Dijkstra's Algorithm

1 Shortest Paths

In a graph with no or uniform edge weights, use BFS to find the shortest path between nodes in $\mathcal{O}(n+m)$ time. However, what if the edge weights differ? Or if they are not necessarily numeric, but are distributions, or outputs of another algorithm? The problem becomes more difficult.

The problem we are trying to solve: Given a digraph $\mathcal{G}$ with edge costs c_e and source node s, find the shortest path (with respect to c_e) to a target node t or to all other nodes in $\mathcal{G}$.

2 Dijkstra's Algorithm

Dijkstra's Algorithm, named for Edsger Dijkstra, is an algorithm to find the shortest path from s to all other nodes in a weighted, directed graph. It separates $\mathcal{G}$ into three sections: the nodes we have finalized, the nodes we have discovered but not finalized, and the nodes we have not discovered. For each node n, the algorithm stores its distance d from s as well as the parent of n, used to recursively compute the path from s to n. Then, the algorithm discovers new nodes such that $\sum d$ across the path is minimized.

Algorithm 1 Dijkstra's Algorithm

```
1: $S = [s]$
2: $d(s) = 0$
3: while $S \neq V$ do
4:     Consider all edges $e = (u, v)$ leaving $u$
5:     Add to $S$ node $v$ such that $d(u) + C_{u,v}$ is minimized
6:     Set $d(v) = d(u) + C_{u,v}$
7:     Set $p(v) = u$
8: end while
```

Now we'll prove correctness of Dijkstra's Algorithm, keeping in mind its two major assumptions: that $\mathcal{G}$ is connected and all $c_e \geq 0$.

Theorem 1. *Dijkstra's Algorithm finds shortest paths from s to all nodes v.*

Theorem 2. *$d(v)$ is the correct distance from $s \to v$ and $p(v)$ is the last hop on an $s \to v$ path.*

Proof. Our loop invariant is that at any point during the execution of the algorithm, Theorem 2 holds for all $V \in S$. We perform a proof by induction on the iterations.

Basis step: When $S = \{s\}$, Theorem 2 holds trivially.

Inductive hypothesis: Theorem 2 holds for all iterations $0 \leq n \leq k$.

Inductive step: Consider iteration $k+1$. We add some node v to S, then set $d(v) = d(u) + C_{u,v}$ and $p(v) = u$. Thus Theorem 2 holds for v, and S remained constant otherwise, so we know that Theorem 2 holds for all $V \in S$ as well.

However, we still need to show that there is no path from $s \to v$ shorter than $d(u) + C_{u,v}$. To do so, we assume for the sake of contradiction that there is a path of total length less than $d(u) + C_{u,v}$ and consider the first time it leaves S along an edge (u', v'). This edge may not necessarily come from u or go to v, but it is along our hypothetical shortest path. Then $d(u') + C_{u,v} = dist(s, u') + C_{u',v'}$. Because we assumed no negative edges, $dist(s, u') + C_{u',v'} \leq dist(s, v) < d(u) + C_{u,v}$. This inequality shows that the algorithm would choose (u', v') over (u, v) in the first place, which is a contradiction. $\square$

3 Implementation

To implement Dijkstra's Algorithm, use d as a priority metric to implement a minheap containing the candidate set (i.e., "nodes discovered but not finalized"). The minheap operations must include:

Operation	Runtime	Times run
insert(h, v)	$\Theta(\log n)$	n
findMin(h)	$\Theta(1)$	n
removeMin(h)	$\Theta(\log n)$	n
decrementPriority(h, v)	$\Theta(\log n)$	m

Thus we can see that Dijkstra's Algorithm runs in $\Theta(m \log n)$ time for non-sparse graphs. In order to optimize this runtime, we would like to reduce the runtime of decrementPriority to $\Theta(1)$ while increasing the runtime of findMin to $\Theta(\log n)$. By doing so, we decrease the overall runtime to $\Theta(m + n \log n)$. This can be implemented with a Fibonacci Heap, which will be discussed next week.