Introduction to Algorithms

Chapter 23 : Minimum Spanning Trees

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2024

Outline of Topics

Basic definitions and properties

In this chapter, we shall examine two algorithms for solving the minimum spanning-tree problem: Kruskal's algorithm and Prim's algorithm.

- Section 23.1 introduces a "generic" minimum-spanning-tree method that grows a spanning tree by adding one edge at a time.
- Section 23.2 gives two algorithms that implement the generic method.

Basic definitions and properties

Definition 1:

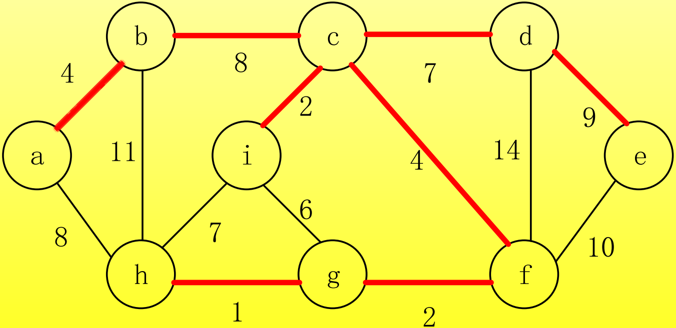
Given a connected, undirected graph $G = (V, E)$, for each edge $(u, v) \in E$, having a weight $w(u, v)$.

We wish to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight is minimized.

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

Since T is acyclic and connects all of the vertices, it must form a tree, which we call a spanning tree since it "spans" the graph G. We call the problem of determining the tree T the minimum-spanning-tree problem(MST).

Example of a connected graph and MST



Greedy Method for MST

The two algorithms (Kruskal's Algorithm and Prim's Algorithm) we consider in this chapter run in time $O(|E|log|V|)$ using **a greedy approach** to the problem.

Greedy strategy:

Grows the minimum spanning tree one edge at a time and manages a set of edges A, maintaining the following loop invariant:

Prior to each iteration, A is a subset of some minimum spanning tree.

Generic MST Algorithm

At each step we determine an edge (u,v) such that $A \cup (u,v)$ is still a subset of a MST and (u,v) is called a **safe edge** for A.

GENERIC-MST(G,w)

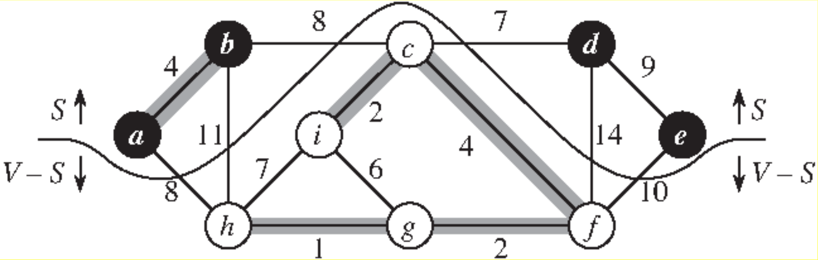
```
A = ∅
while A does not form a spanning tree do
    find an edge (u,v) that is safe for A
    A = A ∪ (u,v)
return A
```

Cut and Light Edge

Definition 2:

- A **cut** $(S, V-S)$ of an undirected graph $G = (V, E)$ is a partition of V.
- An edge $(u, v) \in E$ crosses the cut iff one of its endpoints is in S and the other is in $V - S$
- An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut
- We call a cut **respects** a set A of edges if no edge in A crosses the cut.

Example of Cut and Light Edge



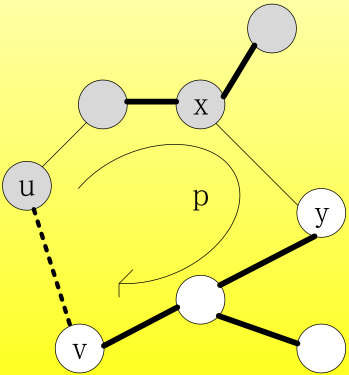
Recognize safe edges

Theorem 23.1:

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$.

Then, edge (u, v) is safe for A.

Recognize safe edges



Recognize safe edges

Proof:

Suppose that T is an MST containing A but not the light edge (u, v), then there is at least one edge on the path p that crosses the cut, say(x, y).

1.form a new spanning tree T':

Then (x, y) is not in A. Because p is the unique path from u to v in T, so removing (x, y) breaks T into two components. Adding (u, v) reconnects them to form a new spanning tree
$T' = (T - \{(x, y)\}) \cup \{(u, v)\}$.

Recognize safe edges

2. show that T' is MST:

Since (u,v) is a light edge crossing $(S, V-S)$ and (x,y) also crosses this cut,$w(u,v) \le w(x,y)$. Therefore,

$$\begin{aligned} w\left(T'\right) &= w(T) - w(x,y) + w(u,v) \\ &\le w(T) \end{aligned}$$

But T is a minimum spanning tree, so that $w(T) \le w(T')$.thus T' must be a minimum spanning tree also.

3. show that (u,v) is a safe edge for A:

We have $A \subseteq T'$, since $A \subseteq T$ and $(x,y) \notin A$,thus $A \cup \{(u,v)\} \subseteq T'$. Since T' is a MST, (u,v) is safe for A.

Corollary to Theorem 23.1

Corollary to theorem 23.1:

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, and let $C = (V_c, E_c)$ be a connected component (tree) in the forest $G_A = (V, A)$.

If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

Proof:

Since the cut $(V_c, V - V_c)$ respects A, and (u, v) is a light edge for this cut, (u, v) is safe for A.

Kruskal and Prim Algorithms

- In **Kruskal's algorithm**, the set A forms a forest. The safe edge added to A is always a least-weight edge in the graph that connects two distinct components
- In **Prim's algorithm**, the set A forms a single tree. The safe edge added to A is always a least-weight edge connecting the tree to a vertex not in the tree.

Kruskal's algorithm

Kruskal's algorithm finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of **the least weight**.

Let C_1 and C_2 denote the two trees that are connected by (u,v). Since (u,v) is a light edge, connecting C_1 to some other tree, (u,v) is a safe edge for C_1.(Corollary 23.2)

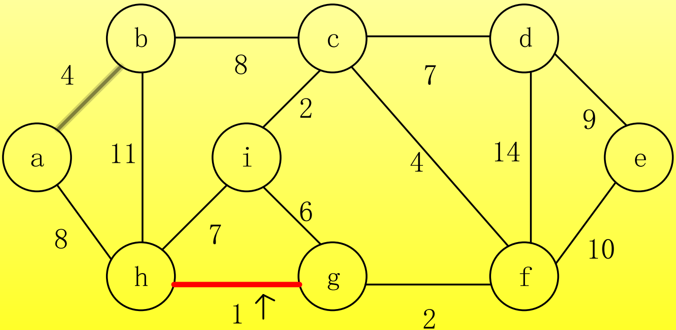
Simply speaking, at each step Kruskal's algorithm adds to the forest an edge of the least possible weight (greedy).

Kruskal's algorithm

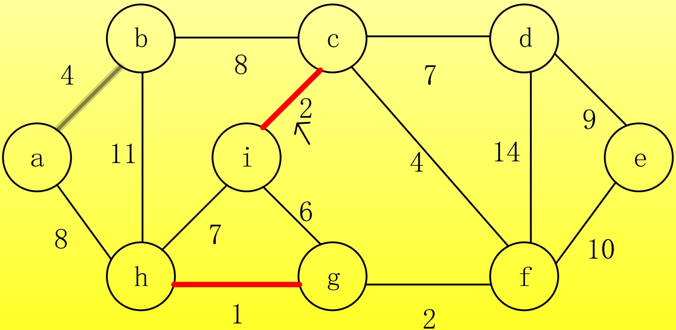
MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V do
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taking in nondecreasing order by weight w, do
    if FIND-SET(u) ≠ FIND-SET(v) then
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

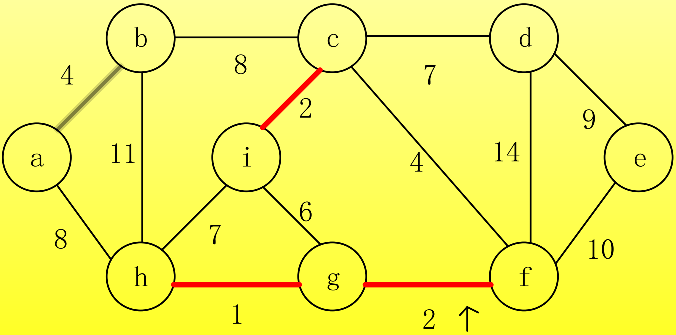
Kruskal's algorithm



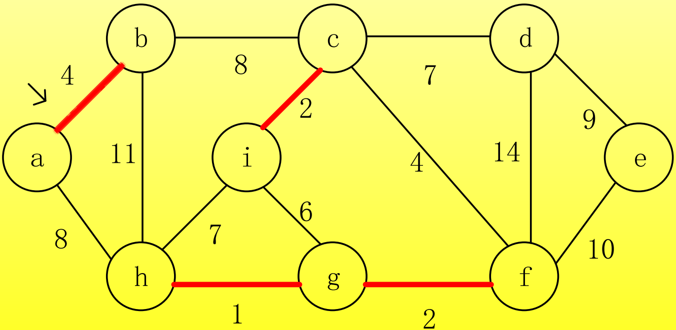
Kruskal's algorithm



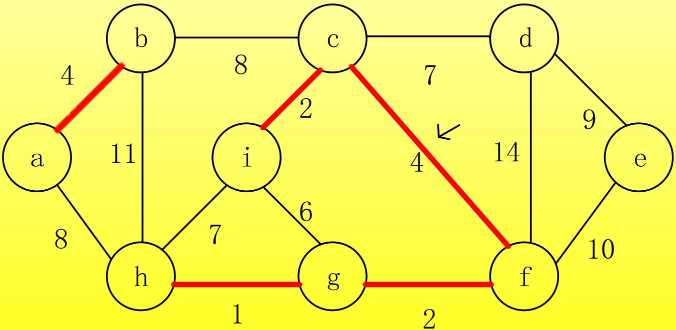
Kruskal's algorithm



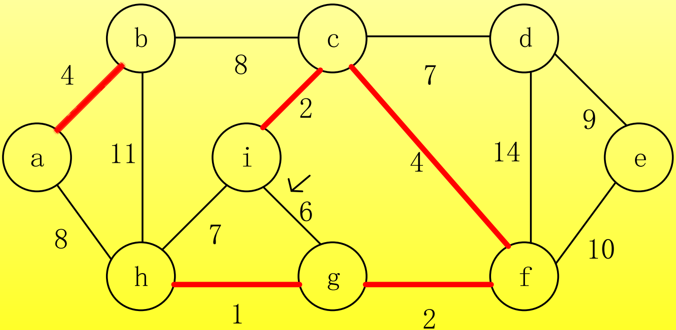
Kruskal's algorithm



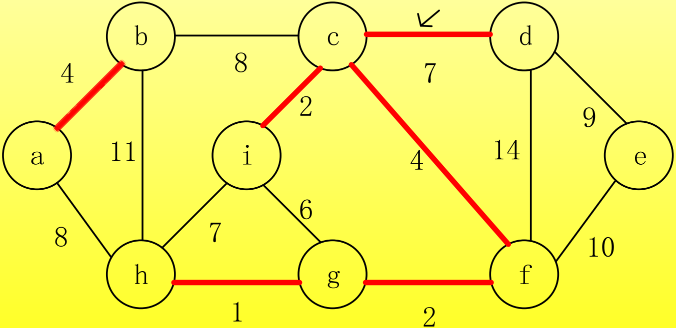
Kruskal's algorithm



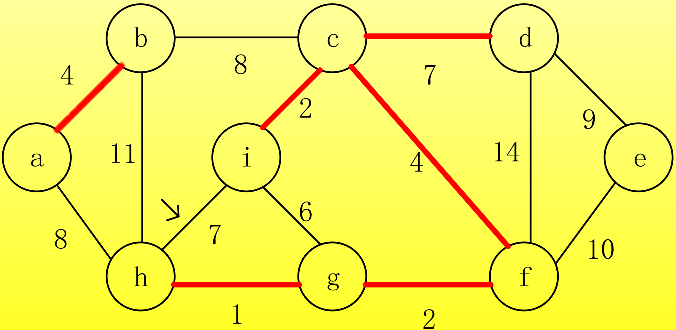
Kruskal's algorithm



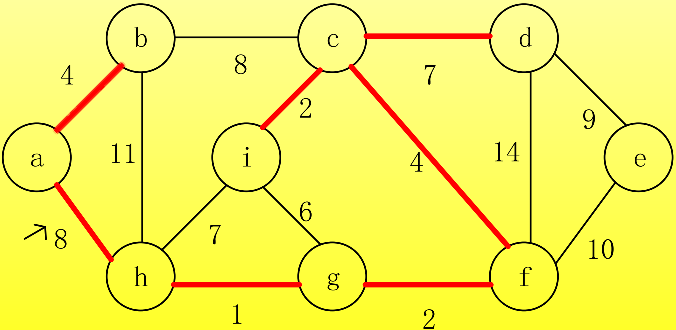
Kruskal's algorithm



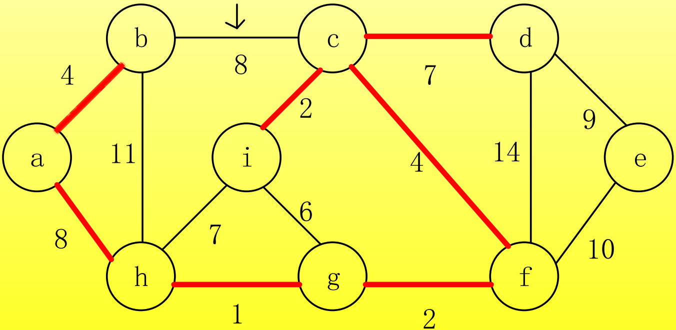
Kruskal's algorithm



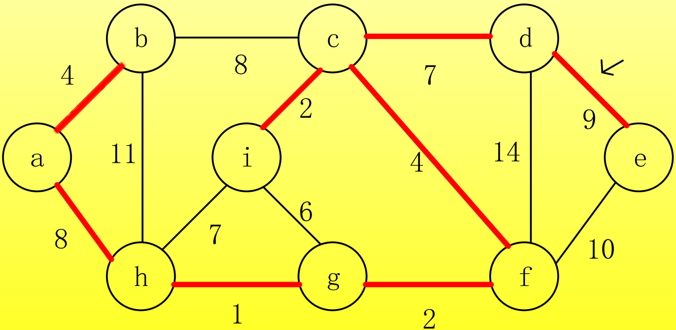
Kruskal's algorithm



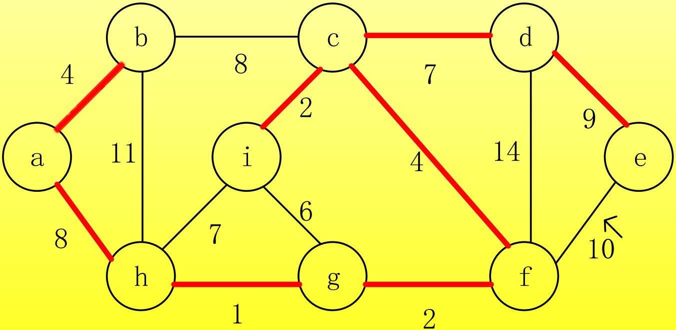
Kruskal's algorithm



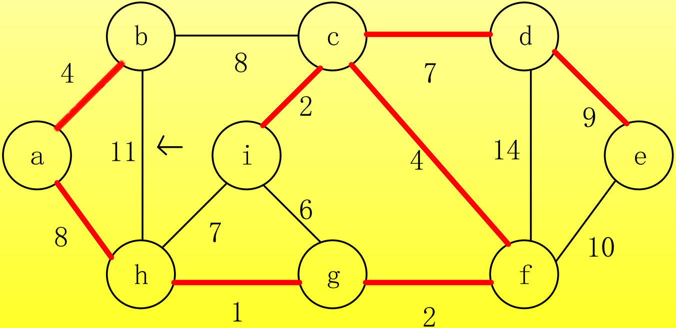
Kruskal's algorithm



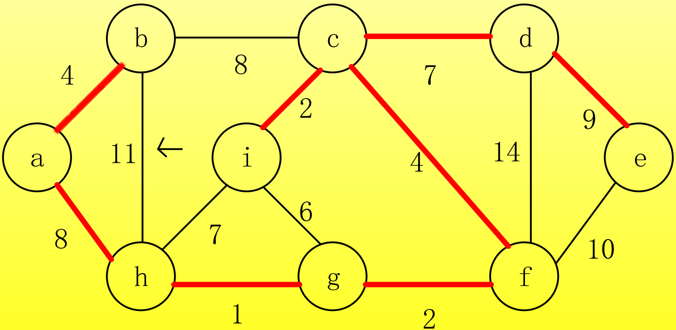
Kruskal's algorithm



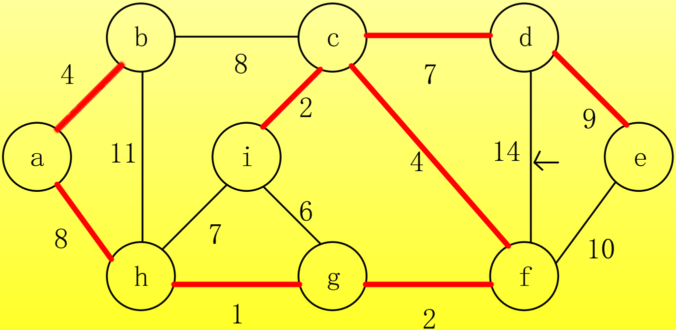
Kruskal's algorithm



Kruskal's algorithm



Kruskal's algorithm



Kruskal's algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V do
    MAKE-SET(v)        // O(V) MAKE-SET
sort the edges of G.E into nondecreasing order by weight w        // O(E log E)
for each edge (u, v) ∈ G.E, taking in nondecreasing order by weight w, do
    if FIND-SET(u) ≠ FIND-SET(v) then
        A = A ∪ {(u, v)}
        UNION(u, v)        //totally O(E) FIND-SET and UNION
return A
```

Kruskal's Algorithm Complexity

Complexity:

Assume that we use the disjoint-set-forest implementation with the union-by-rank and path-compression heuristics

Initializing the set A in line 1 takes$O(1)$ time

Sort the edges in line 4 is $O(E \lg E)$ time

The for loop of lines 5-–8 performs $O(E)$ FIND-SET and UNION operations on the disjoint-set forest

Along with the $|V|$ MAKE-SET operations, these take a total of $O((V+E)\alpha(V))$ time

Since $\alpha(|V|) = O(\lg V) = O(\lg E)$, the running time is $O(E \lg E)$

Since $|E| \leq |V|^2$, $\lg|E| = O(\lg V)$, the running time is $O(E \lg V)$

Prim's algorithm

Prim's algorithm operates much like Dijkstra's algorithm for finding shortest paths in a graph.

Prim's algorithm has the property that the edges in the set A **always form a single tree.**

Each step adds to the tree A a light edge that connects A to an isolated vertex – one on which no edge of A is incident.

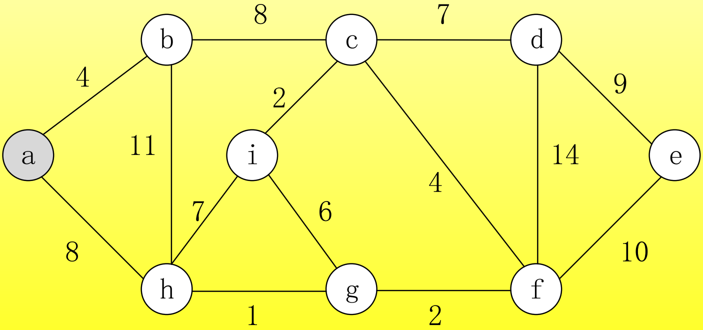
Simply speaking, at each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight (greedy).

Prim's algorithm

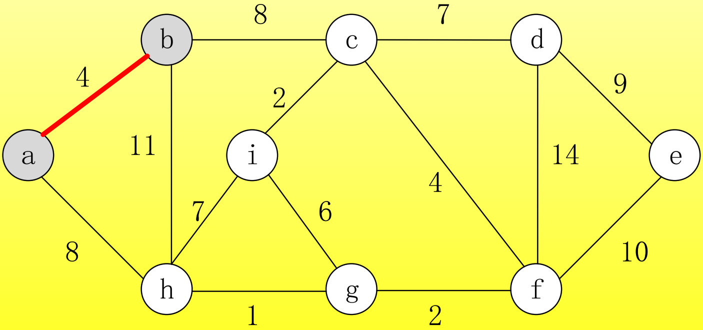
MST-PRIM(G, w, r)

for each vertex $u \in G.V$ **do**
  $u.key = \infty$; //$u.key$ stores the minimum weight of any edge connecting u to a vertex in the current tree
  $u.\pi = NIL$
$r.key = 0$
$Q = G.V$ // Q contains nodes not yet joining the tree
while $Q \neq \emptyset$ **do**
  u = EXTRACT-MIN(Q) //adding $(u.\pi, u)$ to the tree
  for each $v \in G.Adj[u]$ **do**
    if $v \in Q$ and $w(u,v) < v.key$ **then** //updating keys
      $v.\pi = u$
      $v.key = w(u,v)$

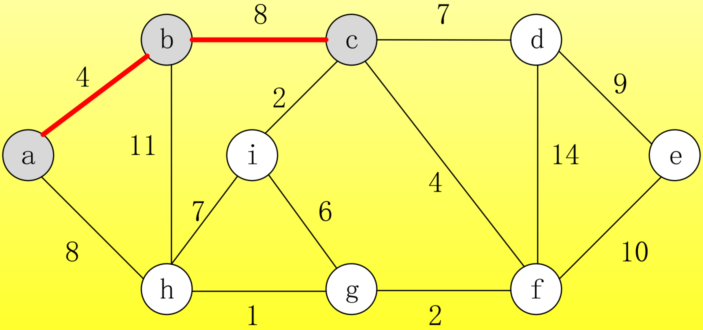
Prim's algorithm



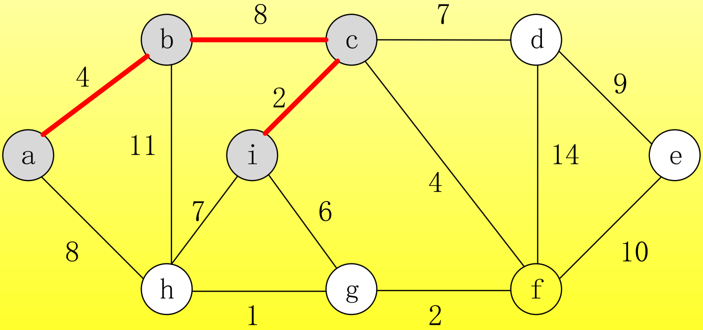
Prim's algorithm



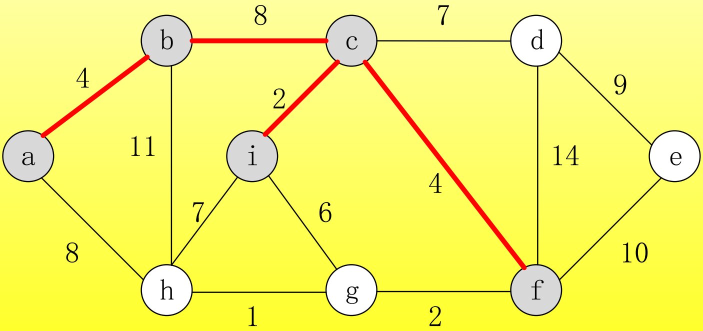
Prim'salgorithm



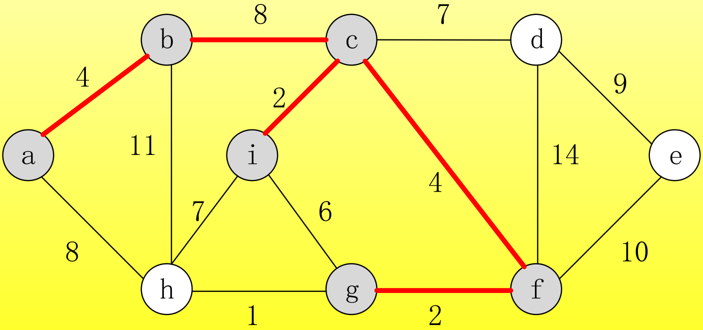
Prim's algorithm



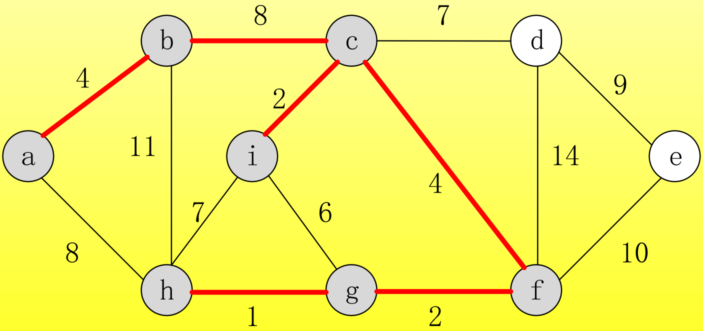
Prim's algorithm



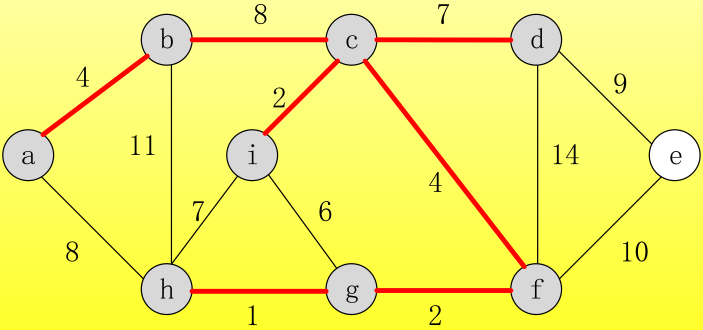
Prim's algorithm



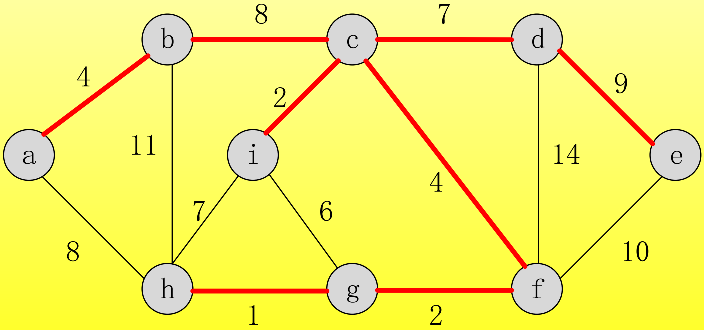
Prim's algorithm



Prim's algorithm



Prim's algorithm



Prim's algorithm

MST-PRIM(G, w, r)

for each vertex $u \in G.V$ **do**
    $u.key = \infty$
    $u.\pi = NIL$
$r.key = 0$
$Q = G.V$ //BUILD-MIN-HEAP, $O(V)$
while $Q \neq \emptyset$ **do** //V loops
    u = EXTRACT-MIN(Q) //$O(\log V)$ for each loop
    for each $v \in G.Adj[u]$ **do** // //2E loops totally
        if $v \in Q$ and $w(u, v) < v.key$ **then**
            $v.\pi = u$
            $v.key = w(u, v)$ //DECREASE-KEY

Prim's algorithm Complexity

Complexity:

Implement the min-priority queue Q as a binary min-heap:

Lines 1 - 5 : use the BUILD-MIN-HEAP to perform $O(V)$

The body of the while loop executes $|V|$ times, since each EXTRACT-MIN operation takes $O(\lg V)$ time, the total time if $O(VlgV)$ time. The for loop in lines 8 - 11 executes $O(E)$ times altogether, since the sum of the lengths of all adjacency lists is $2|E|$.

Line 11 involves an implicit DECREASE-KEY operation on the min-heap, which a binary min-heap supports in $O(\lg V)$ time.

Total time: $O(V\lg V + E\lg V) = O(E\lg V)$

What about implementing the min-priority queue Q as a FIB-Heap?