COMPSCI 330: Design and Analysis of Algorithms September 19

Lecture 7: Greedy Algorithms II

Lecturer: Rong Ge *Scribe: Rohith Kuditipudi*

1 Overview

In this lecture, we continue our discussion of minimum spanning trees (MSTs) from Lecture 14. We describe and analyze two algorithms for computing MSTs: *Prim's algorithm* and *Kruskal's algorithm*.

2 Designing an algorithm to compute MSTs

Given what we know about the properties of MSTs (in particular, recall the Key Lemma from Lecture 14), a natural way to compute an MST for a particular graph would be the following procedure:

1. Initialize an empty tree (i.e. set of edges)
2. Find a cut for which we have yet to add a spanning edge to our current set of edges
3. Add a minimum spanning edge of the cut to our set of edges
4. Repeat steps 2 and 3 until we have added $n-1$ edges to our set

From the Key Property of MSTs (see Lecture 14), if we follow the directions above we are guaranteed the existence of an MST comprised precisely of our resulting set of edges. Unfortunately however, these directions are too vague to constitute an actual algorithm for computing MSTs. In particular, two important ambiguities remain unresolved:

1. In step 2, how do we find a cut for which our set does not contain a spanning edge?
2. In step 3, how do we find a minimum spanning edge over the cut?

As we will see in the following sections, Prim's and Kruskal's algorithms are both well-defined procedures for computing MSTs that resolve both of these ambiguities.

3 Prim's algorithm

In a manner closely resembling Djikstra's algorithm, Prim's algorithm resolves the two ambiguities described in Section 2 by 1) maintaining a set of visited vertices S and 2) at each iteration adding a minimum edge spanning the cut $(S, V\backslash S)$ to the current tree.

Algorithm 1: Prim's algorithm

```
Input: Graph $G = (V, E)$, initial vertex $s$
for $v \in V$ do
    initialize d[$v$] $= \infty$ and prev[$v$] = NULL
end
for $v \in$ neighbors of $s$ do
    d[$v$] $\leftarrow$ w[$(s, v)$], prev[$v$] $= s$
end
for $i = 2, ..., n$ do
    Search through unvisited vertices and find vertex $u$ such that d[$u$] is minimal
    Mark $u$ as visited
    for $v \in$ neighbors of $u$ do
        if $w[(u, v)] < d[v]$ then
            d[$v$] $\leftarrow$ w[$(u, v)$]
            prev[$v$] $\leftarrow u$
        end
    end
end
return $prev$
```

3.1 Runtime analysis

The runtime of Prim's algorithm is $O(|E| + |V| \log |V|)$, or in other words the same as Djikstra's algorithm.

4 Kruskal's algorithm

Kruskal's algorithm is pretty simple. First, sort all edges in ascending order by weight and initialize an empty set of edges S. Then, find the lightest edge e among $E\backslash S$ and add e to S, so long as neither $e \cup S$ contains a cycle nor $|S| = n - 1$ already.

Algorithm 2: Kruskal's algorithm

```
Input: Graph $G = (V, E)$
initialize an empty set of edges $S$
sort $E$ in ascending order by weight while $|S| < n - 1$ do
    pop $e$ from $E$
    if $S \cup e$ contains no cycles then
        $S \leftarrow S \cup e$
    end
end
return $S$
```

4.1 Runtime analysis

Sorting the edges takes $O(|E| \log |E|)$ time. And using a specialized data structure that we will learn about later, determining whether adding e to S will result in a cycle can be done in $o(\log |E|)$ time. Thus, the overall runtime is $O(|E| \log |E|)$.

4.2 Proof of correctness

To prove the correctness of Kruskal's algorithm, we will demonstrate that every edge $e = (u, v)$ we add to S over the course of the algorithm is a minimum spanning edge across at least one cut.

To do so, let C denote the connected component containing u (with respect to the edges already in S). Note that because we only add e to S if $S \cup e$ contains no cycles, we are guaranteed that $v \notin C$. We also know that none of the edges already in S span the cut $(C, V \backslash C)$—from the fact that $e' \in S$ and so u' and v' must lie in the same connected component w.r.t. S, it follows that there cannot exist an edge $e' = (u', v') \in S$ such that $u' \in C$ and $v' \notin C$. And so seeing as e is the lightest edge not already in S, it follows that e is a minimal spanning edge over the cut $(C, V \backslash C)$.

5 Summary

We have discussed and analyzed two algorithms for computing MSTs: Prim's algorithm and Kruskal's algorithm. In general, Prim's algorithm tends to be faster than Kruskal's algorithm. That being said, actually obtaining the optimal runtime of Prim's algorithm requires implementing a specialized data structure. In practice, Kruskal's algorithm is both "good enough" and easier to implement.