CSE 331: Introduction to Algorithm Analysis and Design

Greedy Algorithms

1 Shortest Path With Weighted Edges

1.1 Problem

Input: A weighted graph $G = (V, E)$ (directed or undirected) and a starting node $s \in V$. Each edge $e \in E$ has a weight $l_e > 0$.

Output: The length of the shortest path from s to t for all $t \in V$.

1.2 Dijkstra's Algorithm

```
1. $d'(s) = 0$
2. $R = \{s\}$
3. While $\exists_{(u,v)\in E}$ where $u \in R \wedge v \notin R$
    (a) Choose $v$ with the smallest $d'(v)$
    (b) $R = R \bigcup \{v\}$
    (c) $d(v) = d'(v)$
    (d) $d'(w) = \min(d'(w), d(v) + l_e)$ for each $e = (v, w)$
4. return $d$ for all nodes
```

1.3 Correctness

Let P_v be the length of the shortest path from s to v.

We will prove the correctness of Dijkstra's algorithm by induction on the size of R throughout the run of the algorithm. Specifically, at any point in time during a run of the algorithm, for any node $v \in R$, $d(v)$ is the length of the shortest path from s to v.

Proof. **Base case:** For $|R| = 1$, the only node in R is s and $d'(s)$ is initialized to 0 meaning $d(0)$ ends up being 0 after the first execution of the loop. With no negative edges, we can't find a path shorter than 0 length.

Induction step: Assume that at any point during the run of the algorithm, $d(v)$ is the length of the shortest path from s to v for all $v \in R$. Prove that the next node w to be added to R that $d(w)$ will be the length of the shortest path from s to w.

Since w is the next node to be added to R, there must exist an edge $e = (u, w)$ such that $u \in R$. We want to prove that $d(u) + l_e$ is the length of the shortest path from s to w which we will do by contradiction. To this end, assume that there exists a path P from s to w with length less than $d(u) + l_e$. Since $s \in R$ and $w \notin R$, there must be an edge $e' = (x, y)$ in P such that $x \in R$ and $y \notin R$. Since the total length of P is less than $d(u) + l_e$, then we know that $d(x) + l_{e'} < d(u) + l_e$ which implies that $d'(y) < d'(w)$. This is a contradiction since the algorithm always chooses to add the node with the smallest d' value to R, but the algorithm chose to add w, not y.

$\square$

1.4 Runtime

1. $d'(s) = 0$ $O(1)$
2. $R = \{s\}$ $O(1)$
3. While $\exists_{(u,v) \in E}$ where $u \in R \wedge v \notin R$ Each iteration of the loop adds a node to R making the total number of iterations $O(n')$ where n' is the size of the connected component of s. Since s can't be connected to more nodes than the number of edges in the graph, this loop also runs $O(m)$ times which we will use for the analysis.
   (a) Choose v with the smallest $d'(v)$ This can be preformed in $O(1)$ time with a priority queue implementation.
   (b) $R = R \bigcup \{v\}$ $O(1)$
   (c) $d(v) = d'(v)$ $O(1)$
   (d) $d'(w) = \min(d'(w), d(v) + l_e)$ for each $e = (v, w)$ With a priority queue implementation, this step can be performed in $\log(n)$ time to fix up the heap. Though the loop executes $O(n)$ times and this step can execute $O(m)$ times if v is connected to every edge in the graph, the total number of times d' values are updated is $O(m)$ since each edge is checked at most twice (see the total number of neighbors proof or the runtime analysis for BFS). This makes the total runtime of this step throughout the entire run of the algorithm $O(m \log n)$.
4. return d for all nodes $O(1)$

By utilizing a priority queue, the total runtime is $O(1) + O(1) + O(m) \cdot (O(1) + O(1) + O(1)) + O(m \log n) + O(1)$ which is $O(m \log n)$.