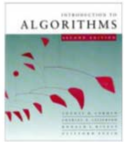
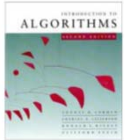


Minimum Spanning Trees



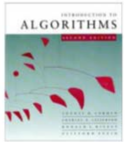
Overview

Problem

- A town has a set of houses and a set of roads.
- A road connects 2 and only 2 houses.
- A road connecting houses *u* and *v* has a repair cost $w(u, v)$.

- ***Goal:*** Repair enough (and no more) roads such that

1. everyone stays connected: can reach every house from all other houses, and
2. total repair cost is minimum.



Overview

Model as a graph

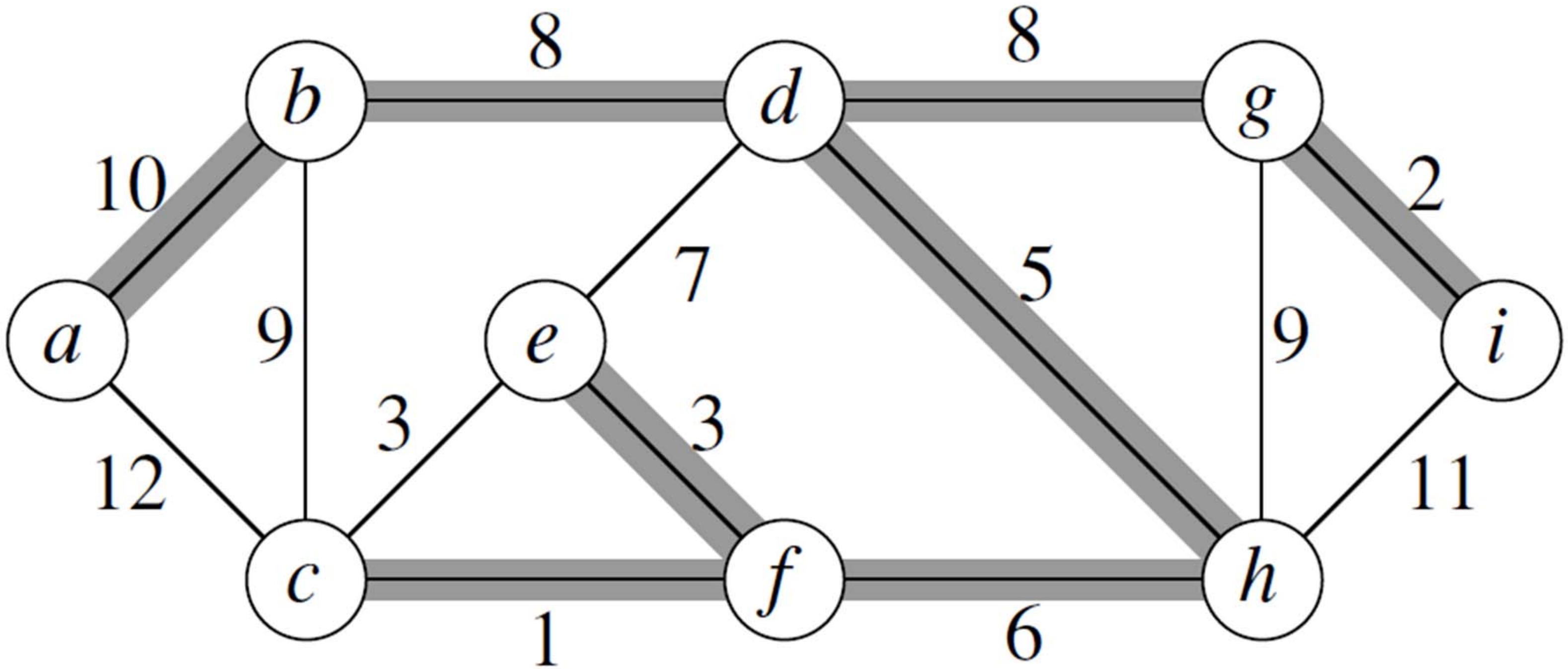
- Undirected graph $G = (V, E)$.
- ***Weight*** $w(u, v)$ on each edge $(u, v) \in E$.
- Find $T \subseteq E$ such that

  1. T connects all vertices (T is a ***spanning tree***), and
  2. $w(T) = \sum_{(u,v)\in T} w(u, v)$ is minimized.

A spanning tree whose weight is minimum over all spanning trees is called a ***minimum spanning tree***, or ***MST***.

Overview

Example of such a graph [edges in MST are shaded]



In this example, there is more than one MST. Replace edge *(e, f)* by *(c, e)*. Get a different spanning tree with the same weight.

Growing a minimum spanning tree

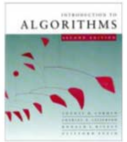
Some properties of an MST:

- It has $|V| - 1$ edges.
- It has no cycles.
- It might not be unique.

Building up the solution

- We will build a set A of edges.
- Initially, A has no edges.
- As we add edges to A, maintain a loop invariant:

  Loop invariant: A is a subset of some MST.
- Add only edges that maintain the invariant. If A is a subset of some MST, an edge (u, v) is ***safe*** for A if and only if $A \cup \{(u, v)\}$ is also a subset of some MST. So we will add only safe edges.



Growing a minimum spanning tree

Generic MST algorithm

GENERIC-MST*(G,w)*
$A \leftarrow \emptyset$
while A is not a spanning tree *(i.e., while nodes are not completely connected)*
  do find an edge *(u, v)* that is safe for A
  $A \leftarrow A \cup \{(u, v)\}$
return A

Use the loop invariant to show that this generic algorithm works.

Initialization: The empty set trivially satisfies the loop invariant.

Maintenance: Since we add only safe edges, A remains a subset of some MST.

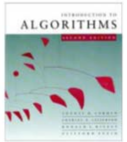
Termination: All edges added to A are in an MST, so when we stop, A is a spanning tree that is also a MST.

Growing a minimum spanning tree

Finding a safe edge

Some definitions: Let $S \subset V$ and $A \subseteq E$.

- A ***cut*** *$(S, V - S)$* is a partition of vertices into disjoint sets S and $V - S$.
- Edge $(u, v) \in E$ ***crosses*** cut $(S, V - S)$ if one endpoint is in S and the other is in $V - S$.
- A cut ***respects*** A if and only if no edge in A crosses the cut.
- An edge is a ***light edge*** crossing a cut if and only if its weight is minimum over all edges crossing the cut. For a given cut, there can be > 1 light edge crossing it.



Growing a minimum spanning tree

Theorem

Let A be a subset of some MST, $(S, V - S)$ be a cut that respects A, and (u, v) be a light edge crossing $(S, V - S)$. Then (u, v) is safe for A.

Proof Let T be a MST that includes A.

If T contains (u, v), done.

So now assume that T does not contain (u, v). We'll construct a different MST T' that includes $A \cup \{(u, v)\}$.

Growing a minimum spanning tree

Recall: a tree has unique path between each pair of vertices.

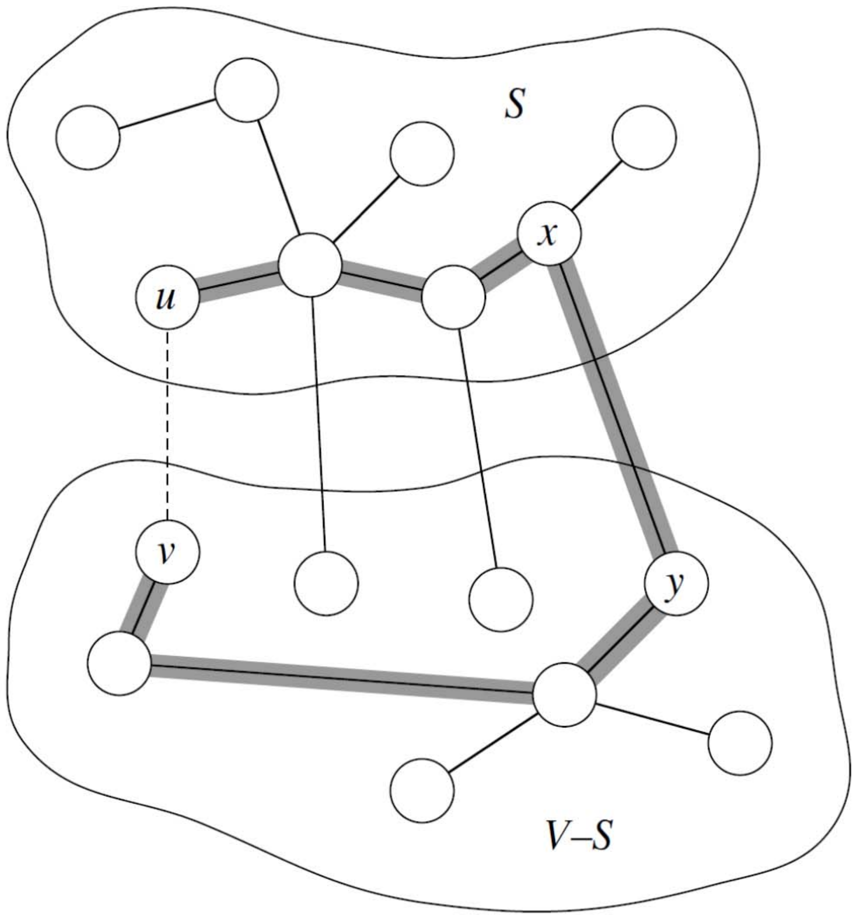
Since T is a MST, it contains a unique path p between u and v.

Path p must cross the cut $(S, V - S)$ at least once. Let (x, y) be an edge of p that crosses the cut.

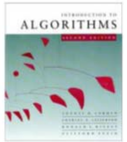
From how we chose (u, v), must have $w(u, v) \leq w(x, y)$.

(Because of given fact that (u, v) is a light edge)

Growing a minimum spanning tree



[Except for the dashed edge (u, v), all edges shown are in T. A is some subset of the edges of T, but A cannot contain any edges that cross the cut $(S, V - S)$, since this cut respects A. Shaded edges are the path p.]



Growing a minimum spanning tree

Since the cut respects A, edge (x, y) is not in A.
To form T' from T:

- Remove (x, y). Breaks T into two components.
- Add (u, v). Reconnects.

So $T' = T - \{(x, y)\} \cup \{(u, v)\}$.
T' is a spanning tree.
$w(T') = w(T) - w(x, y) + w(u, v)$
$\leq w(T)$,
since $w(u, v) \leq w(x, y)$. Since T' is a spanning tree, $w(T') \leq w(T)$, and T is a MST, then T' must be a MST.

Need to show that (u, v) is safe for A:

- $A \subseteq T$ and $(x, y) \notin A \Rightarrow A \subseteq T'$.
- $A \cup \{(u, v)\} \subseteq T'$.
- Since T' is a MST, (u, v) is safe for A. (theorem)

Growing a minimum spanning tree

So, in GENERIC-MST:

- A is a forest containing connected components. Initially, each component is a single vertex.
- Any safe edge merges two of these components into one. Each component is a tree.
- Since a MST has exactly $|V| - 1$ edges, the **for** loop iterates $|V| - 1$ times.

Equivalently, after adding $|V|-1$ safe edges, we're down to just one component (which is MST).

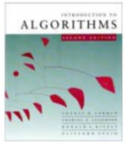
Growing a minimum spanning tree

Corollary

If $C = (V_C, E_C)$ is a connected component in the forest $G_A = (V, A)$ and (u, v) is a light edge connecting C to some other component in G_A (i.e., (u, v) is a light edge crossing the cut $(V_C, V - V_C)$), then (u, v) is safe for A.

Proof Set $S = V_C$ in the theorem. (corollary)

This naturally leads to the algorithm called Kruskal's algorithm to solve the minimum-spanning-tree problem.



Kruskal's algorithm

Kruskal's algorithm

$G = (V, E)$ is a connected, undirected, weighted graph. $w : E \rightarrow \mathbf{R}$.

- Starts with each vertex being its own component.
- Repeatedly merges two components into one by choosing the light edge that connects them (i.e., the light edge crossing the cut between them).
- Scans the set of edges in monotonically increasing order by weight.
- Uses a disjoint-set data structure to determine whether an edge connects vertices in different components.
- There can be more than one connected components during merge process.

Kruskal's algorithm

KRUSKAL(V, E, w)

$A \leftarrow \emptyset$

for each vertex $v \in V$

  do MAKE-SET(v)

sort E into nondecreasing order by weight w

for each (u, v) taken from the sorted list

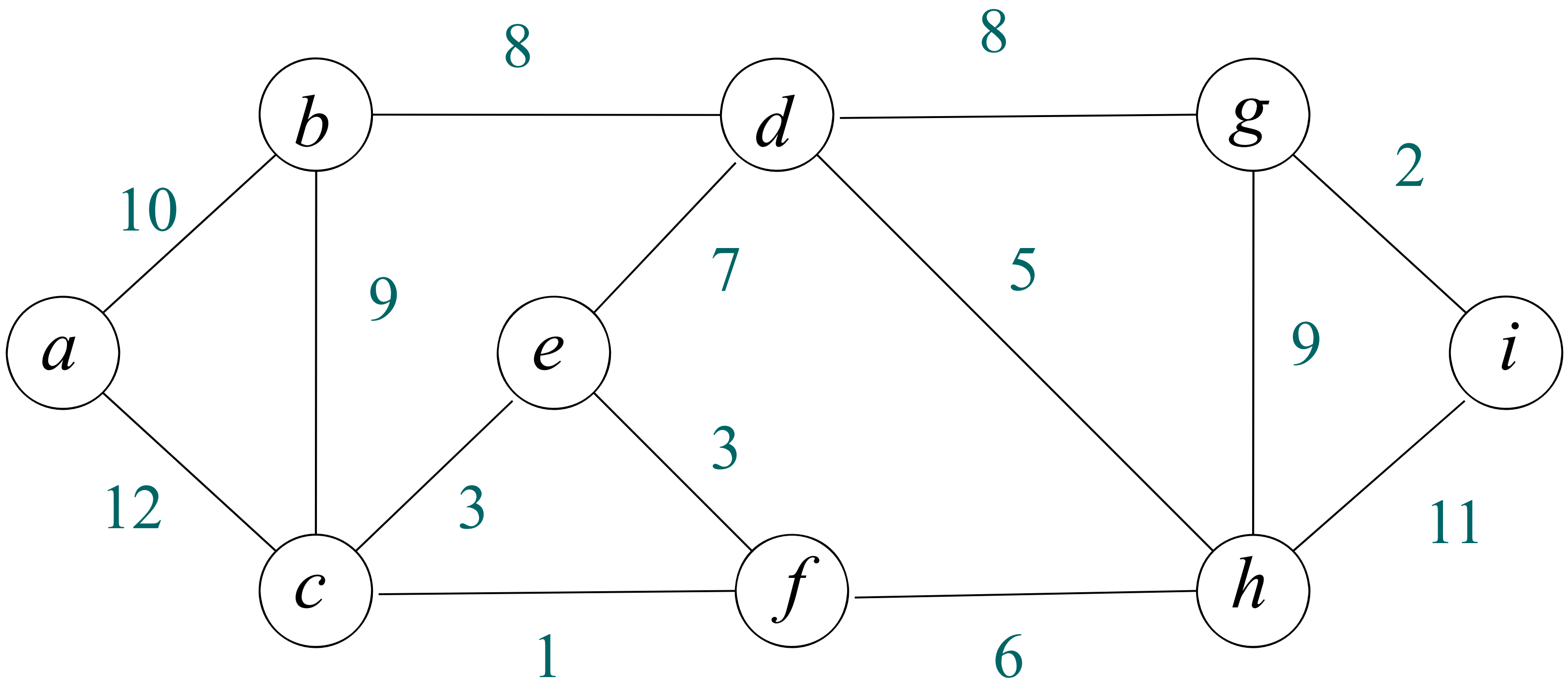
  do if FIND-SET(u) $\neq$ FIND-SET(v)

    then $A \leftarrow A \cup \{(u, v)\}$

      UNION(u, v)

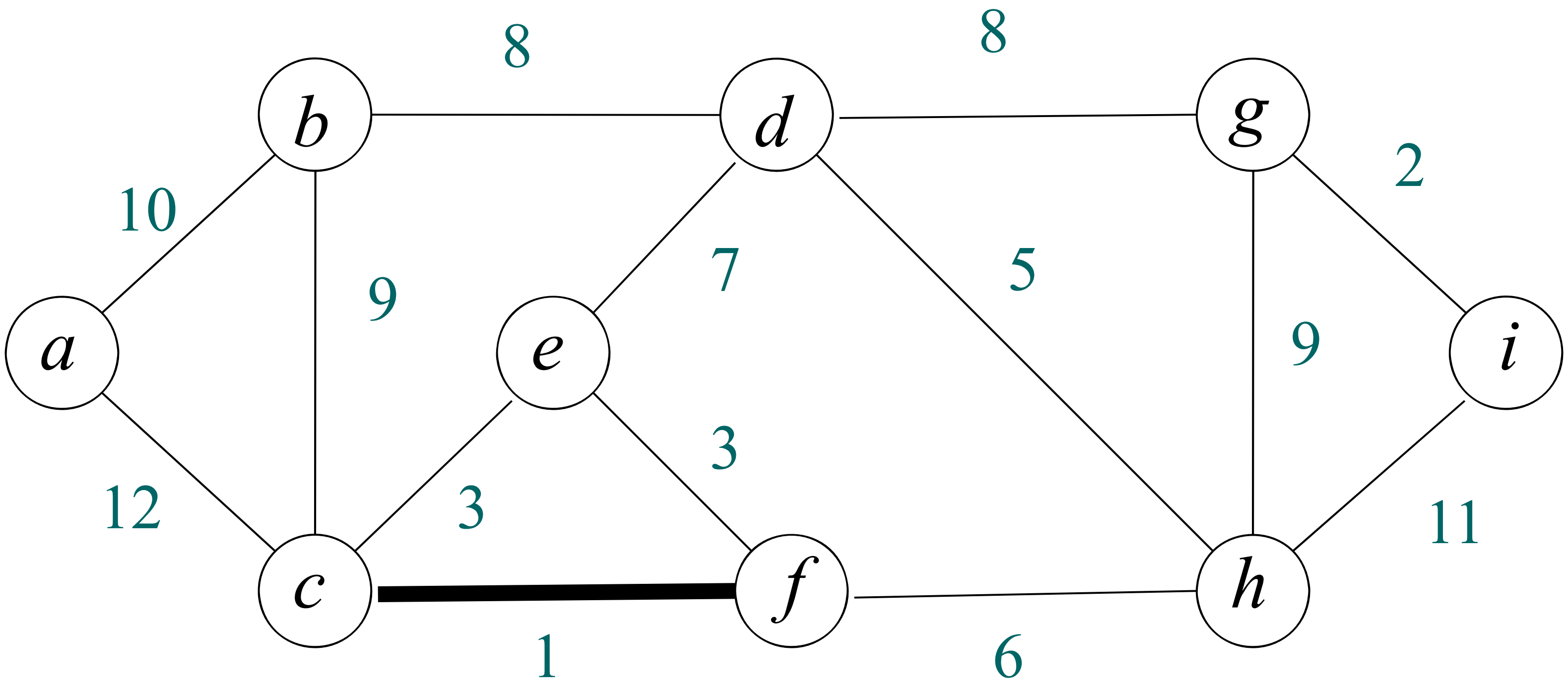
return A

Example of Kruskal’s algorithm



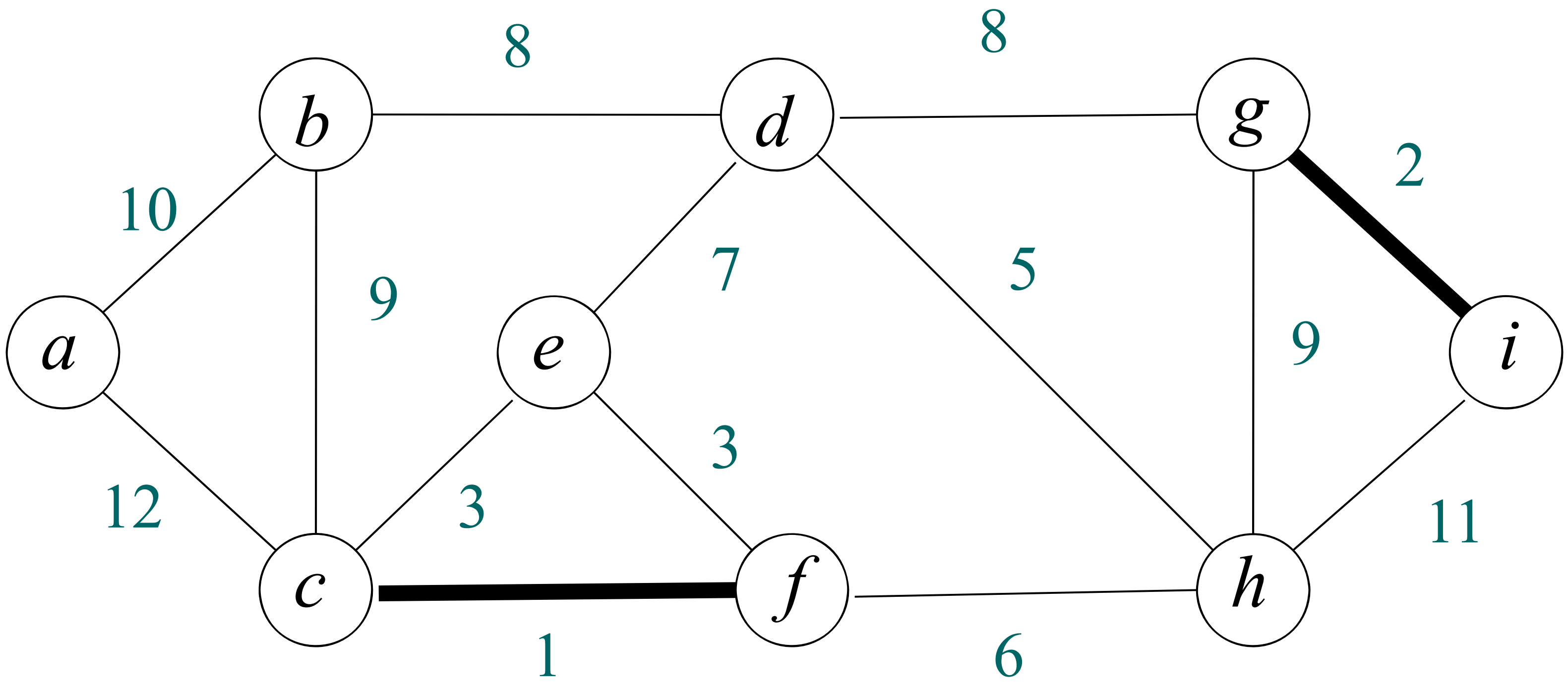
A = {}

Example of Kruskal's algorithm



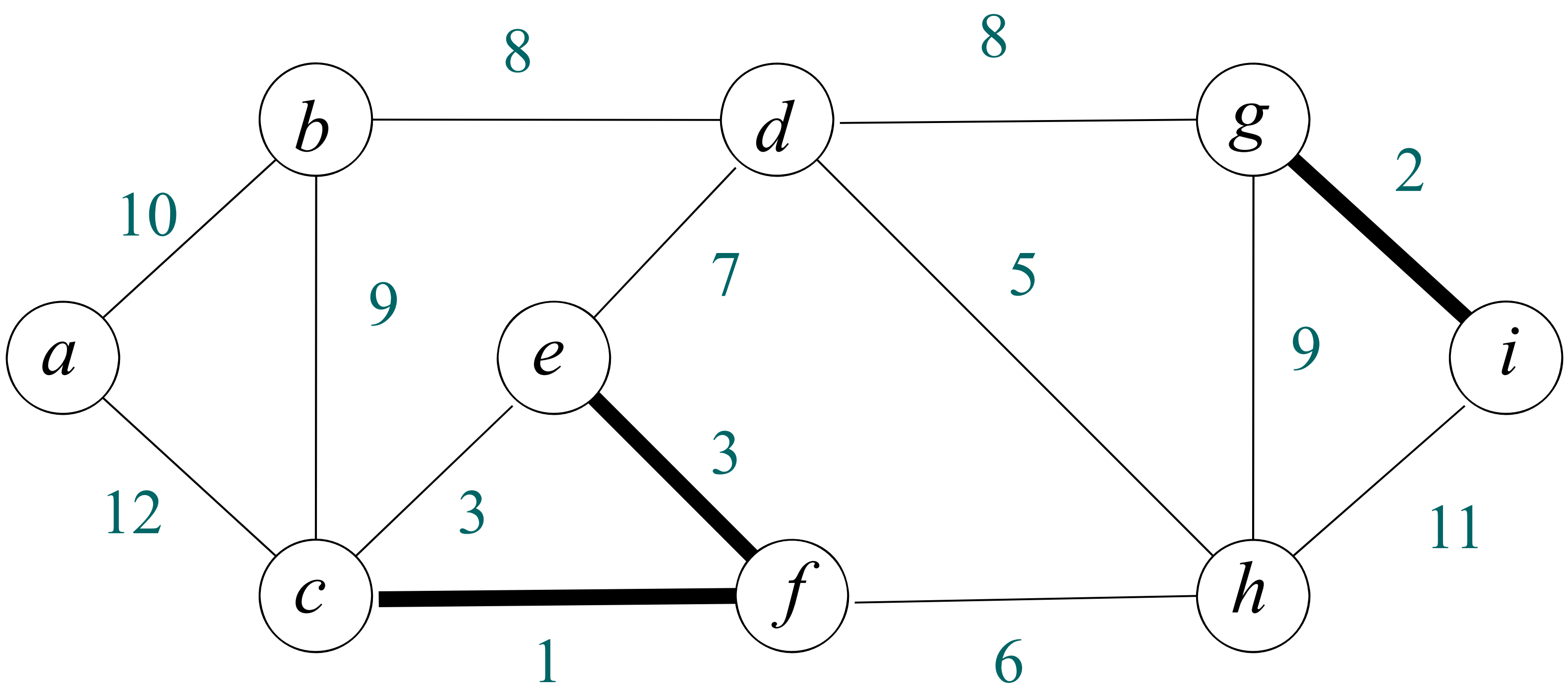
A = {(c, f)}

Example of Kruskal's algorithm

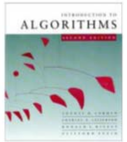


A = $\{(c, f), (g, i)\}$

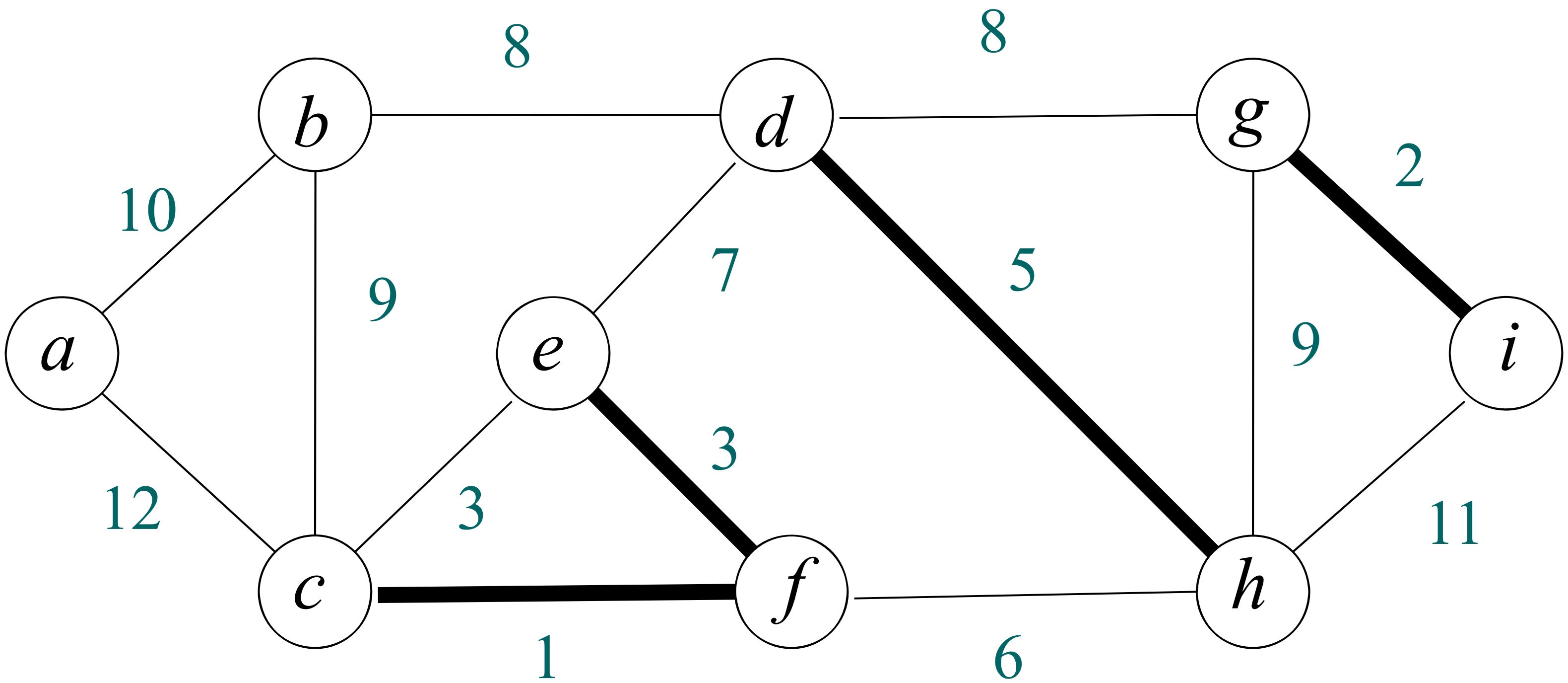
Example of Kruskal's algorithm



A = {(c, f), (g, i), (e, f)}

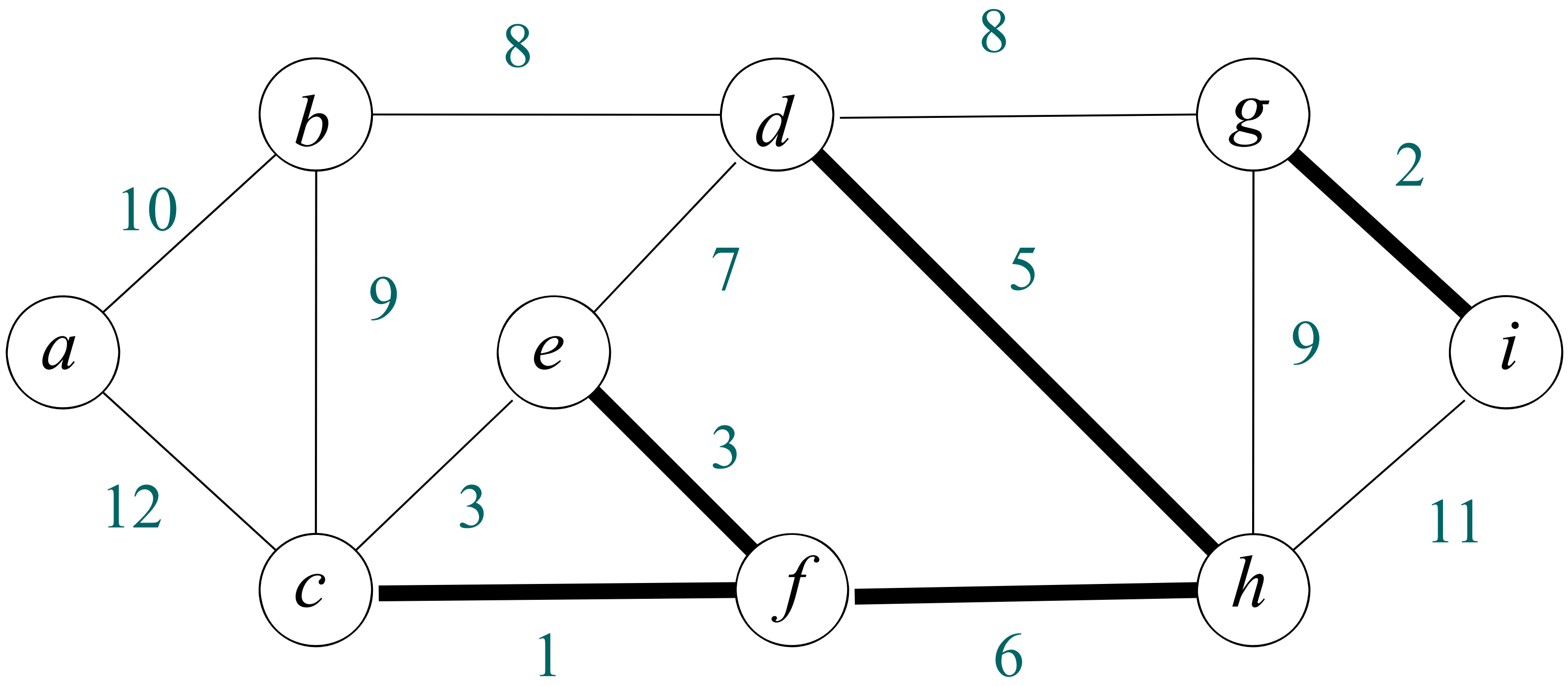


Example of Kruskal’s algorithm



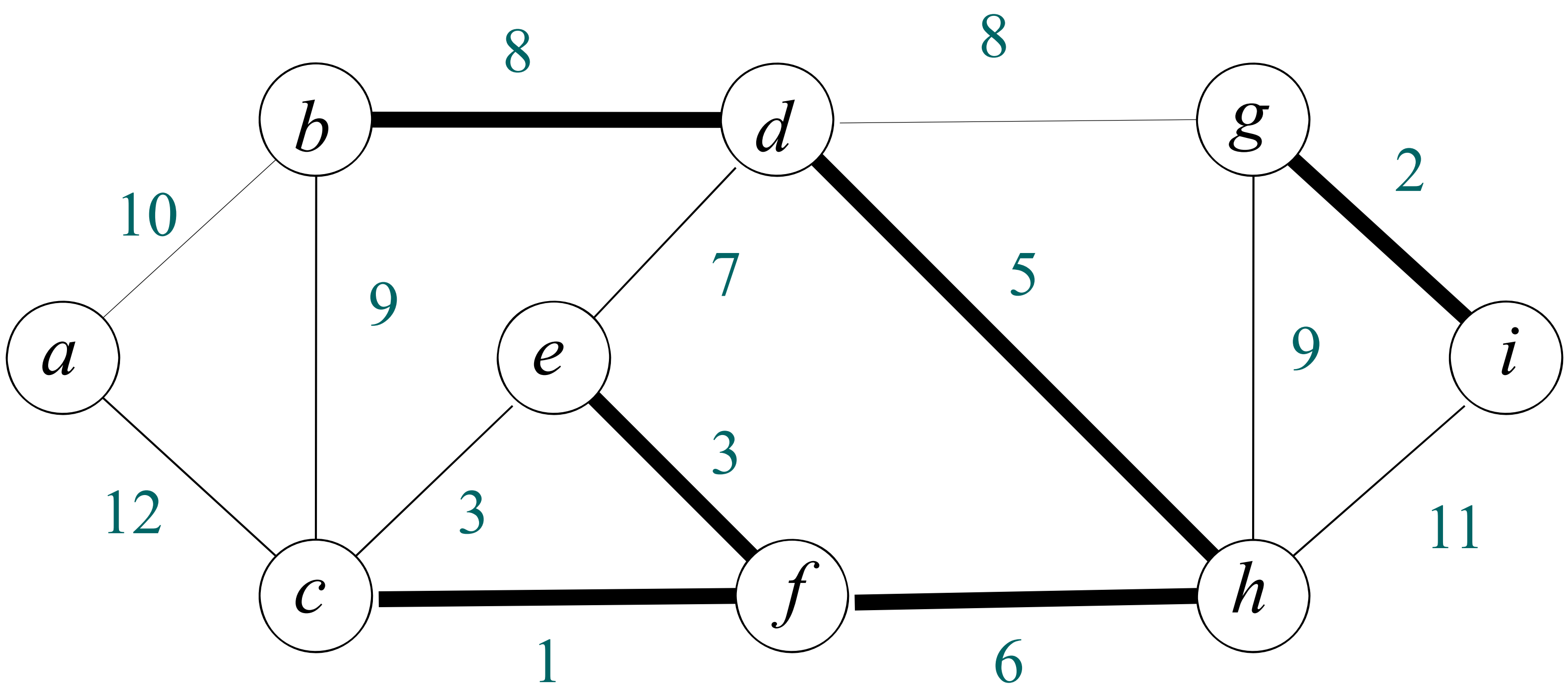
A = $\{(c, f), (g, i), (e, f), (d, h)\}$

Example of Kruskal's algorithm



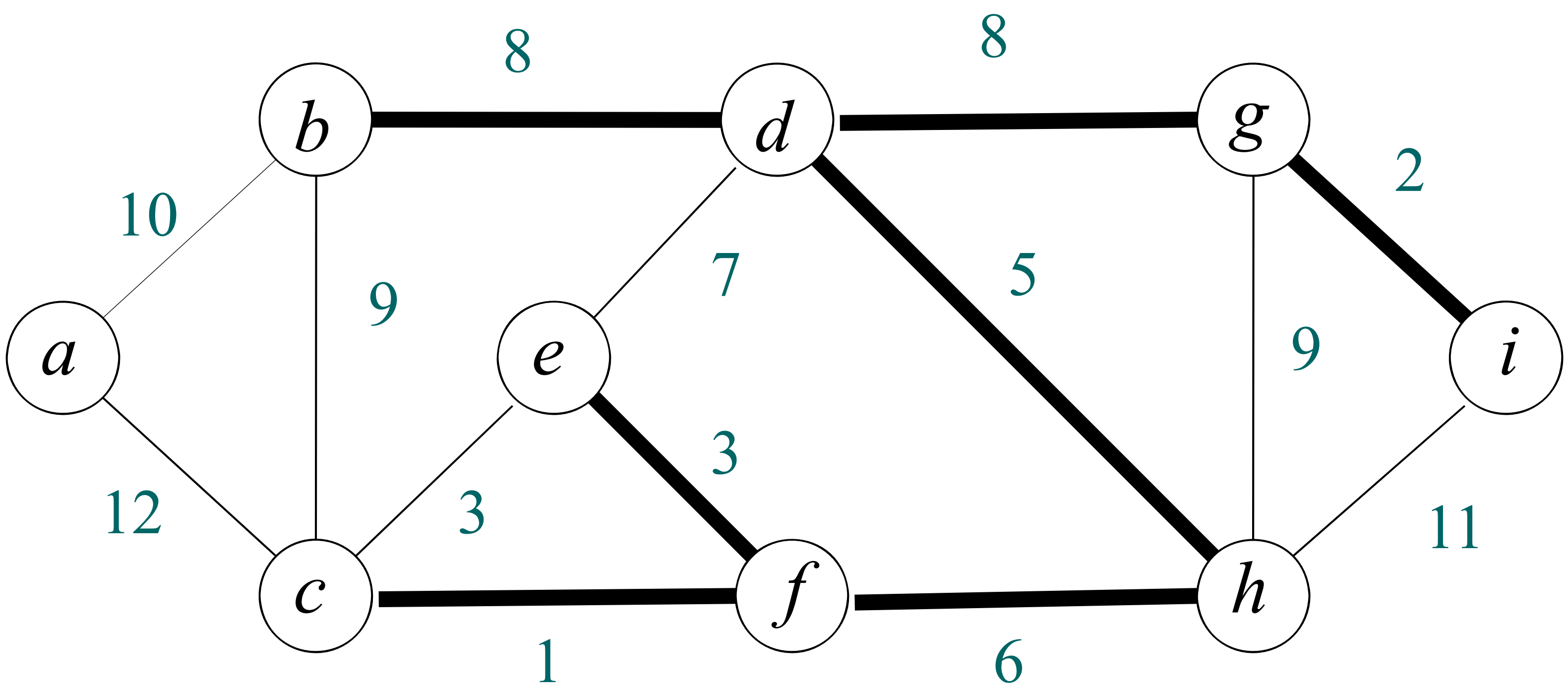
A = {(*c*, *f*), (*g*, *i*), (*e*, *f*), (*d*, *h*), (*f*, *h*)}

Example of Kruskal’s algorithm



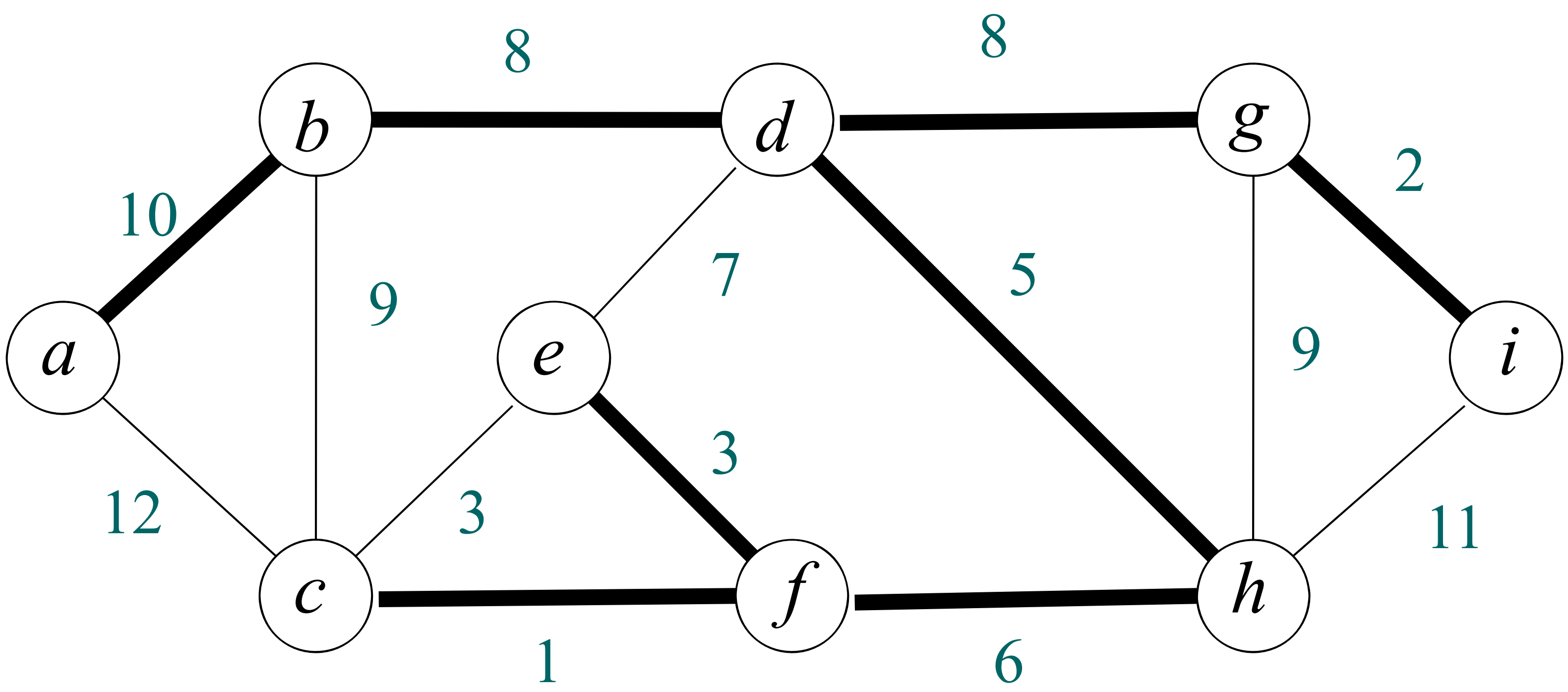
A = {(*c*, *f*), (*g*, *i*), (*e*, *f*), (*d*, *h*), (*f*, *h*), (*b*, *d*)}

Example of Kruskal’s algorithm



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

Example of Kruskal's algorithm



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

Kruskal's algorithm

Analysis

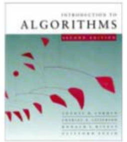
Initialize A: $O(1)$

First **for** loop: $|V|$ MAKE-SETs

Sort E: $O(E \lg E)$ *(sort edges using nlgn alg.)*

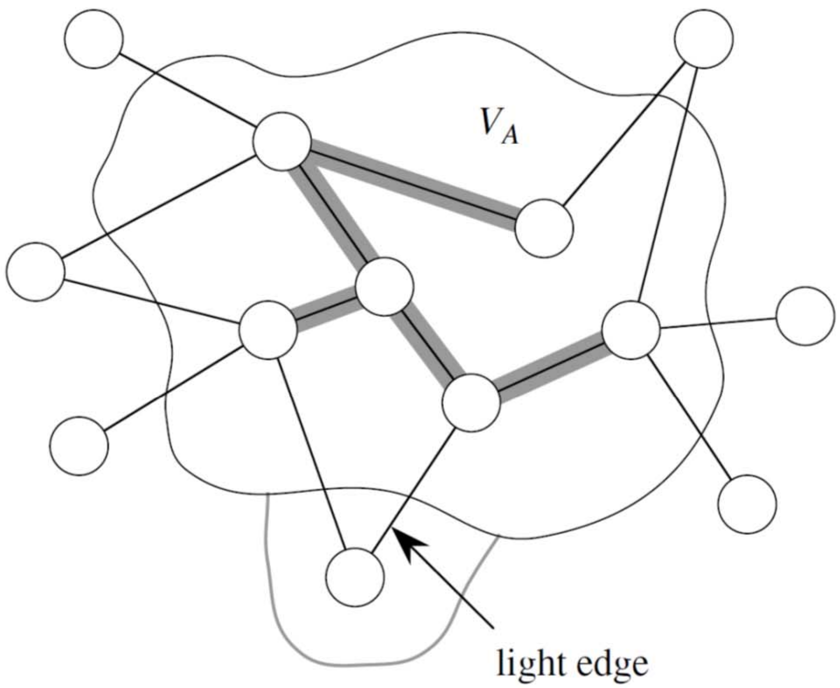
Second **for** loop: $O(E)$ FIND-SETs and UNIONs

Therefore, total time is $O(E \lg E)$.

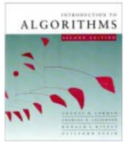


Prim's algorithm

- Builds one tree, so A is always a tree.
- Starts from an arbitrary "root" r.
- At each step, find a light edge crossing cut $(V_A, V - V_A)$, where V_A = vertices that A is incident on. Add this edge to A.



[Edges of A are shaded.]



Prim's algorithm

How to find the light edge quickly?

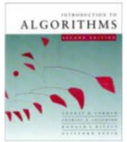
Use a priority queue Q:

- Each object, in Q, is a vertex in $V - V_A$.
- Key of v is minimum weight of any edge (u, v), where $u \in V_A$.
- Then the vertex returned by EXTRACT-MIN is v such that there exists $u \in V_A$ and (u, v) is light edge crossing $(V_A, V - V_A)$.
- Key of v is ∞ if v is not adjacent to any vertices in V_A.

Prim's algorithm

The edges of A will form a rooted tree with root r:

- r is given as an input to the algorithm, but it can be any vertex.
- Each vertex knows its parent in the tree by the attribute $\pi[v]$ = parent of v.
  $\pi[v]$ = NIL if $v = r$ or v has no parent.
- As algorithm progresses,
  $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$.
- At termination, $V_A = V \Rightarrow Q = \emptyset$,
  so MST is $A = \{(v, \pi[v]) : v \in V - \{r\}\}$.

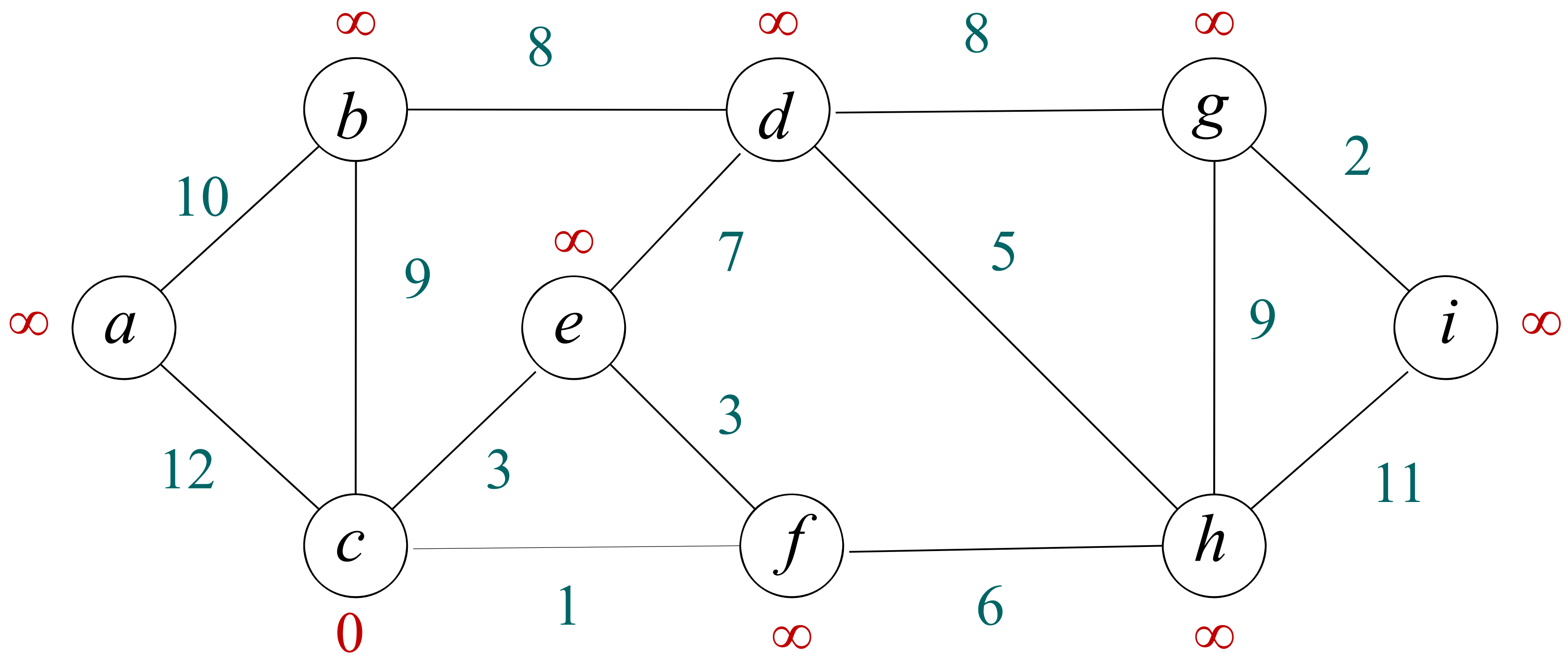


Prim's algorithm

```
PRIM(V, E, w, r)
Q ← ∅
for each u ∈ V
    do key[u] ← ∞
       π[u] ← NIL
       INSERT(Q, u)
DECREASE-KEY(Q, r, 0)      ▷ key[r] ← 0
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      DECREASE-KEY(Q, v, w(u, v))
```

$v \in Q$ *implies* v is not in V_A (i.e., v is not connected).

Example of Prim's algorithm

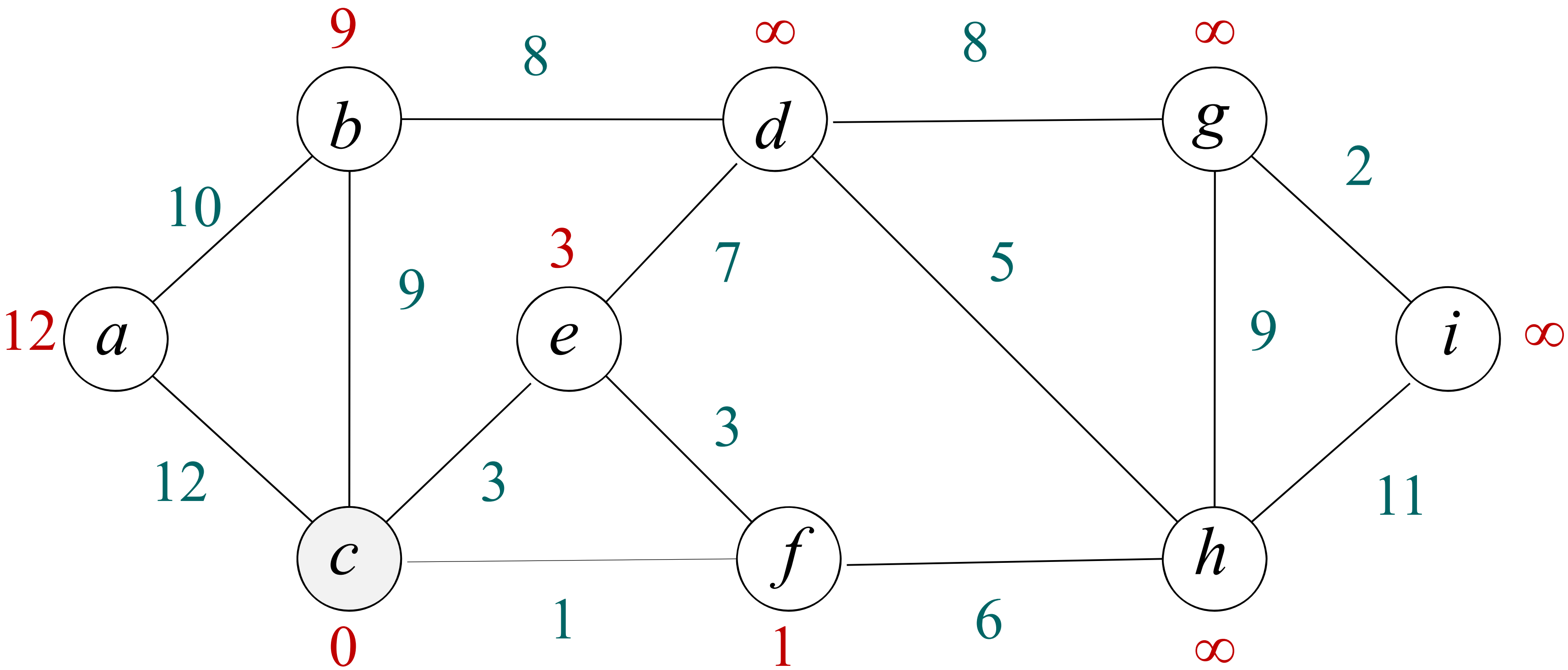


$Q = \{a, b, c, d, e, f, g, h, i\}$

$V_A = \{\}$

$A = \{\}$

Example of Prim's algorithm

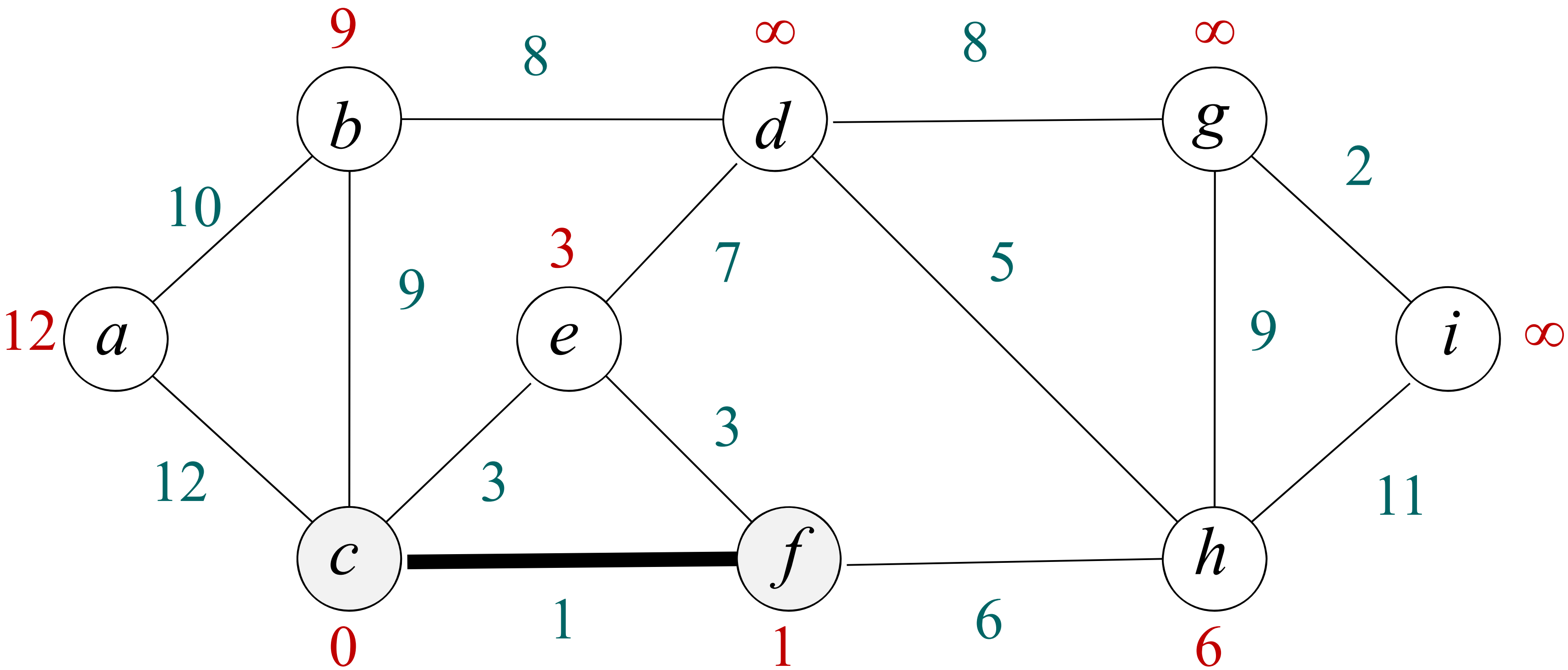


$Q = \{a, b, d, e, f, g, h, i\}$

$V_A = \{c\}$

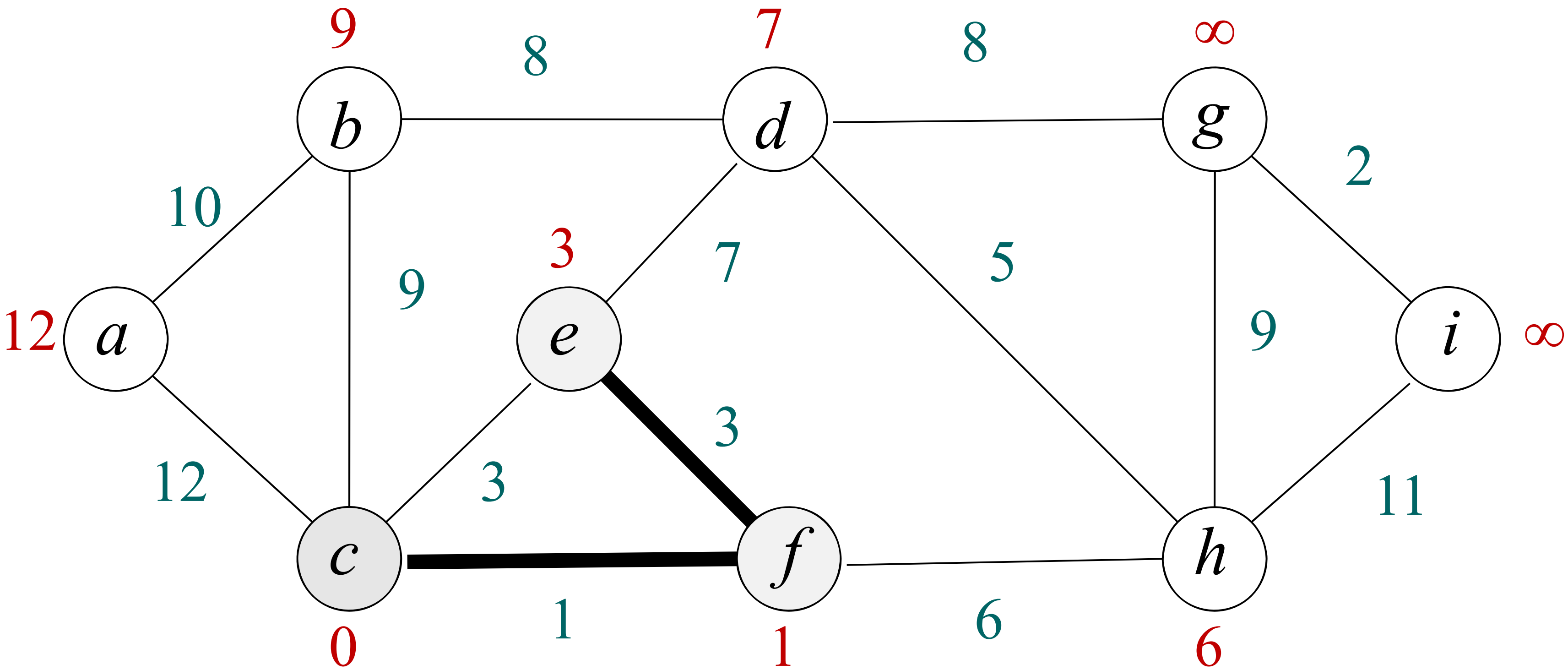
$A = \{\}$

Example of Prim's algorithm



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

Example of Prim's algorithm

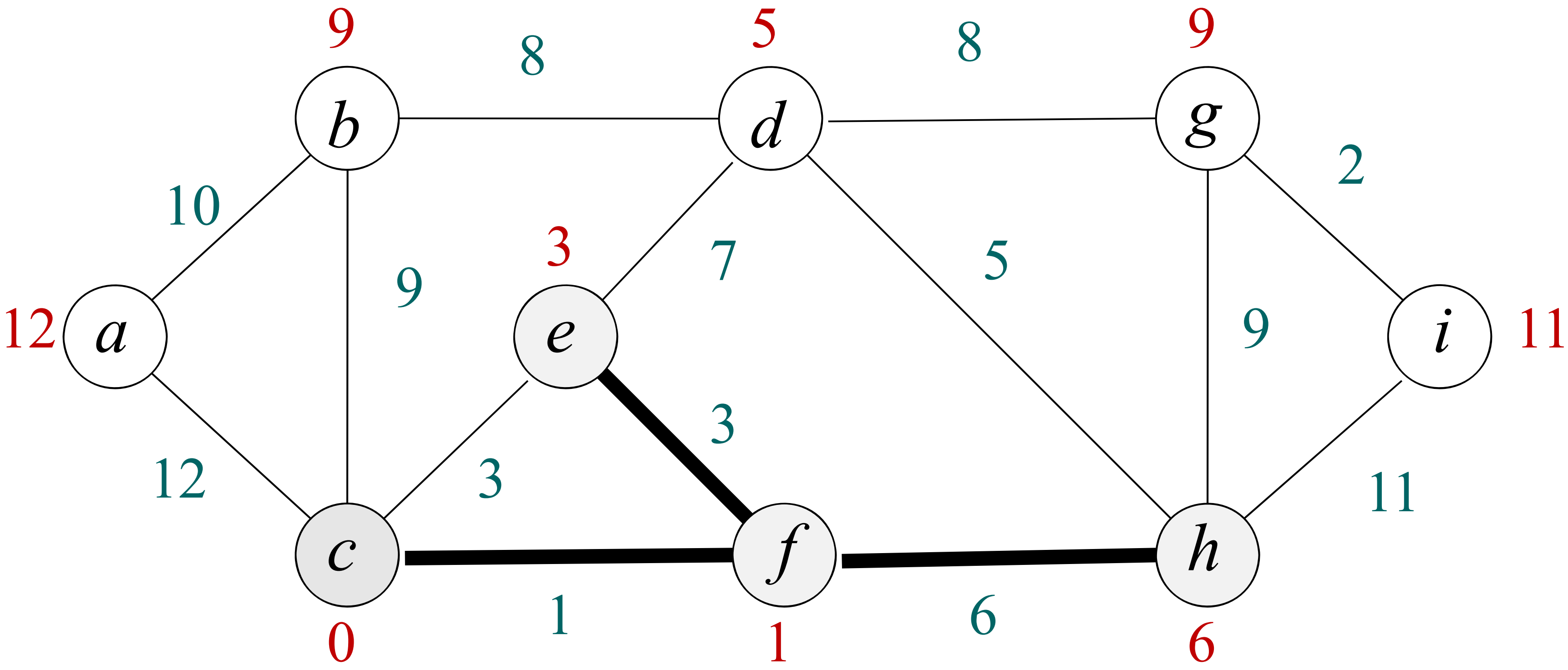


Q = $\{a, b, d, g, h, i\}$

$V_A = \{c, f, e\}$

A = $\{(f, c), (e, f)\}$

Example of Prim’s algorithm

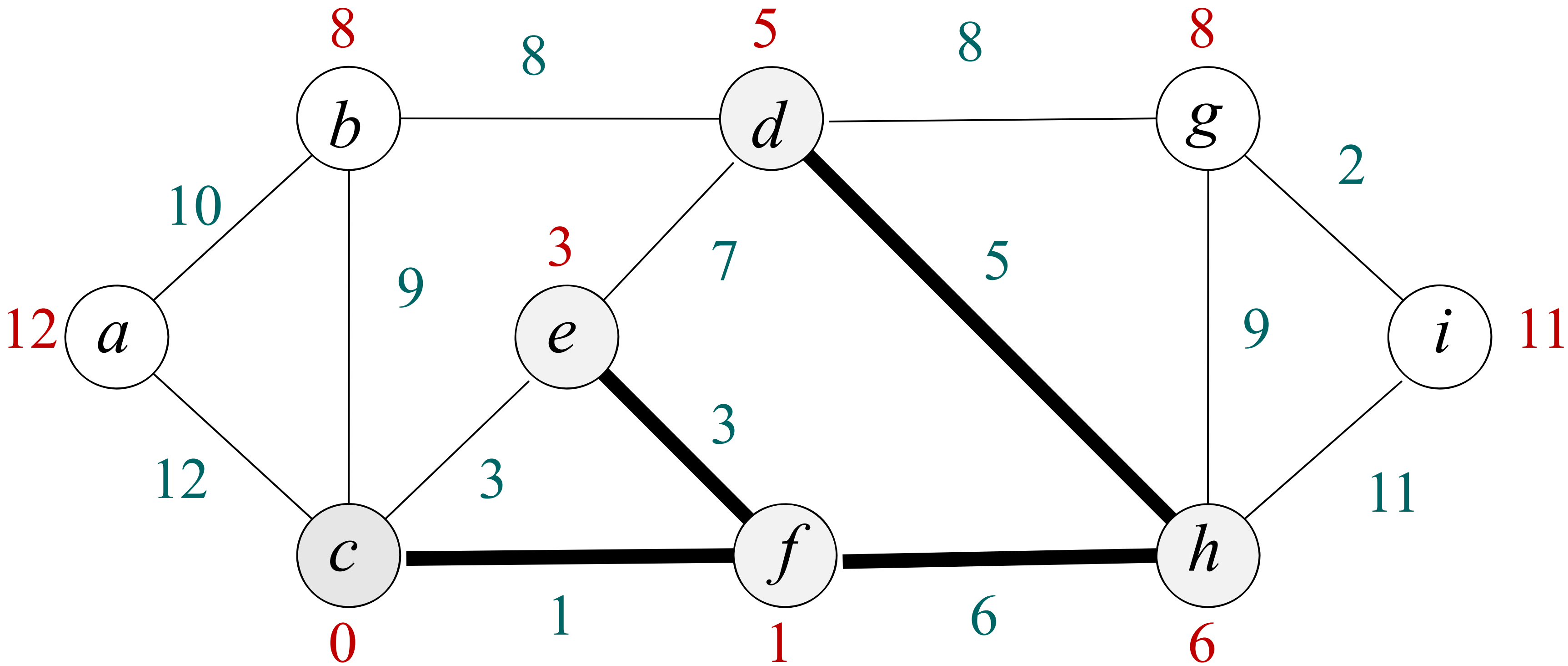


Q = $\{a, b, d, g, i\}$

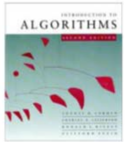
$V_A = \{c, f, e, h\}$

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

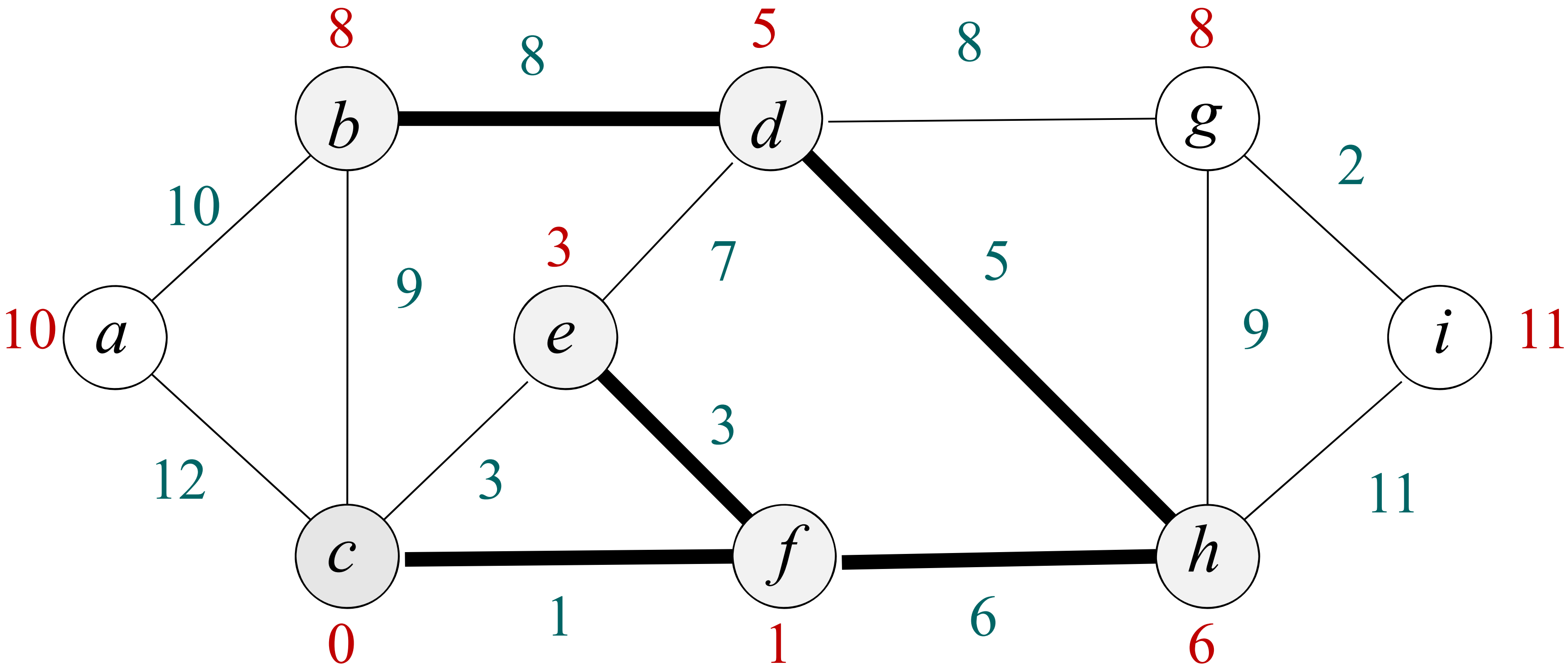
Example of Prim's algorithm



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



Example of Prim's algorithm

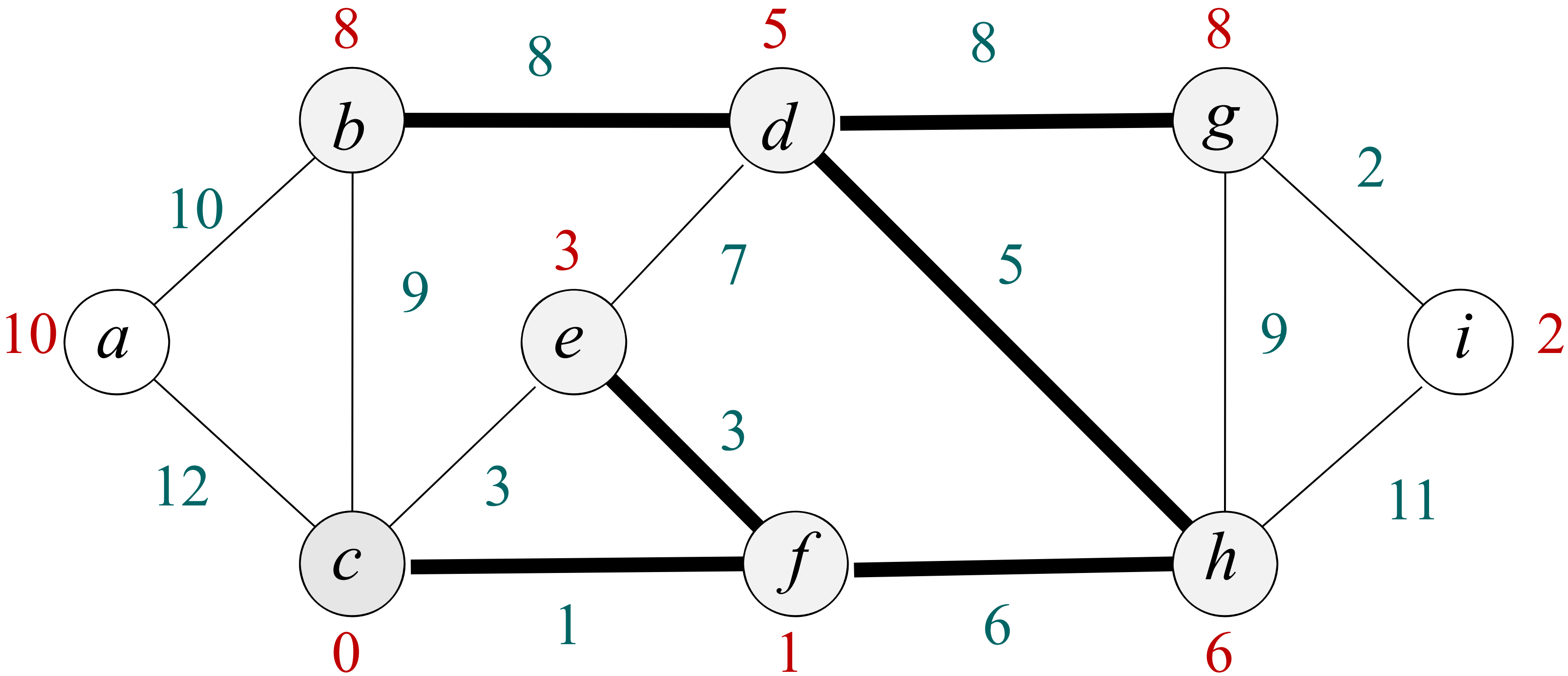


$Q = \{a, g, i\}$

$V_A = \{c, f, e, h, d, b\}$

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

Example of Prim's algorithm

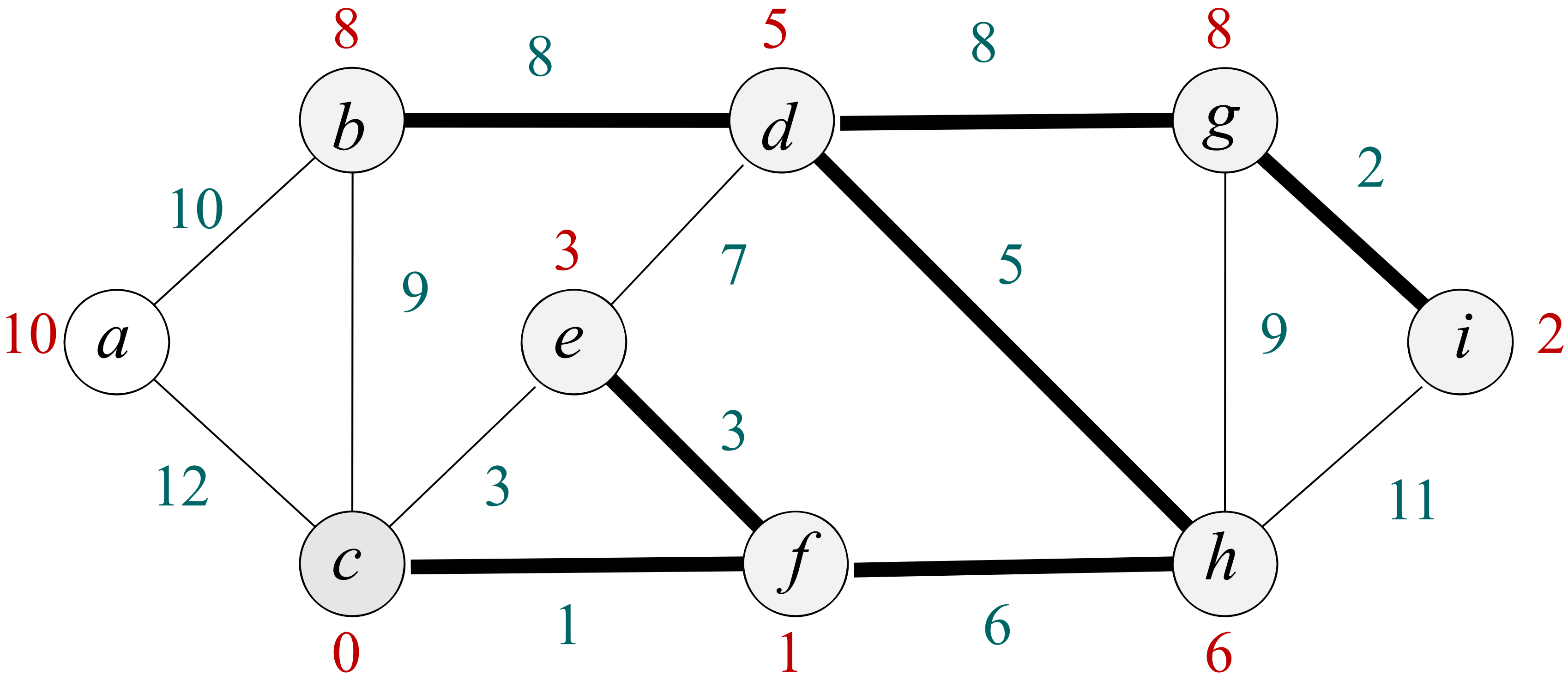


Q = $\{a, i\}$

$V_A = \{c, f, e, h, d, b, g\}$

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

Example of Prim’s algorithm

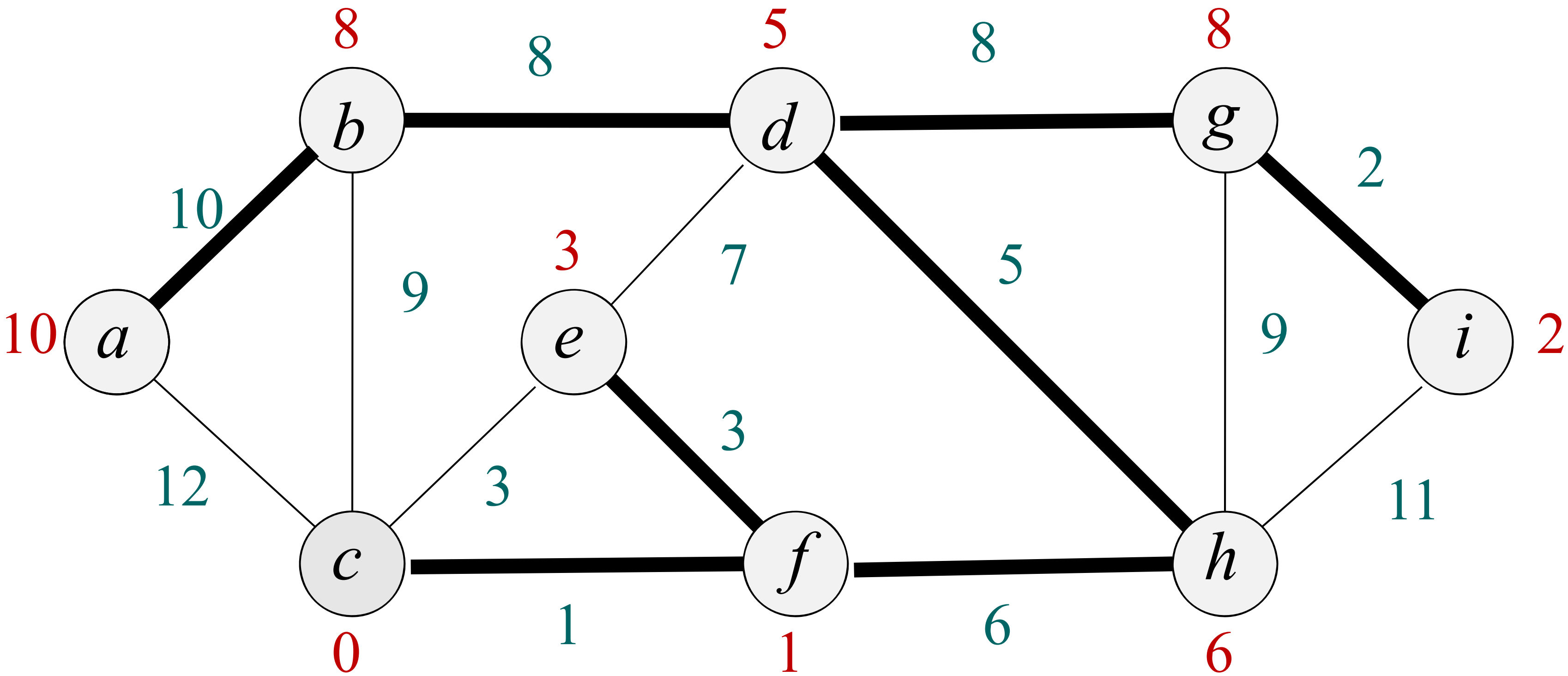


$Q = \{a\}$

$V_A = \{c, f, e, h, d, b, g, i\}$

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

Example of Prim's algorithm



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

Prim's algorithm

Analysis

Depends on how the priority queue is implemented:

- Suppose Q is a binary heap.

  Initialize Q and first **for** loop: $O(V \lg V)$

  Decrease key of r: $O(\lg V)$

  while loop: $|V|$ EXTRACT-MIN calls $\Rightarrow O(V \lg V)$

  $\leq |E|$ DECREASE-KEY calls $\Rightarrow O(E \lg V)$

  ($\leq |E|$, since all the edge-weights can be used in DECREASE-KEY calls or some of them can be already smaller than ***w(u, v)*** being considered.)

  Total: $O(E \lg V)$