CHAPTER 27

MINIMUM SPANNING TREE

27.1 SPANNING TREE

A **spanning tree** of a graph is just a sub graph that contains all the vertices and is a tree. A graph may have many spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A **minimum spanning tree** or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph has a **minimum spanning forest.**

To explain further upon the Minimum Spanning Tree (MST), and what it applies to, let's consider a couple of real-world examples :

1. One practical application of a MST would be in the design of a network. For instance, a group of individuals, who are separated by varying distances, wish to be connected together in a telephone network. Although MST cannot do anything about the distance from one connection to another, it can be used to determine the least costly paths with no cycles in this network, thereby connecting everyone at a minimum cost.
2. Another useful application of MST would be finding airline routes. The vertices of the graph would represent cities, and the edges would represent routes between the cities. Obviously, the further one has to travel, the more it will cost, so MST can be applied to optimize airline routes by finding the least costly paths with no cycles.

To explain how to find a Minimum Spanning Tree, we will look at two algorithms: the Kruskal algorithm and the Prim algorithm. Both algorithms differ in their methodology, but

both eventually end up with the MST. ***Kruskal's algorithm uses edges, and Prim's algorithm uses vertex connections in determining the MST.*** Both algorithms are greedy algorithms that run in polynomial time. At each step of an algorithm, one of several possible choices must be made.

27.2 KRUSKAL'S ALGORITHM

Kruskal's algorithm is an algorithm that finds a minimum spanning tree for a connected weighted graph. It 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 least weight. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. **If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component).**

```
MST-KRUSKAL (G, w)
1.  A ← ϕ
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
```

Analysis

Where E is the number of edges in the graph and V is the number of vertices, Kruskal's algorithm can be shown to run in $O(E \log E)$ time, or equivalently, $O(E \log V)$ time, all with simple data structures. These running times are equivalent because :

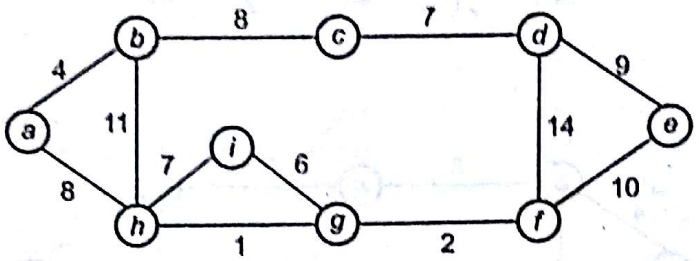
- E is at most V^2 and $\log V^2 = 2 \times \log V$ is $O(\log V)$.
- If we ignore isolated vertices, which will each be their own component of the minimum spanning tree anyway, $V \leq 2E$, so $\log V$ is $O(\log E)$.

We can achieve this bound as follows: first sort the edges by weight using a comparison sort in $O(E \log E)$ time ; Next, we use a disjoint-set data structure to keep track of which vertices are in which components. We need to perform $O(E)$ operations, two find operations and possibly one union for each edge. Even a simple disjoint-set data structure such as disjoint-set forests with union by rank can perform $O(E)$ operations in $O(E \log V)$ time. Thus the total time is

$$O(E \log E) = O(E \log V).$$

Example 1 *Let (u,v) be a minimum weight edge in a graph G. Show that (u,v) belongs to some minimum spanning tree of G.*

Solution. **Proof by contradiction :** assume that (u,v) doesn't belong to some MST of G. If we would add (u,v) to the MST we would get a cycle because MST is a tree. But if we then removed another edge in the cycle we would end up with a MST that is at least as cheap as the original.



Hence, (u,v) does belong to some minimum spanning tree of G.

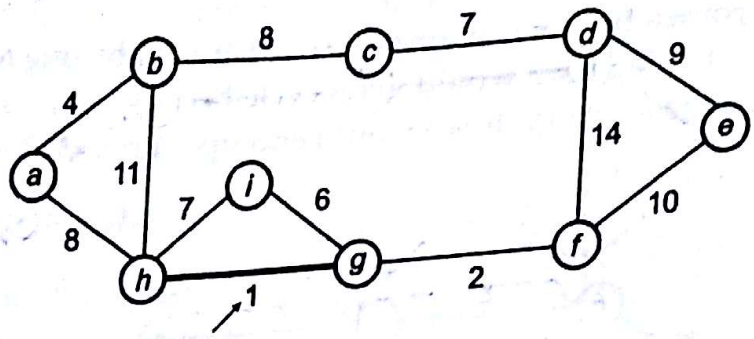
Example 2 *Find the minimum spanning tree of the following graph using Kruskal's algorithm.*

Solution. First we initialize the set A to the empty set and create $|v|$ trees, one containing each vertex with MAKE-SET procedure. Then sort the edges in E into order by non-decreasing weight, *i.e.*,

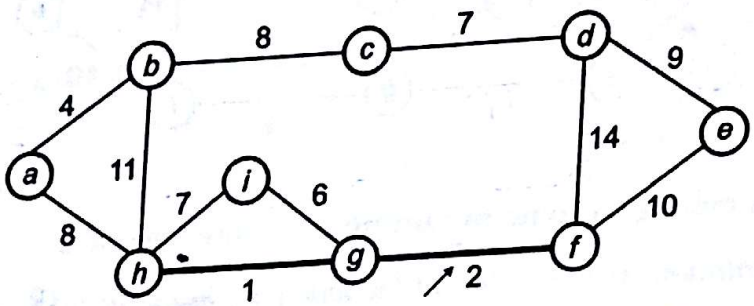
(h,g)	1
(g,f)	2
(a,b)	4
(i,g)	6
(h,i)	7
(c,d)	7
(b,c)	8
(a,h)	8
(d,e)	9
(e,f)	10
(b,h)	11
(d,f)	14

Now, check for each edge (u,v) whether the end points u and v belong to the same tree. If they do, then the edge (u,v) cannot be added. Otherwise, the two vertices belong to different trees and the edge (u,v) is added to A and the vertices in the two trees are merged in by UNION procedure.

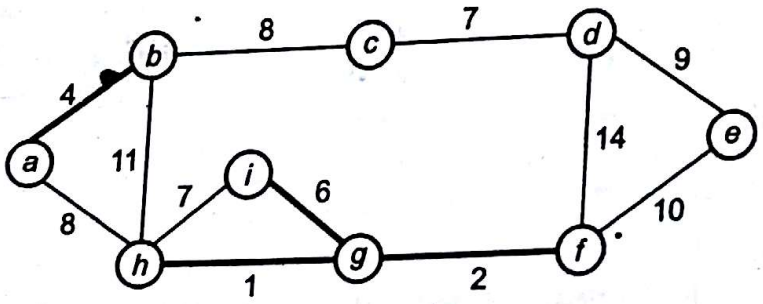
So, first take (h,g) edge



the (g, f) edge.

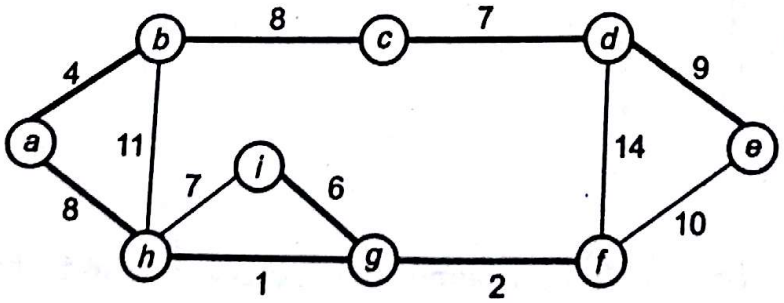


then (a,b) and (i,g) edges are considered and forest becomes.



Now, edge (h,i). Both h and i vertices are in same set, thus it creates a cycle. So this edge is discarded.

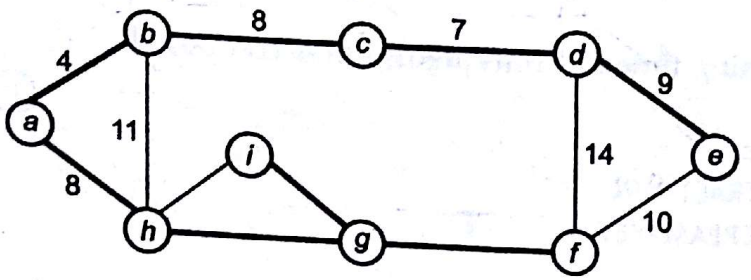
Then edge (c,d), (b,c), (a,h), (d,e), (e,f) are considered and forest becomes.



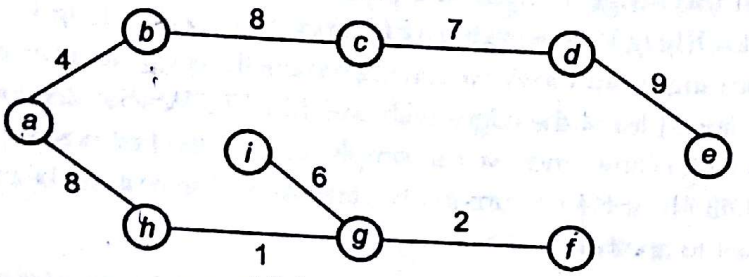
n (e,f) edge both end points e and f exist in same tree so discarded this edge.

hen (b,h) edge, it also creates a cycle.

After that edge (d, f) and the final spanning tree is shown as in dark lines.



or



Minimum cost MST

27.3 PRIM'S ALGORITHM

The main idea of Prim's algorithm is similar to that of Dijkstra's algorithm (discussed later) for finding shortest path in a given graph. Prim's algorithm has the property that the edges in the set A always form a single tree. We begin with some vertex v in a given graph $G=(V, E)$, defining the initial set of vertices A. Then, in each iteration, we choose a minimum-weight edge (u, v), connecting a vertex v in the set A to the vertex u outside of set A. Then vertex u is brought in to A. This process is repeated until a spanning tree is formed. Like Kruskal's algorithm, here too, the important fact about MSTs is we always choose the smallest-weight edge joining a vertex inside set A to the one outside the set A. The implication of this fact is that it adds only edges that are safe for A; therefore when the algorithm terminates, the edges in set A form a MST.

```
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 ≠ ϕ
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)
```

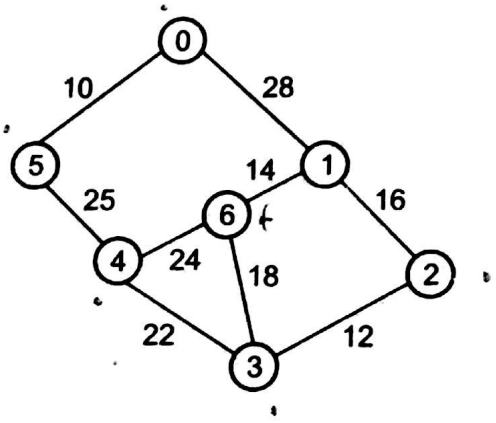
Example 3 Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Prim's algorithm run ? What if the edge weights are integers in the range from 1 to W for some constant W ?

Solution. The running time of Prims algorithm is composed :

$O(V)$ initialization.
$O(V)$-time for EXTRACT-MIN.
$O(E)$-time for DECREASE-KEY.

If the edges are in the range 1.....$|V|$ the priority queue can speed up EXTRACT_MIN and DECREASE-KEY to $O(\lg \lg V)$ thus yielding a total running time of $O(V \lg \lg V + E \lg \lg V) = O(E \lg \lg V)$. If the edges are in the range from 1 to W we can implement the queue as an array $[1 \ldots W+1]$ where the ith slot holds a doubly linked list of the edges with weight i. The $W+1$st slot contains ∞. EXTRACT_MIN now runs in $O(W) = O(1)$ time since we can simply scan for the first nonempty slot and return the first element of that list. DECREASE-KEY runs in $O(1)$ time as well since it can be implemented by moving an element from one slot to another.

Example 4 Generate minimum cost spanning tree for the following graph using Prim's algorithm.



Solution. In Prim's algorithm, first we initialize the priority queue Q. to contain all the vertices and the key of each vertex to ∞ except for the root, whose key is set to 0. Suppose 0 vertex is the root, *i.e.*, *r*. By EXTRACT-MIN(Q) produre, now $u = r$ and Adj[u] = {5,1}.

Removing u from the set Q and adds it to the set $V - Q$ of vertices in the tree. Now, update the key and π fields of every vertex v adjacent to u but not in the tree

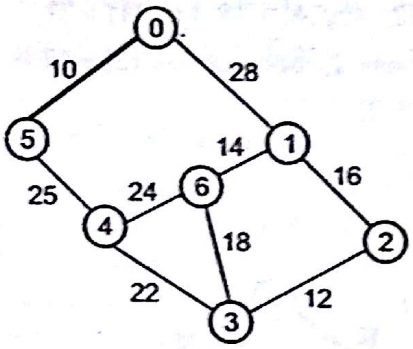
$key\,[5] = \infty$

$w[0,5) = 10$ *i.e.,* $w(u,v) < key\,[5]$

so, $\pi[5] = 0$ and $key\,[5] = 10$.

and $key\,[1] = \infty$

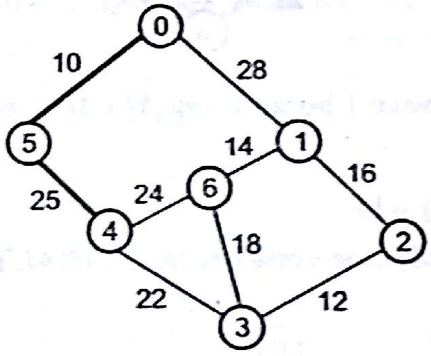
$w(0,1) = 28$ *i.e.,* $w(u,v) < key\,[5]$ so $\pi[1] = 0$ and $key\,[1] = 28$.



Now, by EXTRACT_MIN(Q) Remove 5 because key[5] = 10 which is minimum so $u = 5$.

$Adj\ [5] = \{4\}$

$w(u, v) < key\ [v]$ then $key\ [4] = 25$



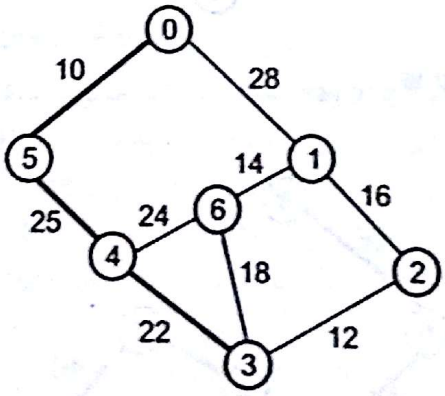
Now remove 4 because $key\ [4] = 25$ which is minimum so $u = 4$.

$Adj\ [4] = \{6, 3\}$

$w(u, v) < key\ [v]$ then $key\ [6] = 24$

$key\ [3] = 22$

Now remove 3 because $key\ [3] = 22$ is minimum so $u = 3$.



$Adj\ [3] = \{4, 6, 2\}$

$4 \neq Q$ $key\ [6] = 24$ now becomes $key\ [6] = 18$.

$key\ [2] = 12$

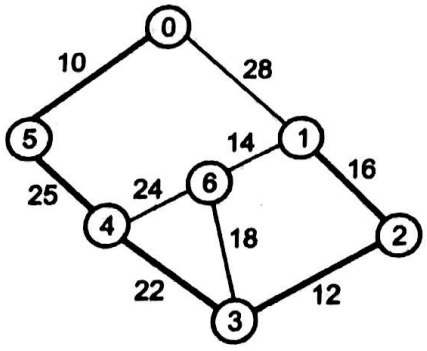
Now, in Q. $key\ [2] = 12$, $key\ [6] = 18$, key $[1] = 28$

By EXTRACT_MIN(Q) Remove 2, because $key\ [2] = 12$ is minimum.

$Adj\ [2] = \{3, 1\}$

$3 \neq Q$.

Now $key\ [1] = 16$.

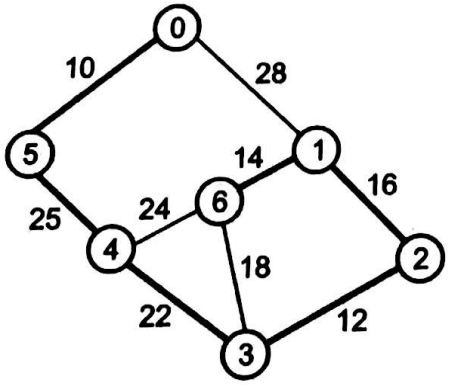


By EXTRACT_MIN(Q) Remove 1 because $key\ [1] = 16$ is minimum.

$Adj\ [1] = \{0, 6, 2\}$

$\{0, 2\} \notin Q$. $key[6] = 14$.

Now becomes Q contains only one vertex 6. By EXTRACT_MIN remove it



Thus, the final spanning tree is