23.2 Minimum Spanning Trees

Kruskal's algorithm:

Kruskal's algorithm solves the Minimum Spanning Tree problem in $O(|E| \log |V|)$ time. It employs the disjoint-set data structure that is similarly used for finding connected components in an undirected graph.

Kruskal's algorithm proceeds iteratively. Within each iteration, it chooses a light edge, so it is a greedy algorithm.

The second algorithm for the MST problem is Prim's algorithm, which also adopts the greedy policy.

23.2 Prim's Algorithm for MST

Prim's Algorithm is a special case of the generic MST algorithm. In this case, the edges in A always form a single subtree. In contrast, **the edges in A in the Kruskal's algorithm always form a forest**.

- We start with $A = \emptyset$, and a vertex set $U = \{r\}$, where r is any vertex in V.
- Add a light edge from $U \times (V \setminus U) \cap E$ to A,
- This procedure continues until the edges in A form a minimum spanning tree.

Here, the edge added to A at each step is:

An edge a of least weight with exactly one of its endpoints being in U.

At the same step, we add the other endpoint of a to U.
a **is light for the cut** $(U, V \setminus U)$.

Therefore, Prim's algorithm is correct. (By the Theorem in Lecture 24.)

23.2 Prim's Algorithm (continued)

```
Prim(G, w, r)
    for each v ∈ G.V do
        v.key ← ∞;
        v.π ← NIL;
    r.key ← 0;
    Q ← V;        /* Priority Queue (Q is a MIN-HEAP) */
    while Q ≠ ∅ do
        u ← Extract_Min(Q)     /* Now u is added to U */
        for each v ∈ G.Adj[u] do
            if v ∈ Q and w(u, v) < v.key then
                v.π ← u;
                v.key ← w(u, v);     /* with Decrease_Key operation */
```

23.2 Prim's Algorithm (continued)

The vertex set U and the edge set A are empty initially. All vertices reside in a min-priority queue Q. Throughout the algorithm, we have $U = V \setminus Q$, that is, U is the set of vertices not in the queue Q. During the algorithm, A is implicitly kept as: $A = \{(v, v.\pi) : v \in V \setminus Q \setminus \{r\}$.

The algorithm maintains the following invariant:
For each vertex v in the queue:

- (1) If there are edges from v to nodes in U, $(v, v.\pi)$ is a least weight edge from v to a node $u \in U$, and $v.key$ is the weight of the edge $w(u, v)$.

- (2) If there is no edge from v to U, $v.\pi = NIL$ and $v.key = \infty$.

At the end, the edges in set $A = \{(v, v.\pi) \mid v \in V \setminus \{r\}$ form an MST.

23.2 Prim's Algorithm (continued)

The performance of Prim's algorithm depends on **how we implement the priority queue Q**. If we use the MIN-HEAP as the priority queue, its time complexity is analysed as follows.

- Steps 1-5 take $O(|V|)$ time in total.
- Step 7 takes $O(\log |V|)$ time. Perform it once for each vertex, it takes $O(|V| \log |V|)$ time in total on Extract_Min procedures.
- The for loop in lines 8-11 executes $O(|E|)$ times altogether, since the sum of the lengths of all adjacency lists is $2|E|$. Step 11 takes $O(\log |V|)$ time, and the other Steps inside this loop can be done in constant time.
- The total running time of the algorithm thus is $O(|V|) + O(|V| \log |V|) + O(|E| \log |V|) = O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$.

Other algorithms for MSTs

Top-down algorithm

Let $e_1, e_2, \ldots, e_E$ be the edge sequence sorted in decreasing order of their weights;
$T \leftarrow G$;
for $i \leftarrow 1$ **to** $|E|$ **do**
  if the removal of edge e_i from T doesn't disconnect it **then**
    Delete e_i from T;

BFS or DFS can be used to check the connectivity in each step of this algorithm.

For a more efficient way to check the connectivity, see the following paper (optional):

S. Even and Y. Shiloach, An On-Line Edge-Deletion Problem, *Journal of the Association for Computing Machinery*, Vol. 28., pp. 1-4, 1981.

Other algorithms for MSTs

Guan's algorithm

$T \leftarrow G$;
while T is not a tree **do**
    Find a cycle C in T;
    Delete a maximum weight edge in C;

MST history: The very first algorithm for finding a minimum spanning tree was developed by Czech scientist Otakar Borůvka in 1926, which is the Kruskal algorithm (reinvented in the mid-50s). See pages 641 and 642 of our textbook, or the following paper (optional):

J. Nešetřil. Otakar Borůvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history, *Discrete Mathematics*, Vol. 233., pp. 3-36, 2001.

Other algorithms for MSTs (continued)

Borůvka's algorithm (1926)

We "choose" a sequence of edges which form a set L of subtrees such that each vertex is in one tree.

L = a set of V trees, each a single vertex
while L has more than one tree **do**
    for each tree T in L **do**, simultaneously
        Choose a minimum edge from T to $V-T$, breaking ties
            according to some ordering of the edges
    Use these chosen edges to combine members of L.

Borůvka's algorithm is the basis of very fast distributed and parallel algorithms for MSTs.

The fastest known running time for MST is $O(|E|\alpha(|E|,V))$ where α is the very slowly growing inverse of Ackermann's function. This algorithm is also based in part on Borůvka's.

23.7 MST's Applications

- Road network building
- Broadcast in communication networks
- Use as a subroutine for multicast
- ...

The Steiner Tree Problem

A **minimum spanning tree** is the least-weight way to connect together all vertices in a graph. In some applications, we don't need to connect together all the vertices but only some particular vertices. This is the **Steiner tree problem**:

Given: A graph $G = (V, E)$ with weights on the edges, and a subset $D \subseteq V$.
Required: Find a subtree T of G of minimum weight such that all the vertices of D lie on T.

The Steiner tree problem is much harder than the MST problem. It is an NP-hard problem and no polynomial time algorithm is known to solve it. However, there is an efficient approximation algorithm with approximation ratio of 2 for it.