Graph algorithms - Dijkstra's algorithm

Problem

Given a graph with non-negative costs and two vertices *s* and *t*, find a minimum cost walk from *s* to *t*.

Idea

Dijkstra's algorithm still relies on Bellman's optimality principle; however, it computes distances from the starting vertex in increasing order of the distances. This way, the distance from start to a given vertex doesn't have to be recomputed after the vertex is processed.

This way, Dijkstra's algorithm looks a bit like the breadth-first traversal; however, the queue is replaced by a priority queue where the top vertex is the closest to the starting vertex.

The algorithm

```
Input:
    G : directed graph with costs
    s, t : two vertices
Output:
    dist : a map that associates, to each accessible vertex, the cost of the minimum
           cost walk from s to it
    prev : a map that maps each accessible vertex to its predecessor on a path from s to it
Algorithm:
    PriorityQueue q
    Dictionary prev
    Dictionary dist
    q.enqueue(s, 0)              // second argument is priority
    dist[s] = 0
    found = false
    while not q.isEmpty() and not found do
        x = q.dequeue()      // dequeues the element with minimum value of priority
        for y in Nout(x) do
            if y not in dist.keys() or dist[x] + cost(x,y) < dist[y] then
                dist[y] = dist[x] + cost(x, y)
                q.enqueue(y, dist[y])
                prev[y] = x
            end if
        end for
        if x == t then
            found = true
        endif
    end while
```

- If all costs are non-negative, the algorithm above doesn't put a vertex into the priority queue once it was extracted and processed (see proof below).
- If there are negative costs, but no negative cost cycles, then a vertex may be processed multiple times. However, if we eliminate the exit on dequeueing the target vertex, the algorithm finishes after a finite number of steps and the result is correct.
- If there is a negative cost cycle accessible from the starting vertex, then the algoritm can end with an incorrect result or it can run forever.

Proof of correctness (for non-negative costs)

Non-negative costs case

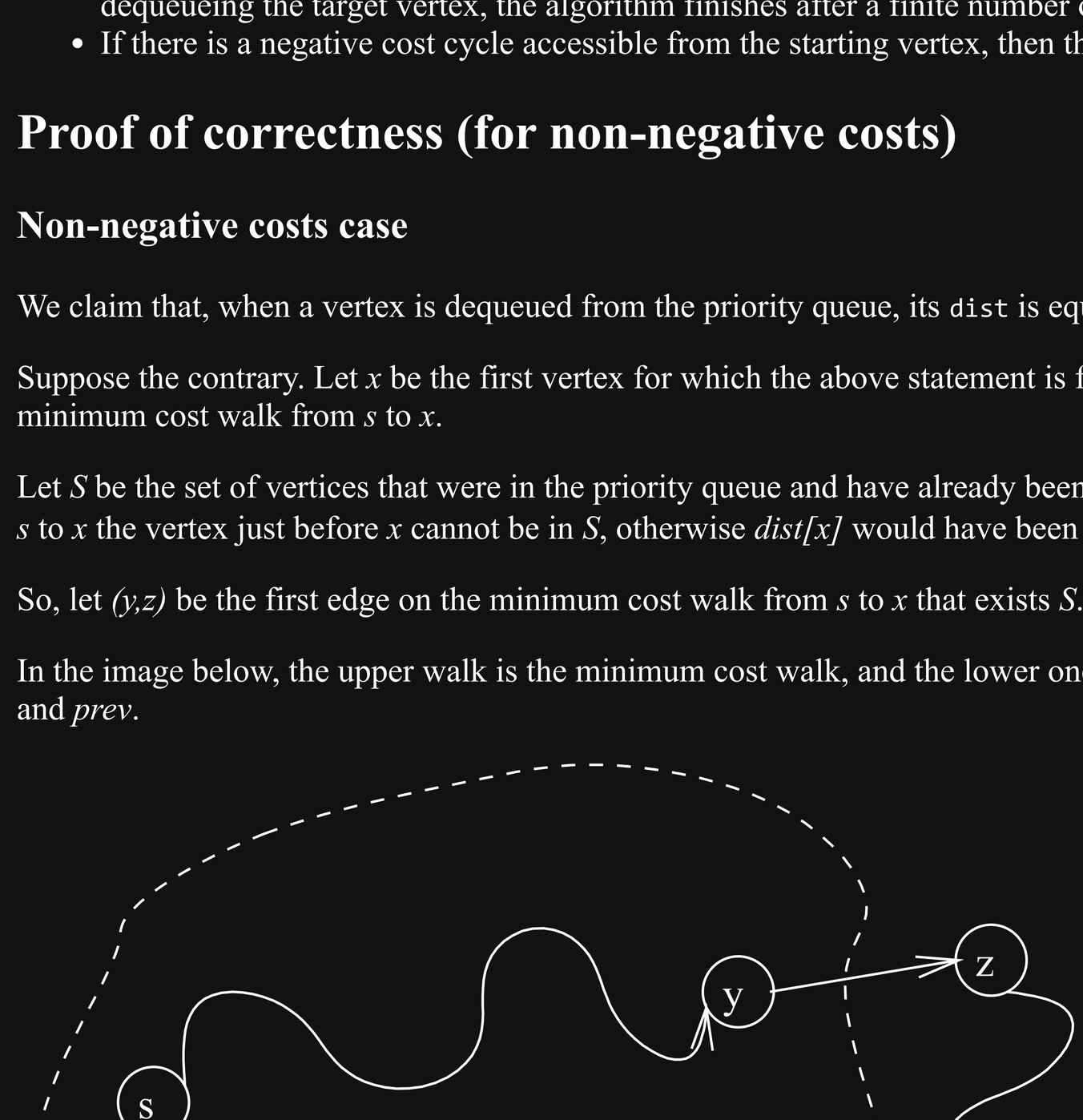
We claim that, when a vertex is dequeued from the priority queue, its `dist` is equal to the cost of the minimum cost walk from the start to it.

Suppose the contrary. Let *x* be the first vertex for which the above statement is false. So, we have that *dist[x]* is strictly smaller than the cost of the minimum cost walk from *s* to *x*.

Let *S* be the set of vertices that were in the priority queue and have already been dequeued from it when *x* gets dequeued ($x \notin S$). On the best walh from *s* to *x* the vertex just before *x* cannot be in *S*, otherwise *dist[x]* would have been correctly computed when that vertex was dequeued.

So, let *(y,z)* be the first edge on the minimum cost walk from *s* to *x* that exists *S*.

In the image below, the upper walk is the minimum cost walk, and the lower one is the one found by the algorithm, and implied by the values of *dist* and *prev*.



However, since *x* is at the top of the priority queue and not *z*, we have that $cost(s,...,y,z) \geq cost(s,...,x)$ and, since all edges have non-negative costs, $cost(z,...,x) \geq 0$. Therefore, the bottom walk, found by the algorithm, cannot have a larger cost than the minimum cost walk, which prove our claim.

The case of negative costs

Radu-Lucian LUPŞA
2016-03-14