
CS 2110 Summer 2011: Dijkstra's Algorithm

1 Dijkstra's Algorithm

```
Dijkstra(WeightedGraph g, Vertex s) {
    set d[v] = infinite for all nodes v
    pq = PriorityQueue containing pairs (v, cost), smallest to largest
    d[s] = 0
    add (v, d[v]) to pq for all nodes v
    while pq is not empty {
        remove lowest cost node v from pq
        for each nbr of v {
            if d[nbr] > d[v] + w(v, nbr) {
                d[nbr] = d[v] + w(v, nbr)
                update cost of nbr in pq
                parent[nbr] = v
            }
        }
    }
    // d encodes distances, parent encodes lowest cost paths
}
```

2 Proof of correctness

The algorithm is shown above. Below I provide a short proof that the algorithm is correct.

Let $\delta(u, v)$ be the cost of the least cost path from u to v . Let $d[v]$ be the cost stored by Dijkstra.

Claim 1: $d[v] \geq \delta(u, v)$.

Informal proof of claim 1: initially, $d[v] = \infty$ and it's only reduced when we find a path to v . Clearly, the cost of the found path can never be less than the cost of the least cost path.

Least cost path from s to u

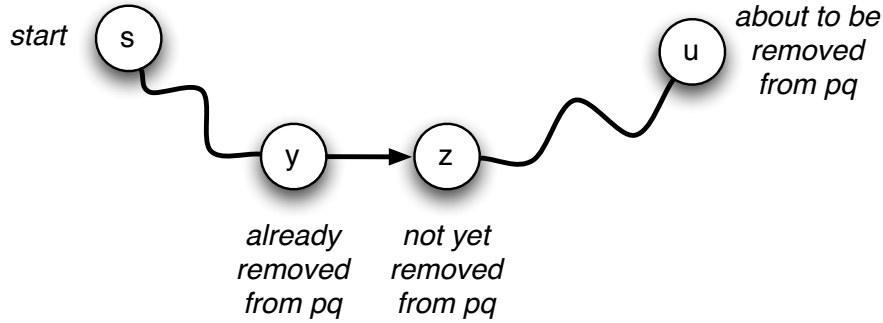


Figure 1: Example graph.

Claim 2: the following invariant holds: in the while loop, when v is removed from the priority queue, $d[v] \leq \delta(s, v)$.

If this claim is true then the two claims above imply that $d[v] = \delta(s, v)$.

Proof sketch of claim 2: This is a proof by contradiction. Suppose the claim is false there is some node for which Dijkstra's computes the wrong answer. In particular, let u be the *first* node removed from the priority queue such that $d[u] > \delta(s, u)$.

There is some shortest path from s to u . Let's call this path P . It's shown in the figure above. Let's consider the moment in the execution of Dijkstra's algorithm when node u is being removed from the priority queue.

Let y be a node that lies on the least cost path and has already been removed. Let z be also on the least cost path but not yet removed. (Note: it's fair to assume that such nodes exist. For instance, we can set $y = s$ and $z = u$.)

We make the following observations

- (a) $d[y] = \delta(s, y)$. This is true because v is defined as the *first* node removed where $d[u] > \delta(s, u)$ and since y has already been removed it must be the case that $d[y] \leq \delta(s, y)$, and we have claim 1 which says $d[y] \geq \delta(s, y)$.
- (b) $d[z] \leq d[y] + w(y, z)$. This is true because when we removed y from the queue, the for each loop ensures that we visited each neighbor of y , including z , and updated its cost. So if at that moment, $d[z] > d[y] + w(y, z)$, then $d[z]$ would have been updated to be $d[z] = d[y] + w(y, z)$. And notice that in the algorithm, costs start at ∞ and can only decrease.
- (c) $d[u] \leq d[z]$. This is true because u is about to be removed and z is defined as a node not yet removed from the queue (so therefore its cost must be not less than u 's).

(d) $\delta(s, z) = \delta(s, y) + w(y, z)$ and $\delta(s, u) = \delta(s, z) + \delta(z, u)$. These claims are true because these nodes lie on P which is defined to be the shortest path.

Putting these claims together we have

$$d[u] \leq d[z] \leq d[y] + w(y, z) = \delta(s, z) \leq \delta(s, z) + \delta(z, u) = \delta(s, u)$$

This contradicts our supposition that $d[u] > \delta(s, u)$. Therefore, our supposition is false and there can be no node such that $d[u] > \delta(s, u)$.