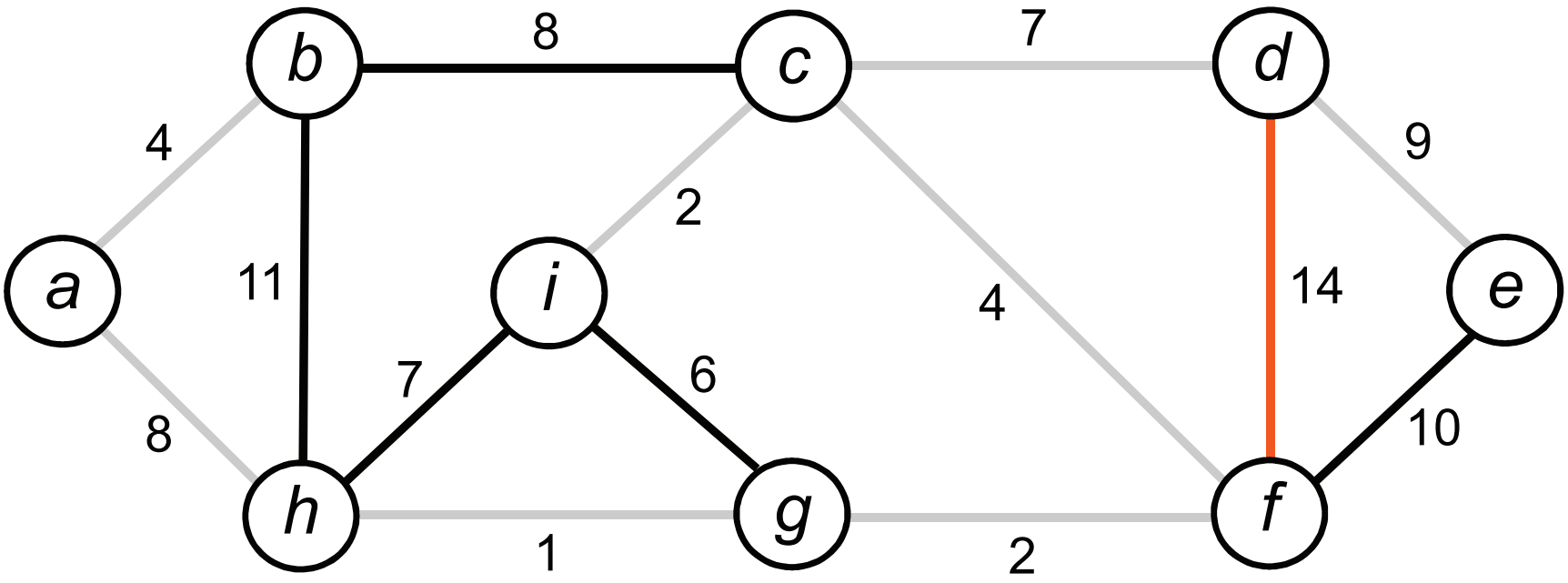
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d), (a,h), (d,e) \}$

Algoritmo de Kruskal



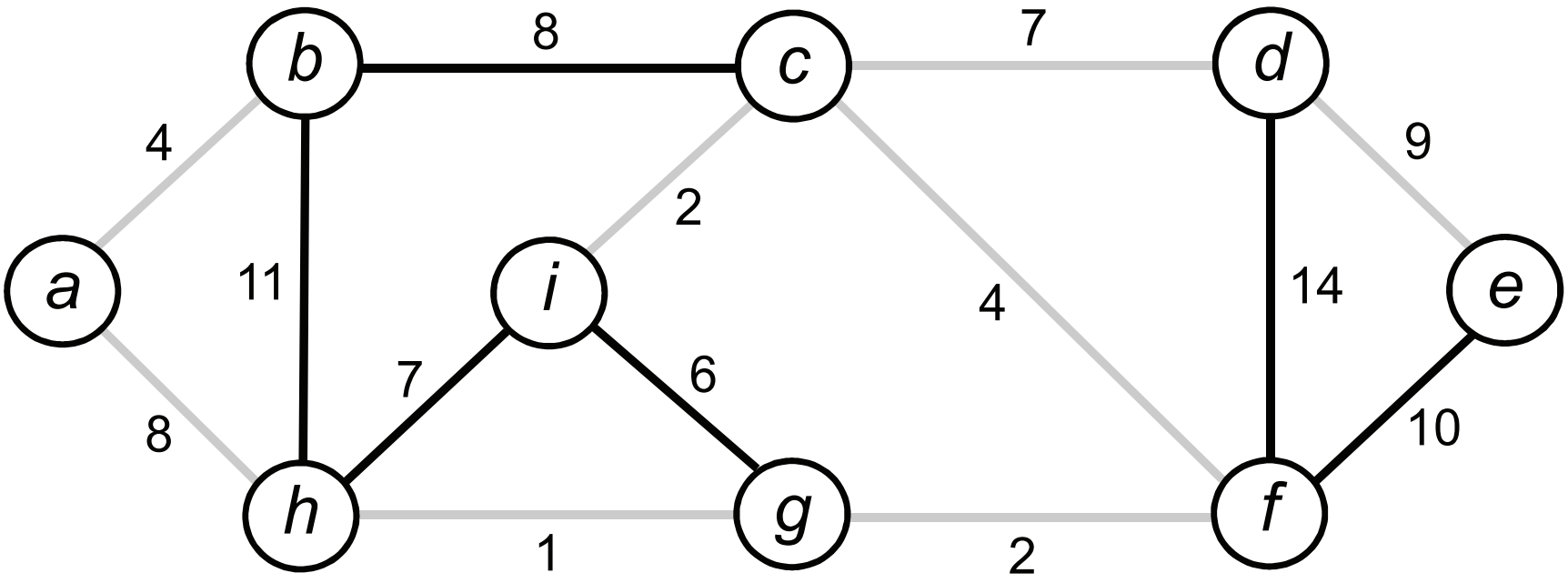
$A = \{ (h,g), (i,c), (g,f), (a,b), (c,f), (c,d), (a,h), (d,e) \}$

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 $O((V+E)\,\alpha(E,V))$
  - Como $|E| \geq V-1$ porque o grafo é ligado, então temos $O(E\,\alpha(E,V))$
- Logo, é possível assegurar $O(E \lg E)$
- Dado que $E < V^2$, obtém-se também $O(E \lg V)$

Algoritmo de Borůvka

Algoritmo de Borůvka

- Começar com um N conjuntos de vértices (N árvores)
  - Noção de arco marcado
- A cada passo do algoritmo,
  - Para cada árvore T
    - seleccionar arco de menor peso incidente em T
    - marcar arco seleccionado (passa a estar incluído em T)
    - arco pode ser duplamente seleccionado
  - Juntar na mesma árvore as árvores ligadas por arcos marcados
- Algoritmo termina quando existir apenas uma árvore

Algoritmo de Borůvka

Complexidade

- Número de passos do algoritmo: $O(lg\ V)$
- Número de arcos analisados em cada passo: $O(E)$
- Manutenção das árvores em cada passo: $O(E)$
- Logo, é possível assegurar $O(E\ lg\ V)$

Correcção do Algoritmo

- Algoritmo correcto apenas se pesos com valores distintos
- Problema: Escolha de dois arcos de peso igual que ligam duas árvores!
- Com pesos iguais é necessário estabelecer relação de ordem (artificial) entre todos os arcos (e.g. ordem lexicográfica)

Algoritmo de Prim

Algoritmo de Prim

- MST construída a partir de um vértice raíz r
- Algoritmo mantém sempre uma árvore A
- 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

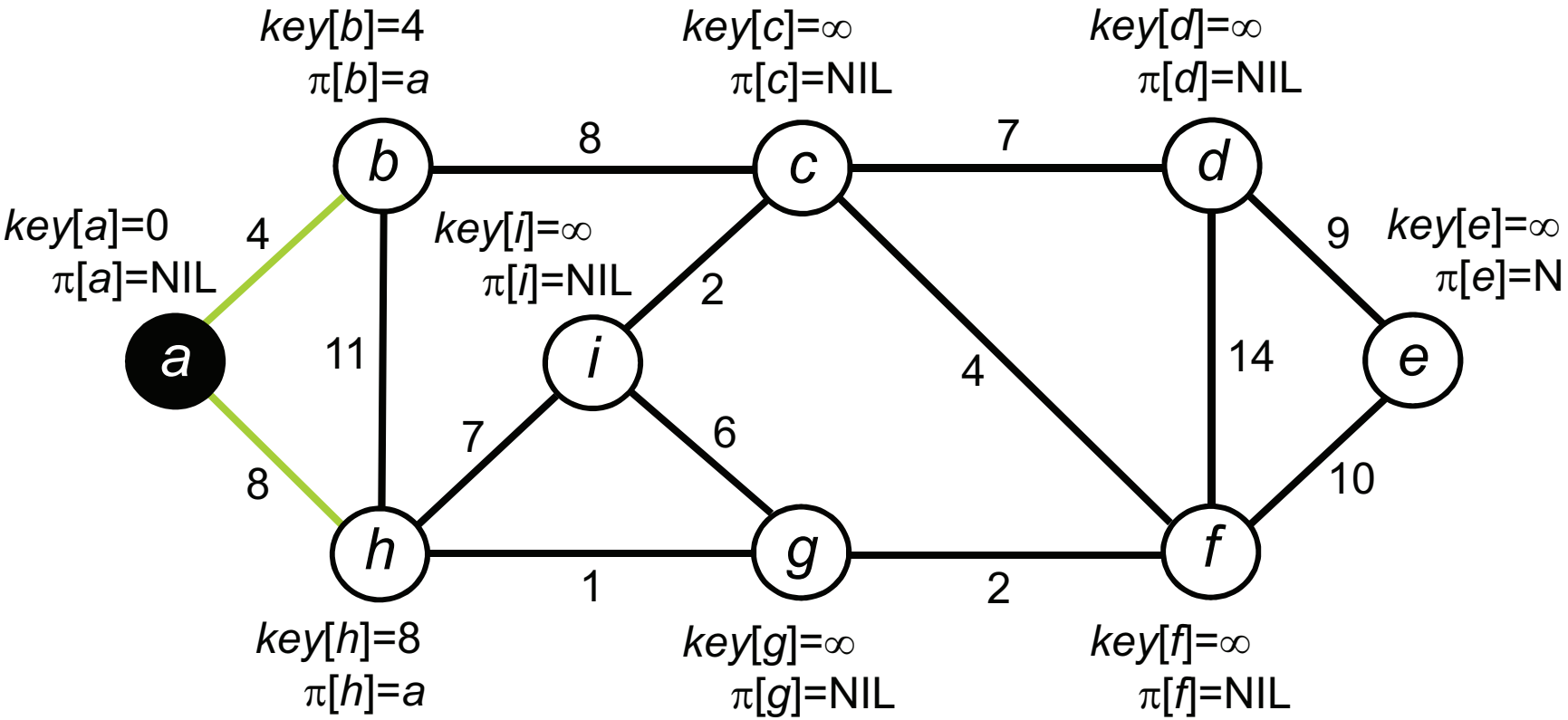
Algoritmo de Prim

Pseudo-Código

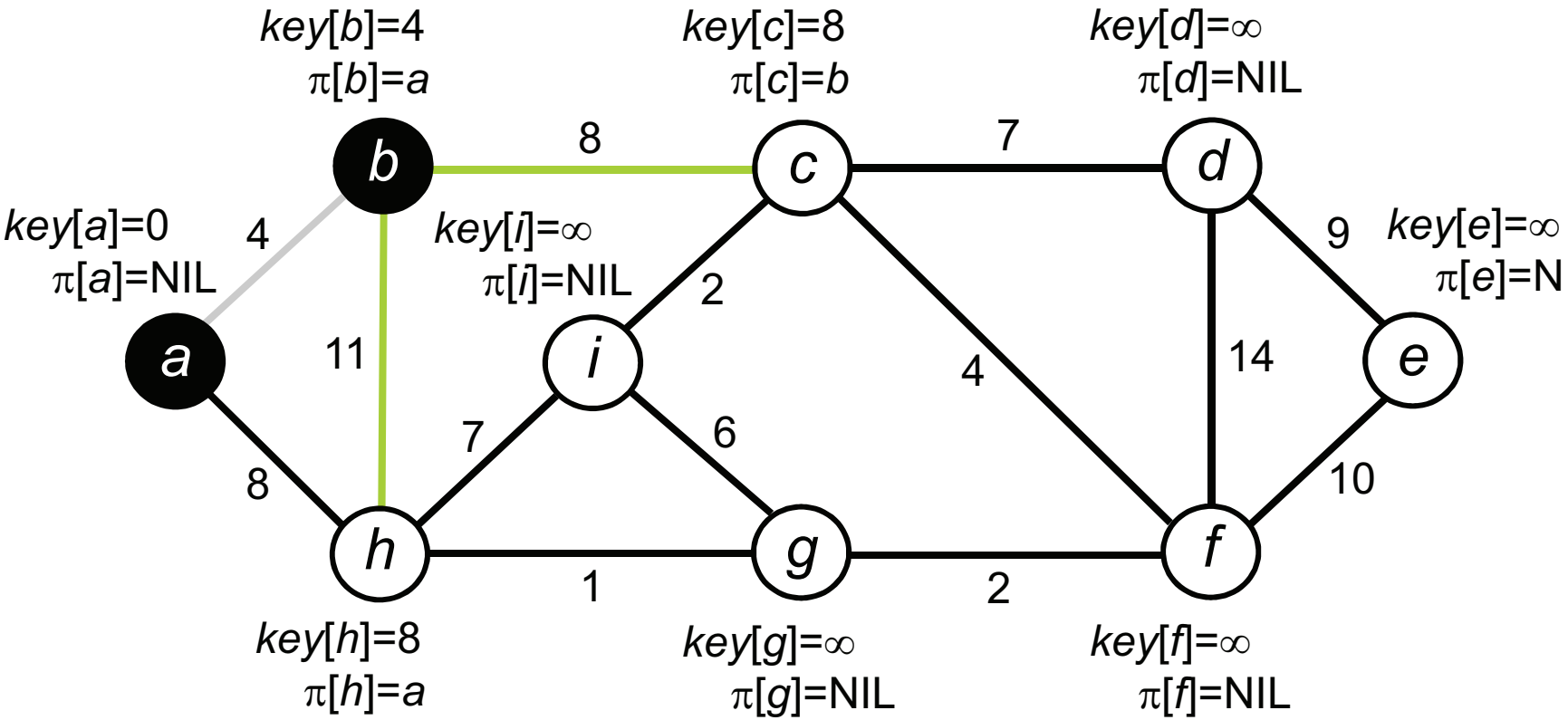
MST-Prim(G,w,r)

```
 1  for each u ∈ V[G]                          ▷ Inicialização
 2        do key[u] = ∞
 3           π[u] = NIL
 4  key[r] = 0
 5  Q = V[G]                                   ▷ Fila de Prioridade
 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) ▷ Actualização de Q
```

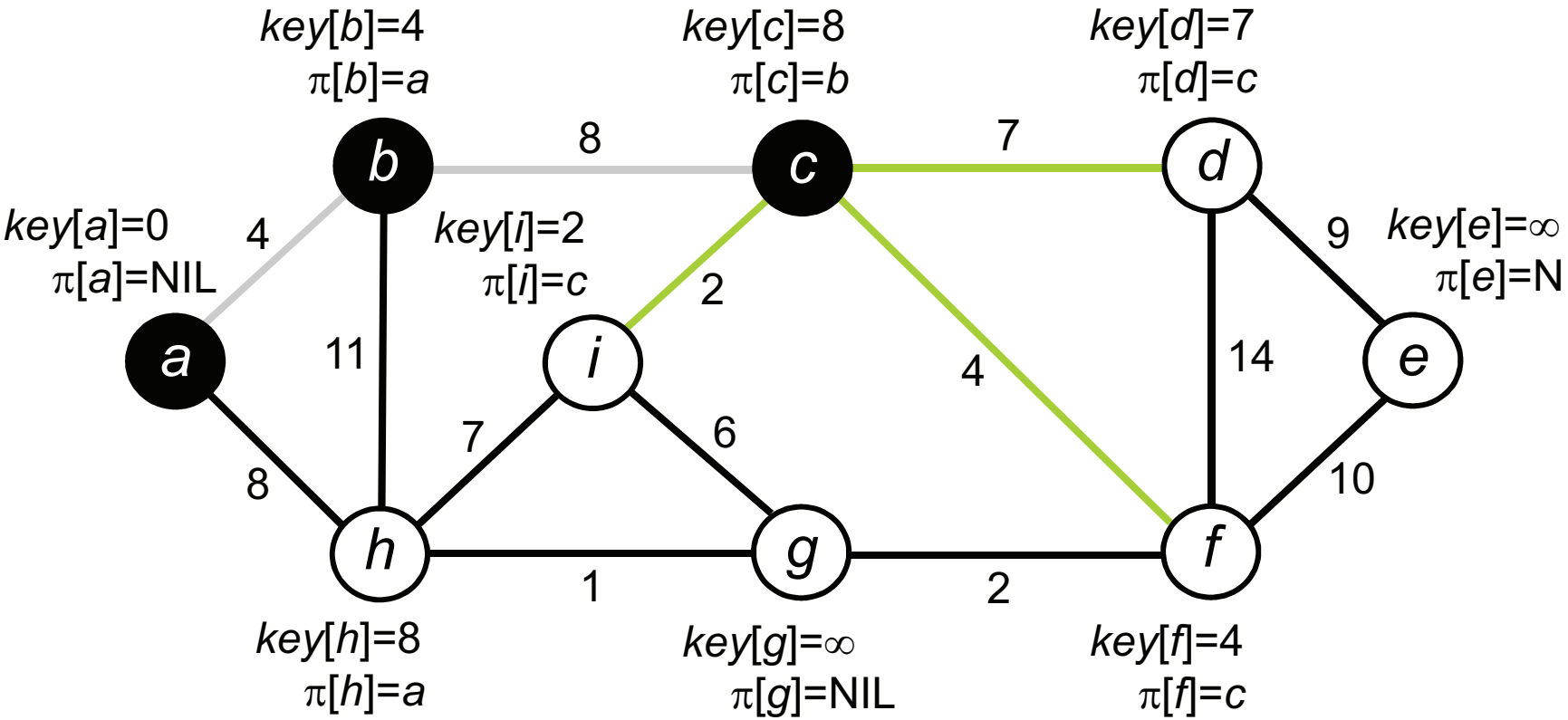
Algoritmo de Prim



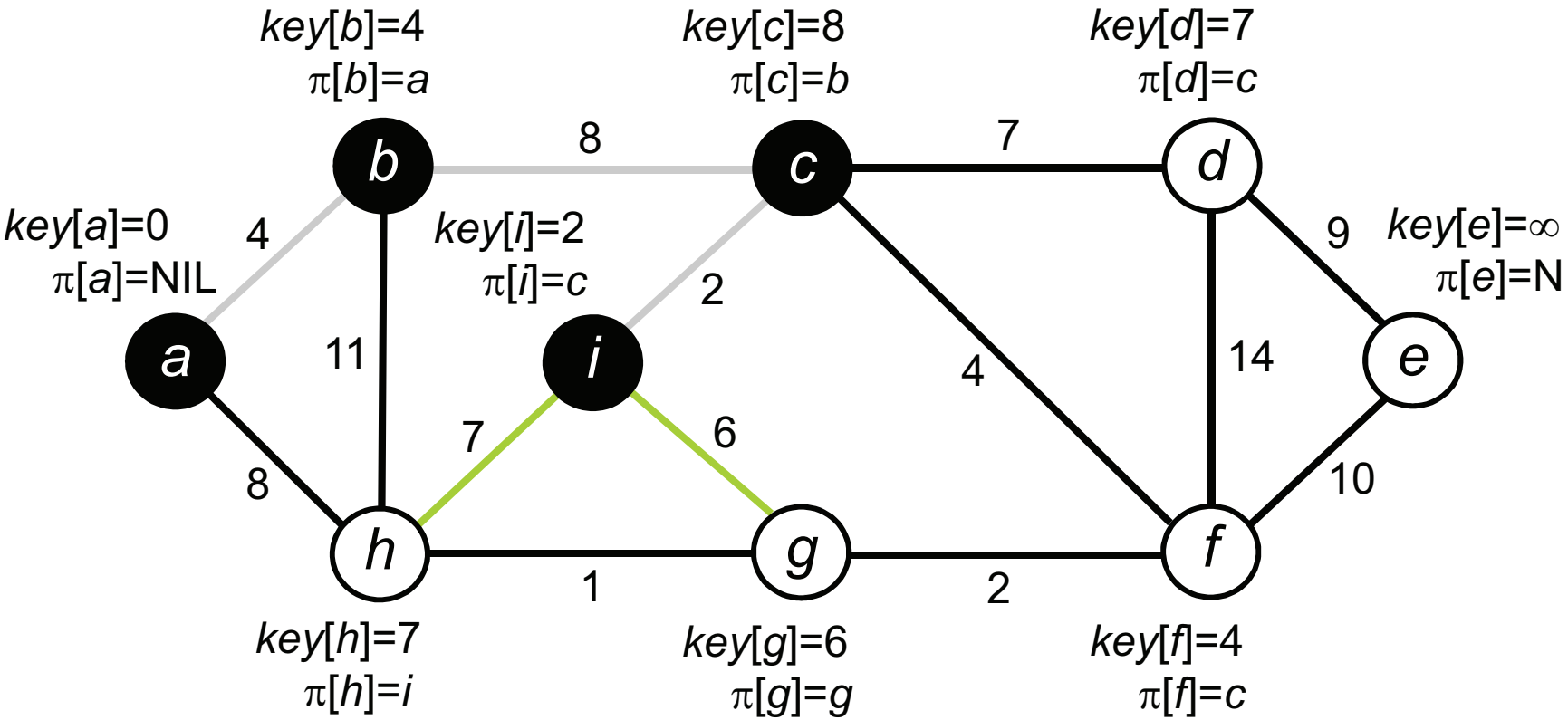
Algoritmo de Prim



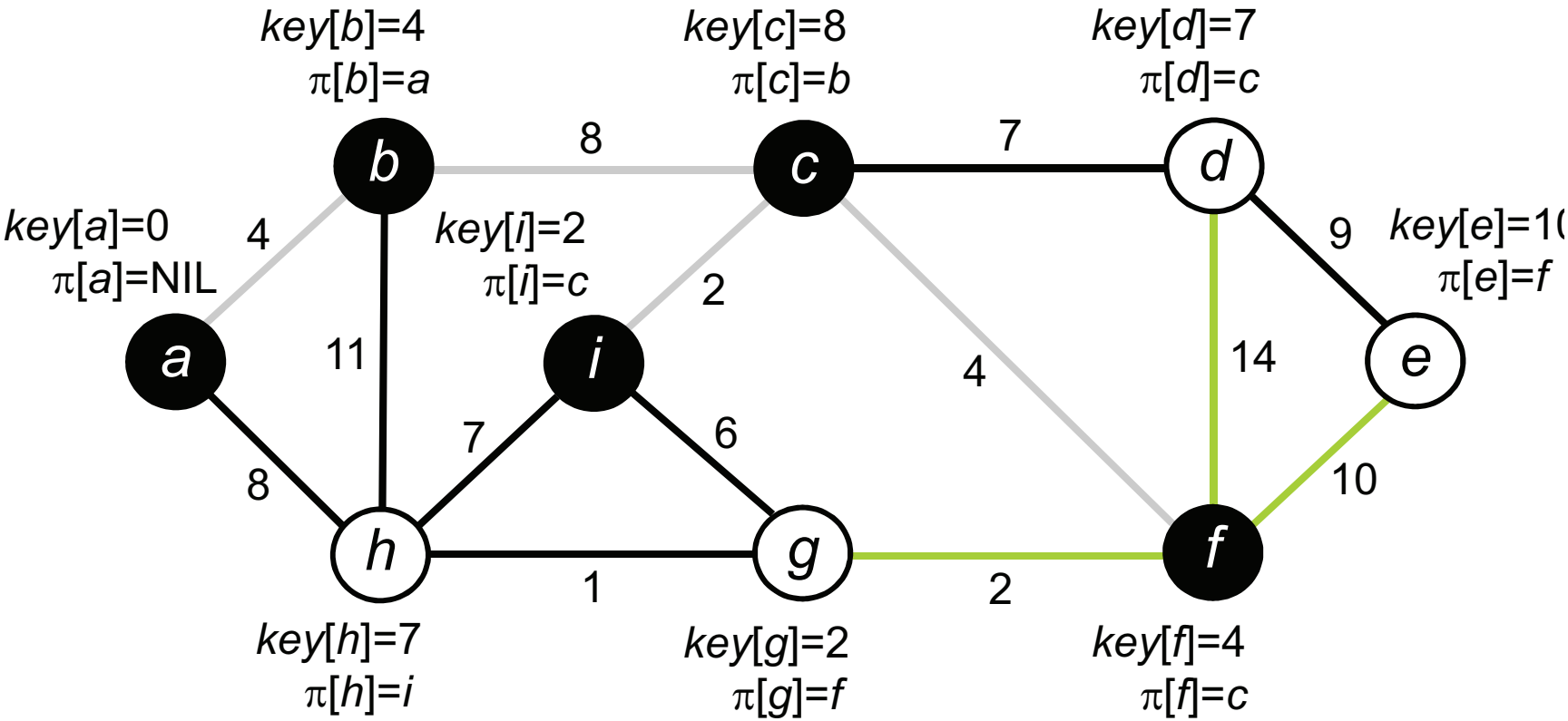
Algoritmo de Prim



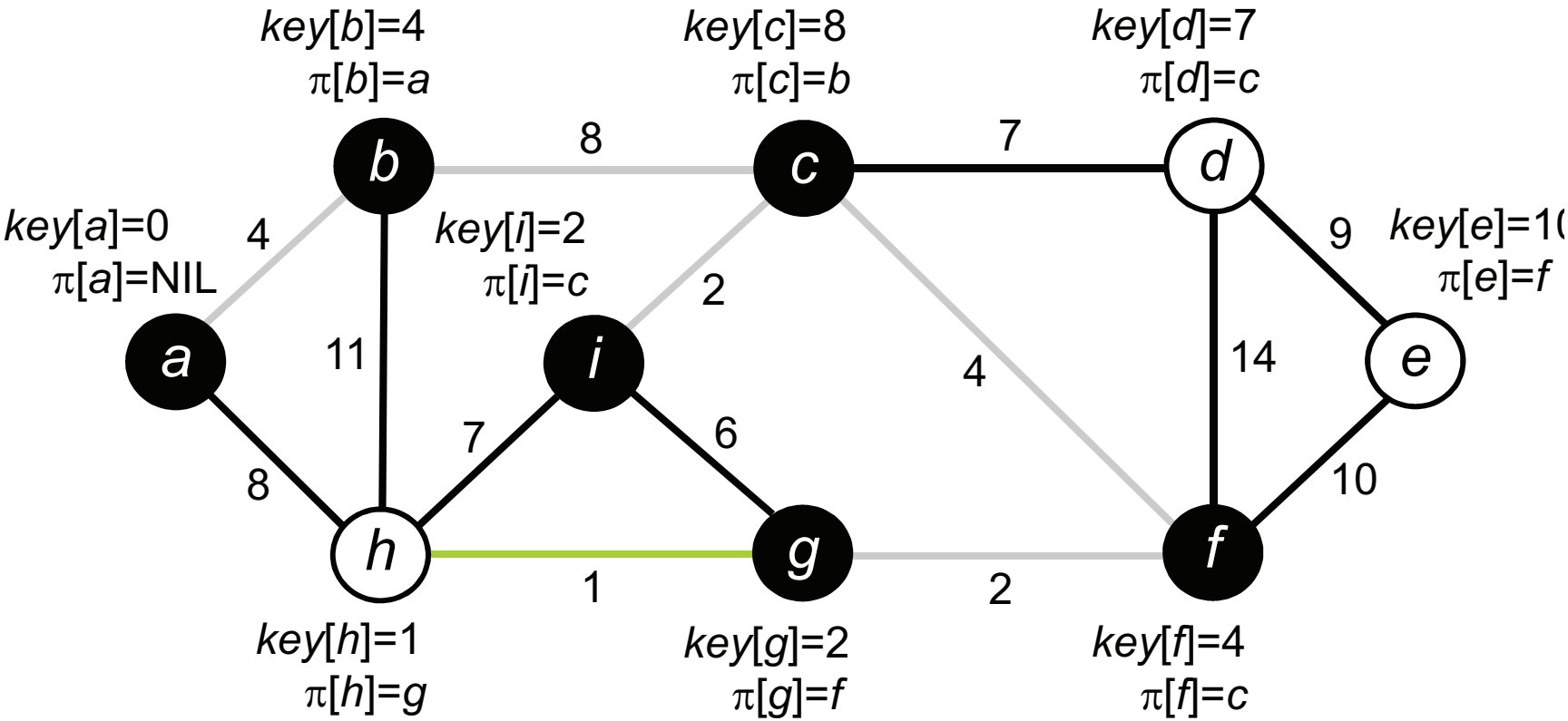
Algoritmo de Prim



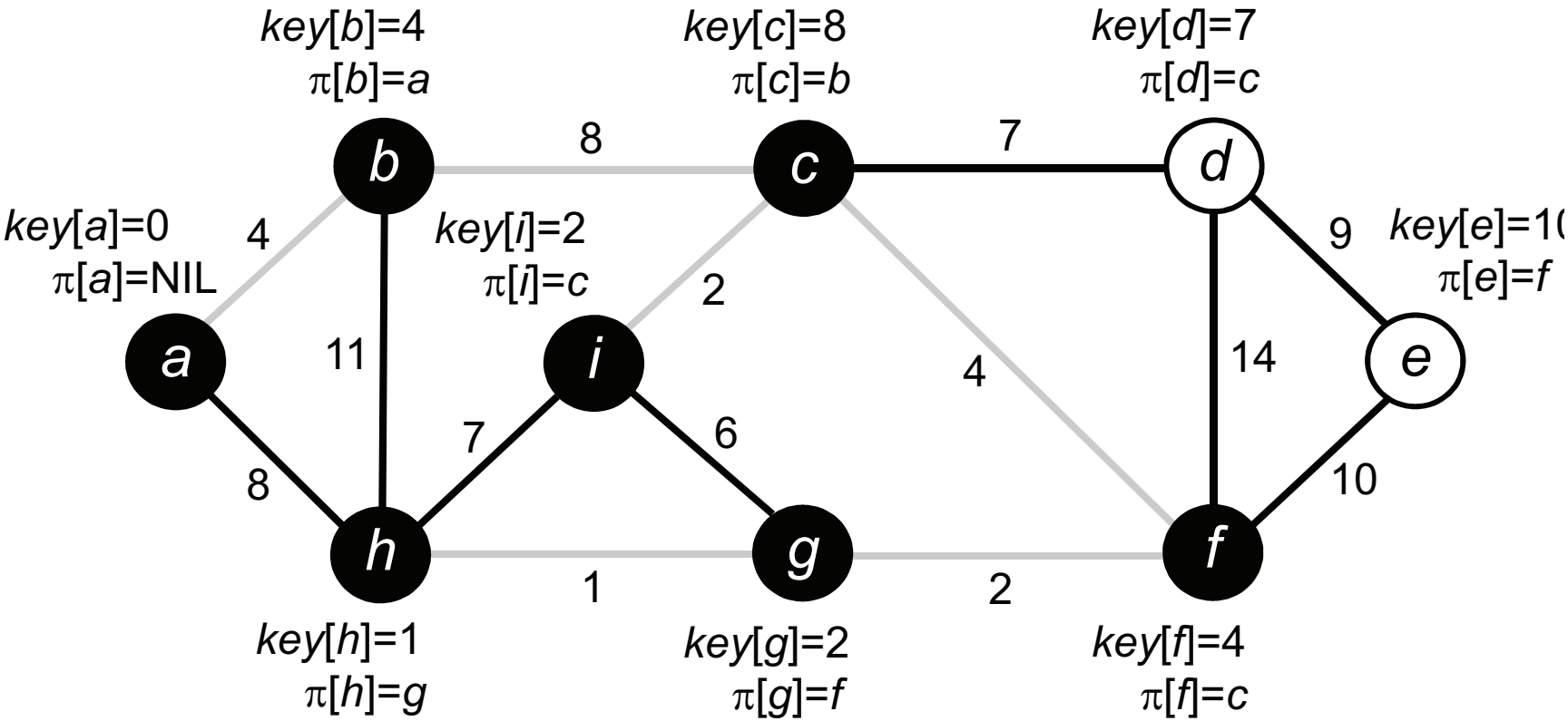
Algoritmo de Prim



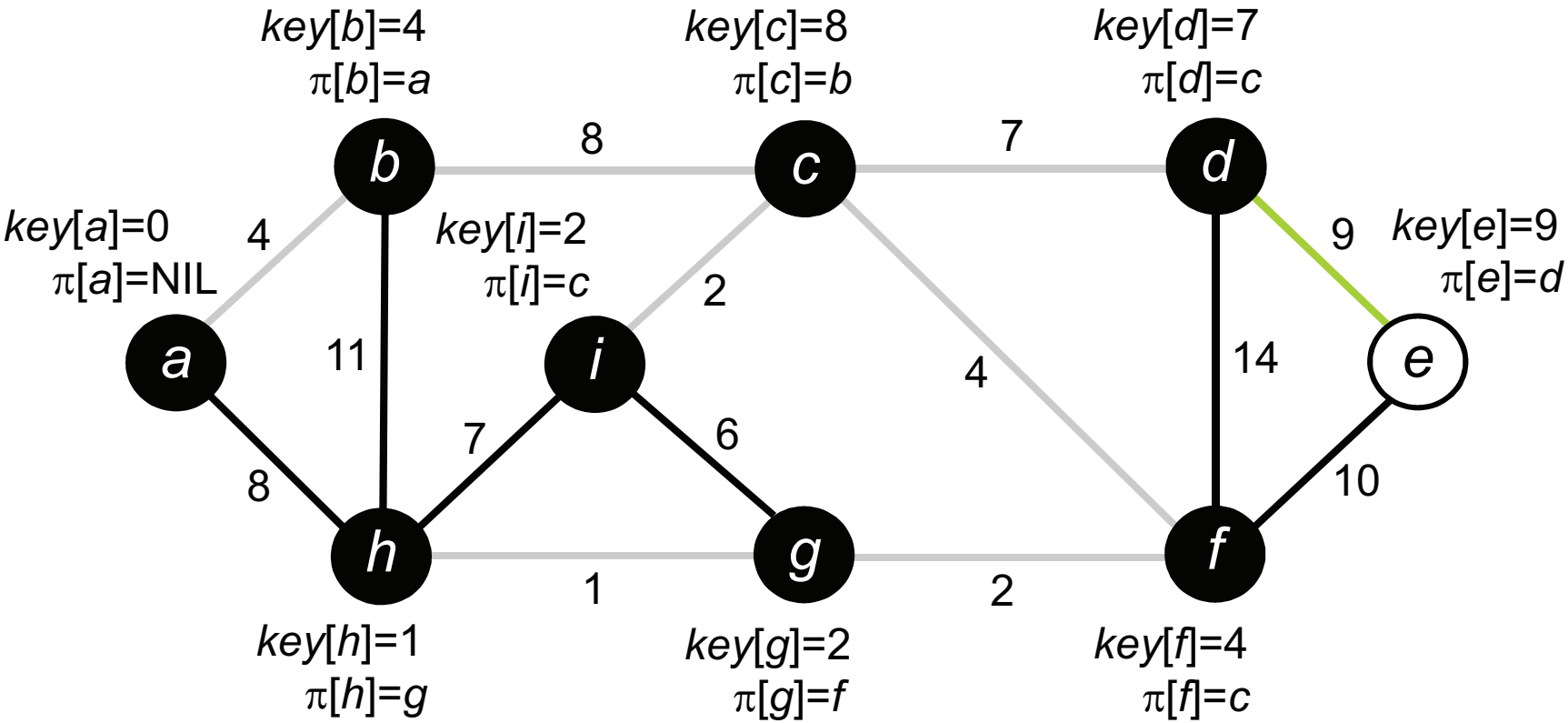
Algoritmo de Prim



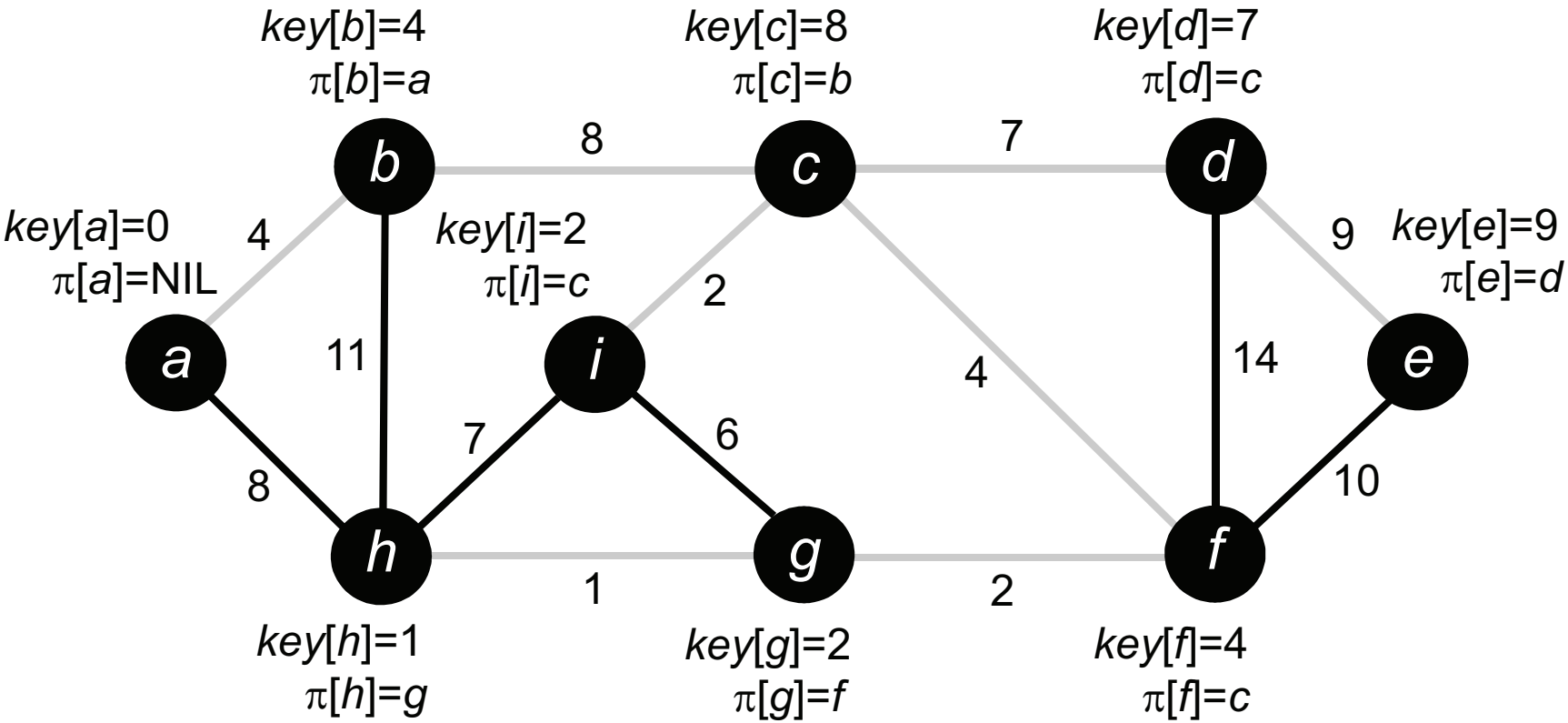
Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim

Complexidade

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