Overview: We will review the problem of finding a minimum spanning tree in a graph. We will revisit the three (deterministic) greedy algorithms: Kruskal's, Prim's, and Borůvka 's. All three has the complexity of $O(m \log n)$. However, by an intelligent modification of the Borůvka 's algorithm, the randomized algorithm can achieve a linear cost of $O(m+n)$.

1 Minimum Spanning Tree

Recall from CS170 the definition of the minimum spanning tree: given an un-directed graph G with weights on its edges, a spanning tree consists of a subset of edges from G that connects all the vertices. Among them, a minimum spanning tree (MST) has the minimum total weight over its edges.

By convention, let n be the number of vertices in G and m be the number of edges. Note that a tree that has n vertices invariably has $n-1$ edges. Therefore, the MST must have $n-1$ edges.
Exercise : Show that without loss of generality, all the edge weights can be assume positive.

The problem of finding a MST is called MSTP and it has a wide range of applications, for instance, in trying to connect up a number of geographical sites by communication cables and minimize the total length of the cable used.

2 Deterministic Algorithms

In the following discussion, assume all the edge weights are distinct.

Two well-known algorithms for computing the MST are the Prim's algorithm and Kruskal's algorithm. Both are based on the greedy property that a minimum weight edge (under certain restriction) can be added safely, without leading off from the MST.

The Prim's algorithm expands a tree. At each step, an edge is added to the tree such that:

1. The edge is connected to the tree (so it remains a tree after the edge is added).
2. Adding the edge doesn't introduce a cycle.
3. The edge has the smallest weight.

One can implement this algorithm by maintaining a heap of the edges incident to the tree, ordered by their weights. Every time a new vertex is added, the heap of candidate edges are updated with the incident edges that the new vertex introduces. Clearly, all those operations can be done in $O(\log m)$ as a heap. And each edge will be encountered, leading to a total cost of $O(m \log m) = O(m \log n)$.

Another algorithm is the Kruskal's algorithm. Instead of keeping a tree, Kruskal's algorithm keeps a forest at each step. Recall that a forest is a collection of disjoint trees. Again, during each step an edge is added:

1. The edge doesn't bring a cycle into the forest.

2. The edge has the smallest weight.

To maintain a data structure for the forest is a bit more cumbersome than that for a tree. But it turns out that the operations required from this algorithm can be implemented efficiently, leading to a total cost of $O(m \log n)$, too.

A third algorithm, which dates back to 1926, is the one by the Czechish Borůvka . Like the other two, it is a remarkably simple idea that exploits so-called greedy property: For an edge e, if there is a cut $(C, V-C)$ such that the edge e is the minimum weight edge connecting C and $V-C$ [1], then the edge e must appear in the MST.

To see this, suppose the MST connects up the two components C and $V-C$ differently, by another edge e', then we reach an immediate contradiction: we may as well erase e' and add in e, which will produce another spanning tree but with a smaller total weight. Q.E.D.

In the proof above, we've introduced a new edge e, which brings a cycle, then we break the cycle by erasing e' and obtain another tree with smaller total weight. This idea of "add an edge, break a cycle" is generally useful in proving properties of the MST.

Exercise : Prove that if the edge weights are distinct, there's a unique MST.

Exercise : Using greedy property, prove that for any vertex, the incident edge with the smallest weight must appear in the MST.

By the result of the second exercise, we know that initially, we can connect many edges at a time: every vertex connects to its nearest neighbor. The resulting edge may already been connected, but you'd encounter the same edge at most twice (from either endpoints). Therefore, after going through all the vertices, you get at least $\frac{n}{2}$ edges.

Those edges scatters across the graph, forming a number of connected components. The goal of connecting up all the vertices now reduces to connecting up those connected components. Think of each connected component as a giant vertex, then we are back to the original problem of a set of unconnected vertices. There we can recurse.

What brings a set of connected vertices into a giant, combined vertex? The idea of contraction. However, instead of edge contraction as we saw in the min-cut algorithm. Here, the way of contraction works differently: we don't want to preserve the cut between the giant vertex and vertices outside the component, which leads to adding up the edge weights. Instead, we want to preserve the *minimum* weight edge between the giant vertex and outside vertices. Given an outside vertex u, we iterate through the vertices v in the connected components c, and find the minimum among the edges connecting u and any v, then assign that weight to the edge between u and the combined vertex c.

Exercise : To clarify the preceding intuition, prove that if we combine the MST of each connected components and the MST of the contracted graph, the resulting tree is the MST of the original graph. (Hint: by greedy property)

The idea of recursive contraction leads to the Borůvka 's algorithm:

At each level of recursion, connecting the edges costs $O(n)$; and we claim that the edge contraction step can be done in $O(m)$ (by the same data structure used in the min-cut algorithm). Hence, the total cost at one level is $O(m)$. Note that the size of the contracted graph G' halves after each recursion. Therefore, the number of levels is $\log n$. Hence the total cost $O(m \log n)$.

[1] meaning that edge e connects a vertex in C with a vertex in $V-C$

Algorithm 1 Borůvka 's algorithm

OUTPUT: the MST
$T \leftarrow \emptyset$
for all $v \in V$ **do**
  $e \leftarrow$ the minimum weight edge incident on v
  $T \leftarrow T + \{e\}$
end for
$G' \leftarrow G$ with all the edges in T contracted
$T' \leftarrow$Borůvka (G')
RETURN T +T'

3 MST verification using a forest

Let F be a forest in G. Define $w_F(u,v)$ to be the maximum weight on the edges in the path from u to v, if one such path exists. Otherwise, set $w_F(u,v) = \infty$. (Note that if a path exists between u and v, then it must be unique because F is a collection of trees.)

Define an edge (u,v) to be *F-heavy* if $w(u,v) > w_F(u,v)$. An edge that is not F-heavy is said to be *F-light*.

Exercise (Exercise 10.14 in M&R): Let F be any forest in graph G. Show that if an edge (u,v) is F-heavy, then it does not lie in the MST of G. Verify that the converse is not true.

Therefore, we can make up any F and delete all F-heavy edges in the graph without hurting the MST. In fact, people have worked out algorithms to identify F-heavy edges efficiently:

Fact 1 (Theorem 10.18 in M&R): Given a graph G and a forest F, all F-heavy edges in G can be identified in time $O(n+m)$.

4 Random sampling of subgraphs

So far, we've known that we can prune edges once we have a forest. However, we don't know what forest to use. A good choice of the forest should leave only a small number of F-light edges. In fact, people have constructed one such F, based on random sampling of the edges.

The key result is (the proof is omitted):

Fact 2 (Lemma 10.19 in M&R): Let F be the minimum spanning forest [2] in the random graph $G(p)$ obtained by independently including each edge of G with probability p. Then, the expected number of F-light edges in G is at most n/p.

5 The linear-time MST algorithm

Recall the factor of m in the running time of the original Borůvka algorithm: $O(m \log n)$. It comes from the number of edges in each recursion step. Imagine that we could bring down the number of edges like what we did to the number of vertices (half each time), then hopefully the running

[2] A minimum spanning forest consists of MSTs of each connected components of the original graph. It is a generalization of the idea of MST.

time should improve. Here, the idea of using a forest to prune the edges comes in handy for this purpose. Following this intuition, and by careful construction, one can reach the following algorithm(for computing the minimum spanning forest, abbrev. as MSF):

Algorithm 2 Randomized-MSF

Input: Weighted, undirected graph G
Output: MSF for G
1. Apply 3 steps of the original Borůvka algorithm to obtain a contracted graph G_1 of at most $n/8$ vertices. Let the set of contracted edges be C. If G_1 is empty then we are done and return C as the MSF.
2. Let $G_2 = G_1(p)$ (as defined in Fact 2) and let $p = \frac{1}{2}$.
3. Recursively apply RANDOMIZED-MST on G_2 to obtain its MSF F_2.
4. Prune the F_2-heavy edges in G_1 to obtain G_3.
5. Recursively apply RANDOMIZED-MST on G_3 to obtain its MSF F_3.
6. RETURN forest $F = C + F_3$.

Let $T(n,m)$ denote the expected running time of the above algorithm. Consider the cost of the steps in the algorithm and it should lead us to a recurrence relation for $T(n,m)$.

Step 1 costs $O(n+m)$ by the argument of the original Borůvka 's algorithm. Step 2 basically copies over the vertices and flips a coin for each edge, hence it runs in $O(n+m)$. Step 3 finds the MSF for G_2, which has $\leq n/8$ vertices and an expected number of $m/2$ edges. Therefore, the cost of step 3 is $T(n/8,m/2)$. The pruning process in step 4 costs $O(n+m)$ by fact 1. And it preserves at most $\frac{n/8}{1/2} = n/4$ edges in G_1, by fact 2. Therefore, the number of edges in G_3 is $\leq n/4$. Therefore, step 5 takes time $T(n/8,n/4)$. And step 6 takes $O(n)$ time.

Putting all this together, we get that

$$T(n,m) \leq T(n/8,m/2) + T(n/8,n/4) + c(n+m)$$

for some constant c. A solution to this recurrence relation is given by $2c(n+m)$, implying that the expected running time of the RANDOMIZED-MSF algorithm is $O(n+m)$. (Note that this is the type of Las Vegas algorithms).