Analysis of Algorithms

Minimum Spanning Trees

Minimum Spanning Trees

- Problem: Connect a set of nodes by a network of minimal total length
- Some applications:
  - Communication networks
  - Circuit design
  - Layout of highway systems

Motivation: Minimum Spanning Trees

- To minimize the length of a connecting network, it never pays to have cycles.
- The resulting connection graph is connected, undirected, and acyclic, i.e., a *free tree* (sometimes called simply a *tree*).
- This is the *minimum spanning tree* or *MST* problem.

Formal Definition of MST

- Given a connected, undirected, graph $G = (V, E)$, a *spanning tree* is an *acyclic* subset of edges $T \subseteq E$ that connects all the vertices together.
- Assuming G is weighted, we define the *cost* of a spanning tree T to be the sum of edge weights in the spanning tree

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

- A *minimum spanning tree (MST)* is a spanning tree of minimum weight.

Figure1 : Examples of MST

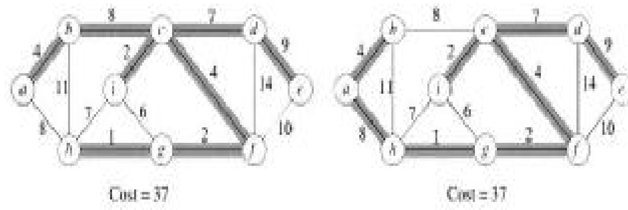

Figure 1: Minimum spanning tree.

- Not only do the edges sum to the same value, but the same set of edge weights appear in the two MSTs. NOTE: An MST may not be unique.

Generic Approaches

- Two greedy algorithms for computing MSTs:
  - Kruskal's Algorithm
  - Prim's Algorithm

Facts about (Free) Trees

- A tree with n vertices has exactly n-1 edges ($|E| = |V| - 1$)
- There exists a unique path between any two vertices of a tree
- Adding any edge to a tree creates a unique cycle; breaking any edge on this cycle restores a tree

Intuition Behind Greedy MST

- We maintain in a subset of edges A, which will initially be empty, and we will add edges one at a time, until equals the MST. We say that a subset $A \subseteq E$ is *viable* if A is a subset of edges in some MST. We say that an edge $(u,v) \in E\text{-}A$ is *safe* if $A \cup \{(u,v)\}$ is viable.
- Basically, the choice (u,v) is a safe choice to add so that A can still be extended to form an MST. Note that if A is viable it cannot contain a cycle. A generic greedy algorithm operates by repeatedly adding any *safe edge* to the current spanning tree.

Generic-MST (G, w)

1. $A \leftarrow \emptyset$ // A trivially satisfies invariant

// lines 2-4 maintain the invariant

2. while A does not form a spanning tree
3. do find an edge (u,v) that is safe for A
4. $A \leftarrow A \cup \{(u,v)\}$

5. return A // A is now a MST

Definitions

- A *cut* $(S, V\text{-}S)$ is just a partition of the vertices into 2 disjoint subsets. An edge (u, v) *crosses* the cut if one endpoint is in S and the other is in $V\text{-}S$. Given a subset of edges A, we say that a cut *respects* A if no edge in A crosses the cut.
- An edge of E is a *light edge* crossing a cut, if among all edges crossing the cut, it has the minimum weight (the light edge may not be unique if there are duplicate edge weights).

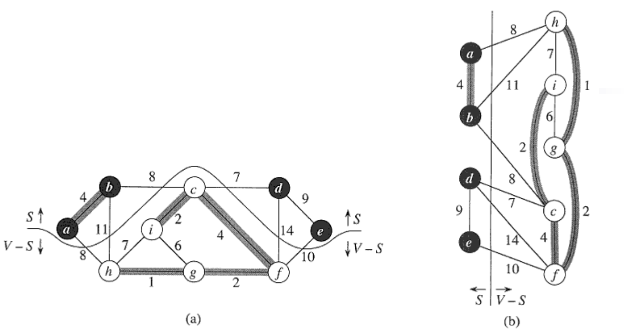

Figure 23.2 Two ways of viewing a cut $(S, V - S)$ of the graph from Figure 23.1. **(a)** The vertices in the set S are shown in black, and those in $V - S$ are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge crossing the cut. A subset A of the edges is shaded; note that the cut $(S, V - S)$ respects A, since no edge of A crosses the cut. **(b)** The same graph with the vertices in the set S on the left and the vertices in the set $V - S$ on the right. An edge crosses the cut if it connects a vertex on the left with a vertex on the right.

When is an Edge Safe?

- If we have computed a partial MST, and we wish to know which edges can be added that do NOT induce a cycle in the current MST, any edge that crosses a respecting cut is a possible candidate.
- Intuition says that since all edges crossing a respecting cut do not induce a cycle, then the lightest edge crossing a cut is a natural choice.

MST Lemma

- Let $G = (V, E)$ be a connected, undirected graph with real-value weights on the edges. Let A be a viable subset of E (i.e. a subset of some MST), let $(S, V\text{-}S)$ be any cut that respects A, and let (u,v) be a light edge crossing this cut. Then, the edge is safe for A.

Proof of MST Lemma

- Must show that $A \cup \{(u,v)\}$ is a subset of some MST.

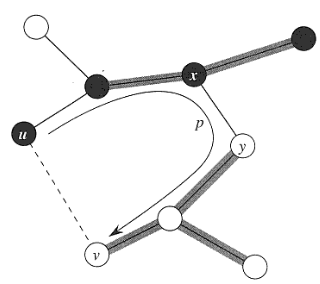

Figure 23.3 The proof of Theorem 23.1. The vertices in S are black, and the vertices in $V - S$ are white. The edges in the minimum spanning tree T are shown, but the edges in the graph G are not. The edges in A are shaded, and (u, v) is a light edge crossing the cut $(S, V - S)$. The edge (x, y) is an edge on the unique path p from u to v in T. A minimum spanning tree T' that contains (u, v) is formed by removing the edge (x, y) from T and adding the edge (u, v).

Proof

- Let T be any MST for G containing A and assume that it does not contain the (u,v).
- The edge (u,v) forms a cycle with the edges on the path p from u to v in T.
- Since u and v are on opposite sides of the cut $(S,V\text{-}S)$, there is a at least one edge in T on path p that also crosses the cut. Let (x,y) be such an edge.
- The edge (x,y) is not in A, because the cut respects A.

Proof

- Since (x,y) is on the unique path from u to v in T, removing (x,y) breaks T into two components.
- Adding (u,v) reconnects them to form a new spanning tree $T' = T - \{(x,y)\} \cup \{(u,v)\}$.
- Since (u,v) is a light edge crossing (S, V-S) and (x,y) also crosses this cut, $w(u,v) \le w(x,y)$.
- $W(T') = w(T) - w(x,y) + w(u,v) \le w(T)$.

MST Lemma: Reprise

- Let $G = (V, E)$ be a connected, undirected graph with real-value weights on the edges. Let A be a viable subset of E (i.e. a subset of some MST), let $(S, V\text{-}S)$ be any cut that respects A, and let (u,v) be a light edge crossing this cut. Then, the edge is safe for A.
- *Point of Lemma*: Greedy strategy works!

Basics of Kruskal's Algorithm

- Attempts to add edges to A in increasing order of weight (lightest edge first)
  - If the next edge does not induce a cycle among the current set of edges, then it is added to A.
  - If it does, then this edge is passed over, and we consider the next edge in order.
  - As this algorithm runs, the edges of A will induce a forest on the vertices and the trees of this forest are merged together until we have a single tree containing all vertices.

Detecting a Cycle

- We can perform a DFS on subgraph induced by the edges of A, but this takes too much time.
- Use "disjoint set UNION-FIND" data structure. This data structure supports 3 operations:
  Create-Set(u): create a set containing u, takes O(1) time.
  Find-Set(u): Find the set that contains u , takes O(1) time.
  Union(u, v): Merge the sets containing u and v , takes O($lg\ n$) time.
- The vertices of the graph will be elements to be stored in the sets; the sets will be vertices in each tree of A (stored as a simple list of edges).

MST-Kruskal(G, w)

```
1. A ← ∅                                  // initially A is empty
2. for each vertex v ∈ V[G]               // line 2-3 takes O(V) time
3.     do Create-Set(v)                   // create set for each vertex
4. sort the edges of E by nondecreasing weight w
5. for each edge (u,v) ∈ E, in order by nondecreasing weight
6.     do if Find-Set(u) ≠ Find-Set(v)    // u&v on different trees
7.            then A ← A ∪ {(u,v)}
8.                 Union(u,v)
9. return A
```

Total running time is $O(E\ lg\ E)$.

Example: Kruskal's Algorithm

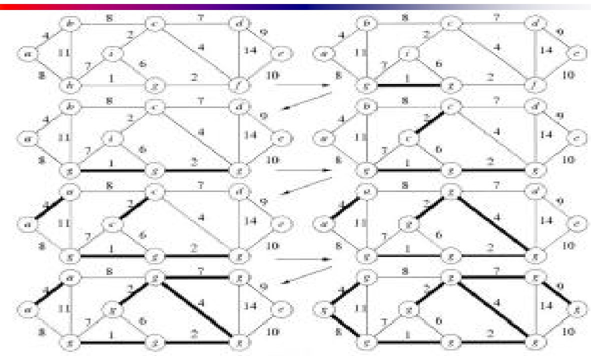

Figure 4: Kruskal's Algorithm

Analysis of Kruskal

- Lines 1-3 (initialization): O(V)
- Line 4 (sorting): O(E lg E)
- Lines 6-8 (set-operation): O(E log E)
- Total: O(E log E)