CS340 Analysis of Algorithms

Handout:	**2**	**Professor:**	**Dianna Xu**
Title:	**Dijkstra's Shortest Path**	**E-mail:**	dxu@cs.brynmawr.edu
Date:		**URL:**	**http://cs.brynmawr.edu/cs340**

Sample description, pseudo code and proof of correctness for Dijkstra's shortest path algorithm. Please refer to lecture notes for details of the algorithm design and time analysis. Recall that given a directed graph $G=(V,E)$, with associated non-negative edge weights $w(u,v)$ for each edge $(u,v)\in E$, and a source vertex $s\in V$, we want to compute the shortest path from s to all vertices in V.

1 Description

The key idea of Dijstra's is to maintain an estimate of the shortest path length `d[v]` for each vertex v from s, stored in an array `d` indexed by the vertices. The algorithm updates these estimates as it processes more and more vertices, a process known as relaxation. Every iteration, the algorithm will select the unvisited vertex u with the smallest known distance from s, i.e. `d[u]` is minimum, mark it visited, and update the estimate of all of its neighbors, by comparing the existing estimate `d[v]` of a neighbor v (note that `d[v]` currently stores the best known shortest path from s to v without going through u) with the distance of going from s through u to v and update if necessary. A priority queue keyed by each `d[u]` is used to keep track of which vertex to process next.

2 Pseudocode

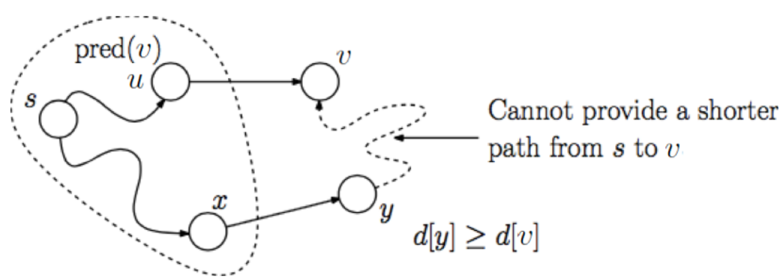
```
Function Dijkstra(G=(V,E),s)
    for each u∈V do
        d[u] = ∞
    end
    d[s]= 0
    Q = priority queue of all vertices u keyed by d[u]
    while Q is not empty do
        u = extractMin from Q
        for each v∈ Adj[u] do
            // if going through u to v is shorter
            if d[u] + w(u, v) < d[v] then
                d[v] = d[u] + w(u, v)
                decrease v's key value in Q to d[v]
            end
        end
    end
```

Note that in the implementation above, no visited tags are necessary and we consider all vertices in the priority queue unprocessed/unvisited. As soon as a vertex is removed from the priority queue, it is considered processed/visited.

3 Correctness Proof

Termination is easily argued because the set Q is finite and we delete at least one vertex in each iteration of the `while` loop. The inner `for` loop also teriminates because G is finite.
We prove the optimality of `Dijstra` by showing that the estimates are computed correctly for all processessed (visited) vertices. Let $\delta(s,v)$ denote the length of the true shortest path from s to v.
Lemma: $d[v]=\delta(s,v)$, $\forall v \in S$, where S is the visited/processed set by Dijkstra's
Proof: by induction.

- Base case: $|S|=1$, S consists of only the source vertex s. We set $d[s]$ to 0 and $\delta(s,s)=0$.
- IH: assume that $d[v]=\delta(s,v)$, $\forall v \in S$, where $|S|=k$
- Want to show: lemma holds for $|S|=k+1$, that is, the $d[v]$ of the next vertex added to S is also computed correctly.
  Let v be the next vertex added to S, along with edge (u,v). Note that u must be a vertex already in S, because otherwise the shortest path will be disconnected. We argue that the true shortest path from s to v must be $d[u]+w(u,v)$. Suppose this is not true, and let us consider any other $s \longrightarrow v$ path P. Note that no matter which edges P uses, because $v \notin S$, P must have an edge that goes across the cut from S to $V \setminus S$. Let (x,y) be the first edge taken by P where $x \in S$ and $y \in V \setminus S$. Note that it may be that $x=s$ and/or $y=v$, but u and x must be distinct.



  By the IH, u and x are both correctly processed and therefore $d[u]=\delta(s,u)$ and $d[x]=\delta(s,x)$. In addition, because relaxations were applied when processing u and x, (recall Dijkstra's relaxes all neighbors when processing a vertex), and we are specifically deciding between adding v to S via the (u,v) edge or y to S via the (x,y) edge, it must be that $d[v]$ was updated to $d[u]+w(u,v)$ when relaxing along (u,v) and $d[y]$ was updated to $d[x]+w(x,y)$ when relaxing along (x,y). By construction, because Dijkstra's selected v and not y as the next vertex to process, we must have $d[v] \leq d[y]$. Therefore, we have

$$\delta(s,v) \leq d[v] \leq d[y]$$

  With a $d[y]$ already not smaller than $d[v]$, and we know that G does not contain any edges with negative weights, there is no way for us to construct a P that connects s to y then to v that can achieve a total length shorter than $d[v]$. Thus we have our contradiction. ■