Algorithms 10

CS201

Kaustuv Nag

Disjoint-set Data Structure

- A disjoint-set data structure maintains a collection $\mathscr{S} = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets. Each set is represented by a some representative member of the set.
- Letting x denote an object, it supports the following operations:

  MAKE-SET(x) creates a new set whose only member (and thus representative) is x. Since the sets are disjoint, x must not already be in some other set.

  UNION(x, y) unites the dynamic sets that contain x and y, say S_x and S_y, into a new set that is the union of these two sets. We assume that the two sets are disjoint prior to the operation. The representative of the resulting set is any member of $S_x \cup S_y$, although many implementations of UNION specifically choose the representative of either S_x or S_y as the new representative. Since we require the sets in the collection to be disjoint, conceptually we destroy sets S_x and S_y, removing them from the collection $\mathscr{S}$. In practice, we often absorb the elements of one of the sets into the other set.

  FIND-SET(x) returns a pointer to the representative of the (unique) set containing x.

Disjoint-set Data Structure

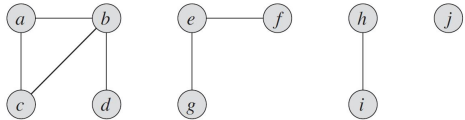
Determining the Connected Components of an Undirected Graph

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ G.V
2      MAKE-SET(v)
3  for each edge (u, v) ∈ G.E
4      if FIND-SET(u) ≠ FIND-SET(v)
5          UNION(u, v)
```

```
SAME-COMPONENT(u, v)
1  if FIND-SET(u) == FIND-SET(v)
2      return TRUE
3  else return FALSE
```

Disjoint-set Data Structure

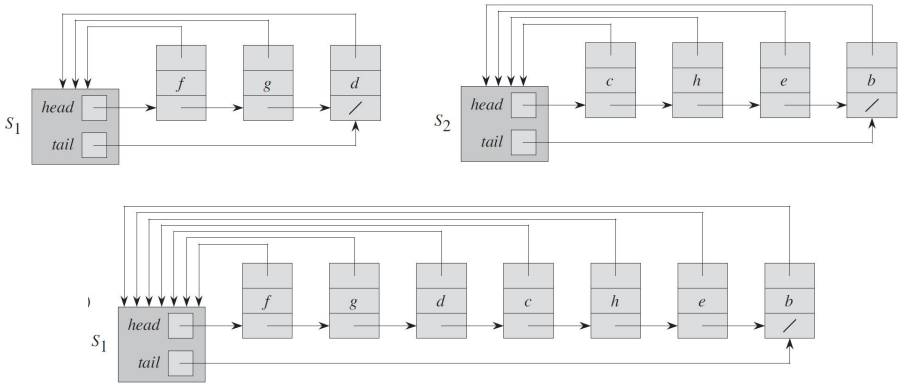
Example



Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

Disjoint-set Data Structure

Implementation

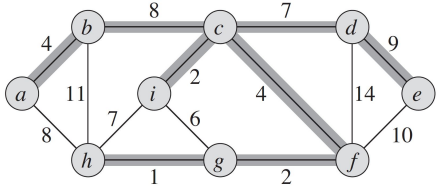


Minimum Spanning Tree

Let there be an undirected graph $G = (V, E)$, where V is the set of vertices, E is the set of edges, and for each edge $(u, v) \in E$, we have a weight $w(u, v)$ specifying the cost to connect u and v. We find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight

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

is minimized. 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.



Growing a Minimum Spanning Tree

- We use a greedy strategy.
- We manage a set of edges A, maintaining the following loop invariant:
  Prior to each iteration, A is a subset of some minimum spanning tree.
- At each step, we determine an edge (u, v) that we can add to A without violating this invariant, in the sense that $A \cup \{(u, v)\}$ is also a subset of a minimum spanning tree.
- We call such an edge a *safe edge* for A, since we can add it safely to A while maintaining the invariant.

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

Use of Loop Invarinat

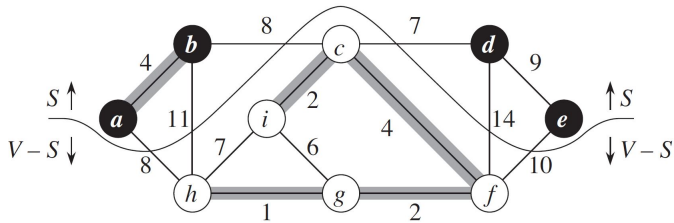
Initialization: After line 1, the set A trivially satisfies the loop invariant.

Maintenance: The loop in lines 2–4 maintains the invariant by adding only safe edges.

Termination: All edges added to A are in a minimum spanning tree, and so the set A returned in line 5 must be a minimum spanning tree.

A Cut of an Undirected Graph

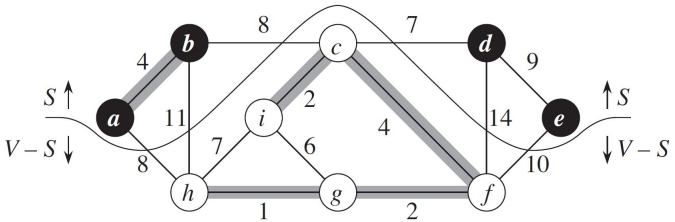
- A *cut* $(S, V-S)$ of an undirected graph $G=(V,E)$ is a partition of V.
- An edge $(u,v) \in E$ *crosses* the cut $(S, V-S)$ if one of its endpoints is in S and the other is in $V-S$.
- A cut *respects* a set A of edges if no edge in A crosses the cut.
- An edge is a *light edge* crossing a cut if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



A Cut of an Undirected Graph

Theorem

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.



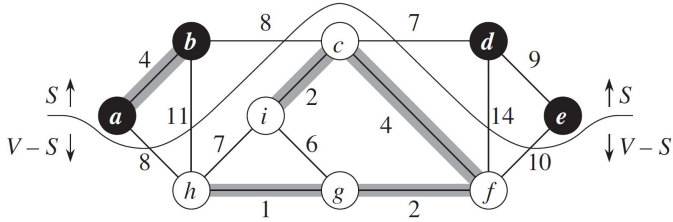
A Cut of an Undirected Graph

Corollary

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

The cut $(V_C, V - V_C)$ respects A, and (u, v) is a light edge for this cut. Therefore, (u, v) is safe for A.



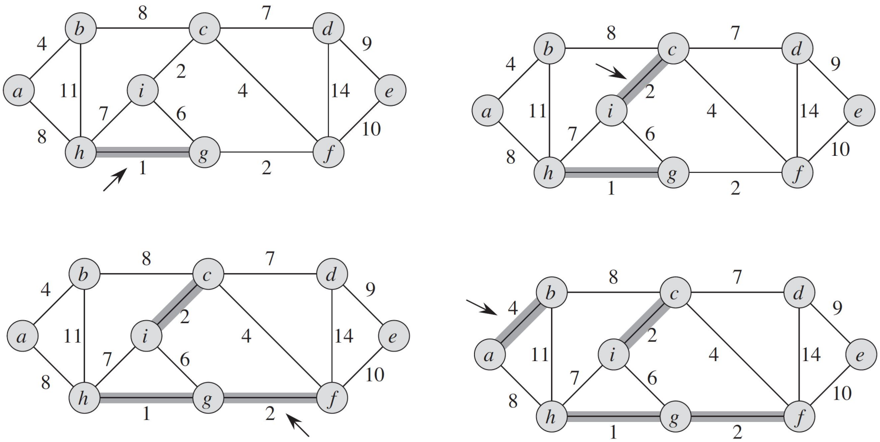
The algorithms of Kruskal and Prim

- In Kruskal's algorithm, the set A is a forest whose vertices are all those of the given graph. 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.

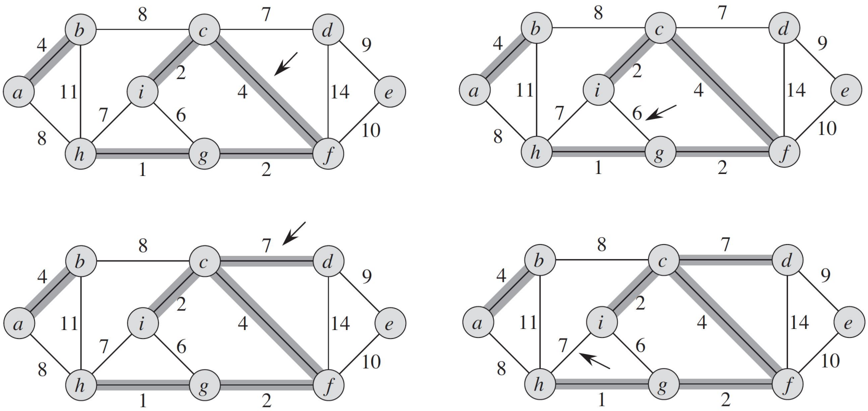
Kruskal's Algorithm

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

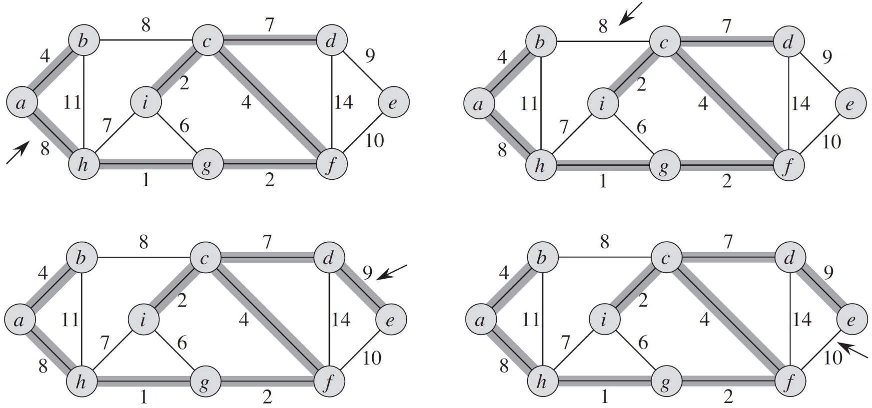
Kruskal’s Algorithm



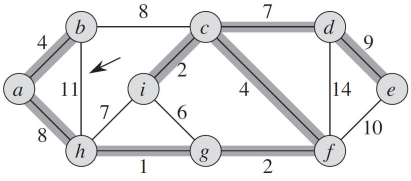
Kruskal's Algorithm

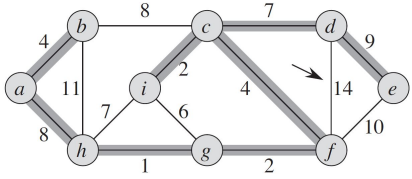


Kruskal’s Algorithm



Kruskal's Algorithm

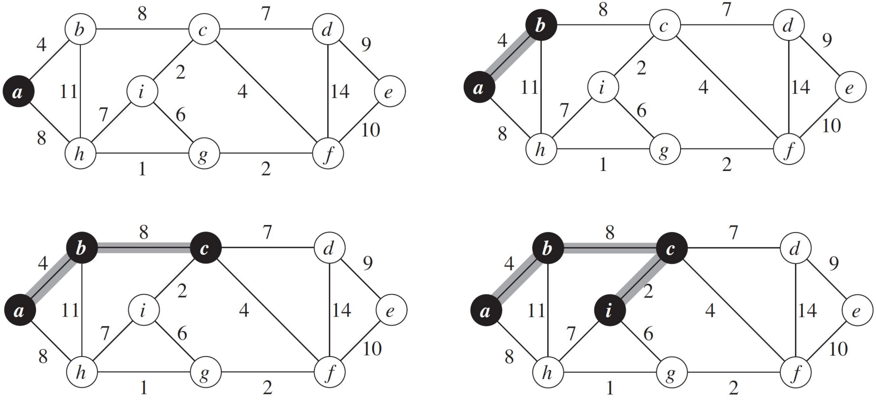




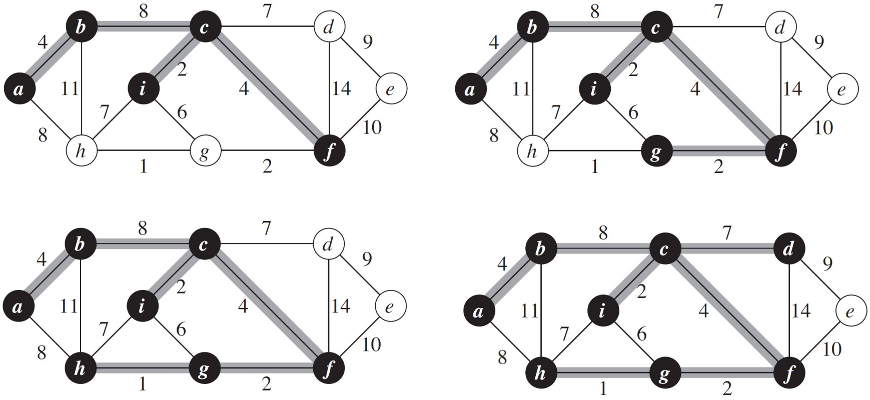
Prim's Algorithm

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

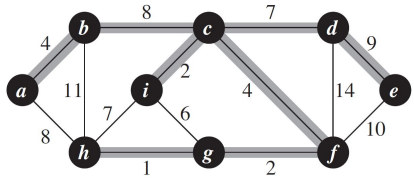
Prim’s Algorithm



Prim's Algorithm



Prim's Algorithm



White Board

White Board