Chapter 23: Minimum Spanning Trees

Given a connected, undirected graph $G =< V, E >$, each edge $(u, v) \in E$ has a weight $w(u, v)$. The minimum spanning tree T is an acyclic subset of E that connects all vertices of V and whose weight $w(T) = \sum_{(u,v) \in V} w(u, v)$ is minimized.

Since T is acyclic and connects all vertices, it is a tree. We call it minimum spanning tree.

- ## Growing a Minimum Spanning Tree:

  - The generic algorithm:
    Manage a set A of edges that is always a subset of some minimum spanning tree.
    Initially $A =$. At each step, wee add an edge $(u, v) \in E$ into A, where (u, v) is **safe** to A.
    An edge (u, v) is safe to A if $A \cup \{(u, v)\}$ is still a subset of some minimum spanning tree.

```
Generic-MST(G, w)
A <-- {}
while A does not form a minimum spanning tree
      do find an edge (u,v) that is safe to A
         A <-- A U {(u,v)}    // U: union
return A
```

  - What kind of edges are safe to A?
    * Let a **cut** $(S, V - S)$ of an undirected graph $G =< V, E >$ is a partition of V.
    * **An edge** $(u, v) \in E$ **crossing the cut** $(S, V - S)$ if one of its end points is in S and the other one 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 among all edges crossing the cut.

    Theorem: *If A is 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, the edge (u, v) is safe to A.*

    We ignore the proof of the theorem. The above theorem suggests us a way to find a minimum spanning tree.
    Ex:
    It is easy for human with eye vision to find a sequence of cuts that respects A. But for a computer algorithm, it is not easy. Any minimum spanning tree algorithm tries to suggest a sequence of cuts respects the growing A.

  - ## Kruskal's Algorithm:
    The safe edge added to A at each step is always the least-weight edge in the graph that

connects two distinct components.

The algorithm is greedy because at each step it adds to *A* an edge with the least possible weight.

```
MST-Kruskal(G, w)
A <-- {}
for each vertex v in V[G]
    do Make-Set(v)
sort the edges of E by non-decreasing weight w
for each edge (u,v) in E, in order by non-decreasing weight
    do if Find-Set(u) != Find-Set(v)
          then A <-- A U {(u,v)}
               Union(u,v)
return A
```

Ex:

- ## Prim's Algorithm:

  The set A maintained is a growing tree.

  The safe edge added to A at each step is always the least-weight edge crossing the cut $(B, V - B)$, where B is the set of vertices connected by edges in A.

  The algorithm is greedy because the tree is augmented at each step with an edge that contributes the minimal amount of possible weight.

```
MST-Prim(G, w, r)  // r: source vertex
Q <-- V[G]     // Q: priority queue
for each u in Q
    do key[u] <-- infinity
key[r] <-- 0
p[r] <-- nil
while Q != {}
      do u <-- Extract-Min(Q)
         for each v in Adj[u]
             do if v in Q and w(u,v) < key[v]
                   then p[v] <-- u
                        key[v] <-- w(u,v)
```

  Prim's algorithm uses a priority queue Q based on a key field.

  The queue Q maintains all vertices that are still not in the growing tree.

  The key field for each vertex v in Q, key[v], is the weight of the light edge connecting v to the tree.

Ex:

Actually, the sequence of cuts suggested by Prim is the sequence of cuts that separate the growing tree from the rest of vertices at each step.