1 Dijkstra's Algorithm & Priority Queues

Given a graph where each edge has some length ℓ, how do we find the shortest path between two vertices?

1.1 Trivial Way

The idea is the break edges into unit length edges. So, an edge of length 5 can actually be seen as 5 edges of length 1. With this conversion, we can run BFS. However, the runtime is O(Sum of Edge Lengths).

1.2 Another Way

If we have very long edge lengths, most steps will just consist of advancing slightly along a bunch of edges. So, how do we "fast forward" through these boring steps? Well, occasionally, we have interesting steps where the wavefront hits a new vertex.

1.3 Algorithm

```
Distances(G, s, l)
    dist(v) = 0
    While not all distances found
        Find minimum over (v, w) in E
            with v discovered w not
            of dist(v) + l(v, w)
        dist(w) = dist(v) + l(v, w)
        prev(w) = v
```

1.4 Why Does This Work?

Proposition. *Whenever the algorithm assigns a distance to a vertex v, that is the length of the shortest path from s to v.*

Proof. We use induction.

Base Case: We know that `dist(s) = 0`. The empty path has length 0.

Inductive Step: When assigning distances to w, suppose that all previously assigned distances are correct. TODO

□

This runs in $O(|V| \cdot |E|)$ time. There are $O(|V|)$ iterations and $O(|E|)$ edges.

- This is too slow because every iteration we have to check every edge.
- The idea is that most of the comparison doesn't change much iteration to iteration. So, we can use this to save time.
- Specifically, record for each w the best value of $\text{dist}(v, w) + \ell(v, w)$.

1.5 Better Algorithm

```
Distances(G, s, l)
    For v in V
        dist(v) = infinity
        done(v) = false
    dist(s) = 0
    done(s) = false
```

```
while not all vertices done
    Find v not done with minimum dist(v)
    done(v) = true
    For (v, w) in E
        if dist(v) + l(v, w) < dist(w)
            dist(w) = dist(v) + l(v, w)
            prev(w) = v
```

The initialization is $O(|V|)$, and the while loop is $O(|V|)$. The for loop is $O(|E|)$ time. Thus, the runtime is:

$$O(|V|^2 + |E|)$$

- This repeatedly asks for the smallest vertex. Even though not much is changing from round to round, the algorithm is computing the minimum from scratch every time.
- We can use a data structure to help answer a bunch of similar questions faster than answering each question individually (like the one above).

1.6 Priority Queue

A **priority queue** is a data structure that stores elements sorted by a **key** value. Its operations are:

- `insert`: Adds a new element to the priority queue.
- `decreaseKey`: Changes the key of an element of the priority queue to a specified smaller value.
- `deleteMin`: Deletes the element with the lowest key from the priority queue.

1.7 Even Better Priority Queue

```
Dijkstra(G, s, l)
    Initialize Priority Queue Q
    For v in V
        dist(v) = infinity
        Q.insert(v) // using dist(v) as key
    dist(s) = 0
    while Q not empty
        v = Q.deleteMin()
        For (v, w) in E
            if dist(v) + l(v, w) < dist(w)
                dist(w) = dist(v) + l(v, w)
                prev(w) = v
                // w has a faster path and needs to updated in
                // the priority queue
                Q.decreaseKey(w)
```

The runtime is as follows:

- We need to iterate $O(|V|)$ times for the initial loop.
- In the `while` loop, we run $O(|V|)$ times.
- We need to run through the edges $O(|E|)$ times.

So, $O(|V|)$ `inserts` + $O(|V|)$ `deleteMins` + $O(|E|)$ `decreaseKey`.