Minimum Spanning Tree

- Spanning Tree of a graph is a Subgraph that is a tree and connect all the vertices together.

- A Single graph can have many different spanning tree.

- A Minimum Spanning Tree (MST) for a weighted, connected and undirected graph is a Spanning Tree with weight less than or equal to the weight of every other Spanning Tree.

- A MST has (V-1) edges where V = no of vertices.

Safe Edge

An edge (u, v) is a safe for A if $A \cup \{(u, v)\}$ is also a subset of some MST.

Definition - A **Cut** $(S, V-S)$ of $G = (V, E)$ is a partition of V.

- An edge $(u, v) \in E$ **Crosses** the Cut $(S, V-S)$ if one of its endpoint is in S and other is in V-S.

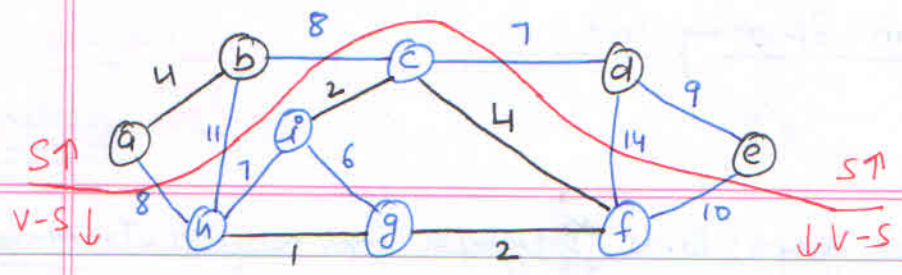
- A Cut that **respects a set A of edges** if no edge in A Crosses the Cut.

- An edge is a **light edge** Crossing a Cut if its weight is the minimum of any edge Crossing the Cut.

Generic-MST(G, w)

1. $A = \emptyset$
2. While A does not form a Spanning tree
3. find an edge (u, v) that is Safe for A
4. $A = A \cup \{(u, v)\}$
5. return A

(AJAY RAWAT)



- Black vertics = Set S
- Blue vertices = (V-S)
- (d,c) edge is the unique light edge crossing the cut.
- A subset A of the edges (black) shaded, cut (S, V-S) respects A since no edge of A crosses the cut.

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 connect two distinct components.

Steps

1) Sort all edges in non-decreasing order of their weight.
2) Pick the smallest edge. check if it form a cycle with the spanning tree formed so far.
3) If cycle is not formed include this edge, else discard it.
4. Repeat step 2,3 until there are (V-1) edges in the spanning tree.

MST-Kruskal (G, w)

O(1)	1.	$A = \phi$	O(1)
	2.	for each vertex $v \in G.V$	] * |V|
O(1)	3.	Make-Set (v)	O(1)]
|E|	4.	Sort the edges of G.E into nondecreasing order by weight w.	

5. for each edge $(u, v) \in G.E$ taken in nondecreasing order by weight
6. if FIND-SET(u) $\neq$ FIND-SET(v)
7. $A = A \cup \{(u, v)\}$
8. UNION(u, v)
9. return A

- It uses a disjoint-set data structure to maintain several disjoint sets of elements.

- Each set contains the vertices in one tree of the current forest.

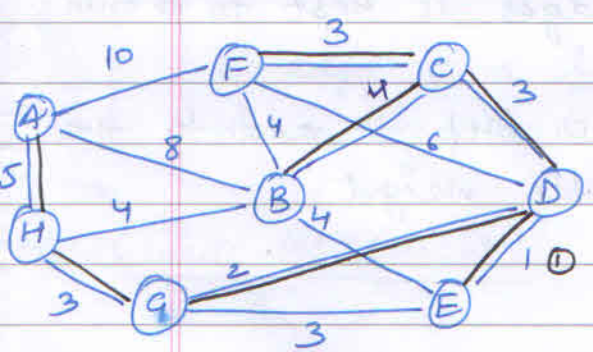
⇒ operations

MAKE-SET(x) - Create a new set whose only member is x.

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

UNION(x, y) - unites the sets that contain x and y, means S_x and S_y into a new set that is the union of the two sets.

Example



- Sort the edges by decreasing weight

edge	w_i	Solution	edge	w_i	Selection
(D,E)	1	✓	(B,E)	4	×
(D,G)	2	✓	(B,F)	4	×
(E,G)	3	×	(B,H)	4	×
(C,D)	3	✓	(A,H)	5	✓
(G,H)	3	✓	(D,F)	6	×
(C,F)	3	✓	(A,B)	8	×
(B,C)	4	✓	(A,F)	10	×

Total cost = $\sum w_i = 21$

Analysis

Theorm - A Seq of 'm' MAK-SET, UNION, find-Set operations, 'n' of which are make-Set operation Can be performed on a disjoint Set forest with Union rank and path Compression in worst Case time. $O(m\alpha(n))$
(α is the very slowly growing function)

- According to above theorm in kruskal algo.
- V makeset operation (n in above theorm)
- (V+E) Seq of make-set, Union, findset operation (m in theorm) $[((V+E)\alpha(V))]$
- As G is Connected we have $|E| \geq V-1$ so disjoint Set operation take $O(E\alpha(V))$ time.
- Moreover $\alpha(|V|) = O(\log V) = O(\log E)$
- Total Running time of kruskal algo is $O(E \log E)$
- Observing that $|E| < |V|^2$ we have $\log|E| = O(\log V)$
- we Can restate running time as $O(E \log V)$

* - This algo works best if no of edges is kept to a min.

- Kruskal is greedy because at each Step it add to the forest an edge of least possible weight.

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.
- Tree starts from an arbitrary root vertex 'r' and grows until the Tree spans all the vertices in 'V.'
- Each Step adds to the tree A a light edges that Connect A to an isolated vertex.
- When algorithm terminates, the edges in A form a minimum Spanning tree.
- During execution, all vertices that are not in the tree reside in a min-priority queue Q based on a key attribute.
- For each v, attribute $v.key$ is the minimum weight of any edge Connecting v to a vertex in the tree.
- $v.key = \infty$, if there is no such edge.
- $v.\pi$ names the parent of v in the tree.

```
MST-PRIM (G, w, r)
(V)  1. for each u ∈ G.V
     2.     u.key = ∞
     3.     u.π = NIL
     4. r.key = 0
O(V) 5. Q = G.V          // Q is a min heap (priority queue)
```

(AJAY RAWAT)

$\|V\|$	6.	while $Q \neq \emptyset$
$O(\log V)$	7.	$u = \text{Extract-Min}(Q)$
$\|E\|$	8.	for each $v \in G.Adj[u]$
	9.	if $v \in Q$ and $w(u,v) < v.key$
	10.	$v.\pi = u$
$(\log V)$	11.	$v.key = w(u,v)$ // Decrease key operation

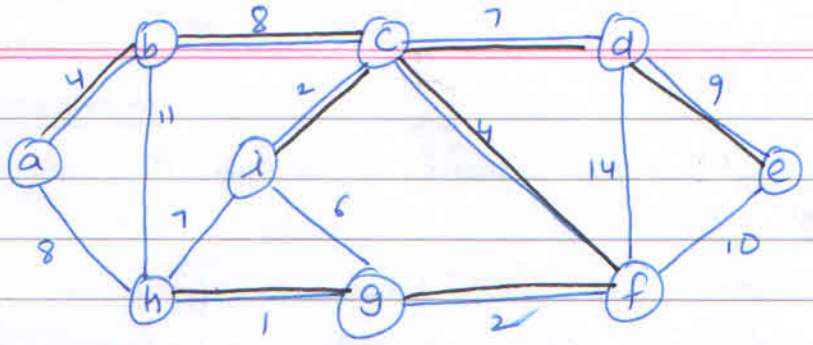
- Sum of the length of all adjacency lists is $2|E|$. (line 8-11)

Total time if we implement Q as a binary min-heap.

- line 1-5 (Build min heap) $O(V)$ time
- while loop executes $|V|$ times and each Extract Min take $O(\log V)$ so total time is $O(V \log V)$
- line 8-11 take $O(E)$
- line 11 involves an implicit Decrease-key operation on min-heap ($O(\log V)$) time.

Thus Total time $O(V \log V + E \log V) = O(E \log V)$

- We can improve the asymptotic running time of Prim's by using Fibonacci heaps.
  - Fibonacci heap holds $|V|$ elements
  - Extract-Min operation takes $O(\log V)$ amortized time
  - Decrease-key operation takes $O(1)$ amortized time
  - If we Can fibonacci heap to implement Priority queue Q running time improves $O(E + V \log V)$



Prim's Algorithm

- Prim's uses a greedy since at each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight.