1 Greedy Algorithm

We continue our discussion of greedy algorithms.

1.1 Problem: Minimum Spanning Trees

Given a weighted graph G, find a minimum spanning tree of G.

1.1.1 Trees and Cuts

Lemma 1.1

Let G be a weighted graph, and let C be a cut (i.e. a partition of the vertices of G into two sets).

- Let e be a lowest weight edge crossing C. Then, there exists a MST of G that contains e.
- Let e be the unique lowest weight edge crossing C. Then, every MST of G contains e.

Proof. Suppose we have a cut C which splits the vertices V into two subsets V_1 and V_2. Let T be a MST; then, T's edges will have to cross this cut at some point. If $e \in T$, we're done. Otherwise, we can add the edge to T such that it crosses the cut. This creates a cycle R, which implies that R contains some other e' that crosses the cut C. Then, let

$$T' = T \cup \{e\} \setminus \{e'\}$$

Then, the weight of T' is given by

$$w(T') = w(T) + w(e) - w(e')$$

but since e is a minimum weight edge, and the weight of e must be no more than e', so it follows that

$$w(T') \leq w(T)$$

Additionally, if e is a unique minimum weight edge, then $w(T') < w(T)$, which is a contradiction as this implies that T wasn't a minimum spanning tree. So, we are done. □

1.1.2 Prim's Algorithm

Prim's algorithm relies on the above lemma. Let $b(v)$ be the lightest weight edge (for vertices that we have not reached) that we have discovered that allows us to reach v from the vertices that we have reached.

```
Prims(G):
    T = {}
    b(v) = inf
    b(s) = 0
    Insert all V's into priority queue Q
    while Q not empty:
        u = DeleteMin(Q)
        If u != s:
            Add (u, prev(u)) to T
        For all (u, v) in E:
            if v in Q and b(v) > l(u, v):
                b(v) = l(u, v)
                DecreaseKey(v)
                prev(v) = u
    return T
```

Basically, what's going on is that:

- For all v in the priority queue (which we haven't reached), $b(v)$ will store the cheapest length of an edge that connects it to some vertex that is already connected to s, and `prev(v)` will tell you the other end of that best edge.
- So, find the vertex in Q with the smallest with the smallest value of b, which is the lightest edge which connects some vertex in u which has been connected to s to some vertex that hasn't. We will add that edge to the tree, and then we need to do some updates.

This algorithm looks nearly like Dijkstra's algorithm, and in fact has the same runtime as Dijkstra's algorithm. That is, $\mathcal{O}(|V|\log(|V|) + |E|)$.