Graph algorithms - Dijkstra's algorithm

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.

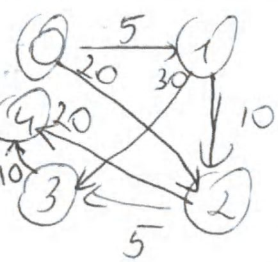
This way, Dijkstra's algorithm looks a bit like the breadth-first traversal; however, the queue is replaced by a priority queue where the top vertex is the closest to the starting vertex.

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.

Dijkstra's algorithm: < the minimum cost walk from a vertex s to all the other vertices; graph with non-negative costs.



$s = 0, t = 4$

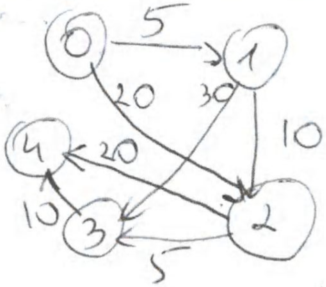
	x	y	dist: dictionary	q: priority queue	prev: dictionary
initialization			0: [0]	← [(0,0)] ←	
iteration 1	0				
iteration 1.1		1	0: 0, 1: 5	← [(1,5)] ←	1: 0
iteration 1.2		2	0: 0, 1: 5, 2: 20	← [(1,5) (2,20)] ←	1: 0, 2: 0
iteration 2	1			← [(2,20)] ←	
relaxation ← iteration 2.1		2	0: 0, 1: 5, 2: 15	← [(2,15)] ←	1: 0, 2: 1
iteration 2.2		3	0: 0, 1: 5, 2: 15, 3: 35	← [(2,15) (3,35)] ←	1: 0, 2: 1, 3: 1
iteration 3	2			← [(3,35)] ←	
relaxation ← iteration 3.1		3	0: 0, 1: 5, 2: 15, 3: 20	← [(3,20)] ←	1: 0, 2: 1, 3: 2
iteration 3.2		4	0: 0, 1: 5, 2: 15, 3: 20, 4: 35	← [(3,20) (4,35)] ←	1: 0, 2: 1, 3: 2, 4: 2
iteration 4	3			← [(4,35)] ←	
relaxation ← iteration 4.1		4	0: 0, 1: 5, 2: 15, 3: 20, 4: 30	← [(4,30)] ←	0: , 1: 0, 2: 1, 3: 2, 4: 3
iteration 5	4			x = t = 4 stop:	

The minimum cost walk from s=0 to t=4 is built backwards from prev:

$t = \underline{4}$, prev[4] = $\underline{3}$, prev[3] = $\underline{2}$, prev[2] = $\underline{1}$, prev[1] = 0 = $\underline{s}$

$0 \to 1 \to 2 \to 3 \to 4$, cost = dist[4] = 30

Dijkstra's algorithm. applied backwards from t to s



$s=0, t=4$

	X	Y	dist: dictionary	q: priority queue	next: dictionary.
initializ.			[, , , , 0]	(4,0)	
iteration 1	4.				
iteration 1.1		2.	[, , 20, , 0]	(2,20)	[, , 4,]
iteration 1.2		3	[, , 20, 10, 0]	(3,10) (2,20)	[, , 4, 4,]
iteration 2	3			(2,20)	
relaxation ← iteration 2.1		2.	[, , 15, 10, 0]	(2,15)	[, , 3, 4,]
iteration 3	2				
iteration 3.1		0	[35, , 15, 10, 0]	(0,35)	[2, , 3, 4,]
iteration 3.2		1	[35, 25, 15, 10, 0]	(1,25) (0,35)	[2, 2, 3, 4,]
iteration 4.	1			(0,35)	
relaxation ← iteration 4.1		0	[30, 25, 15, 10, 0]	(0,30)	[1, 2, 3, 4,]
iteration 5	0			$X = 0 = s$ ⟹	stop.

The minimum cost walk from $s=0$ to $t=4$ has the cost $= dict[0] = 30$ and it is build by using next dictionary:

$\underline{s=0}$, $next[0] = \underline{1}$, $next[1] = \underline{2}$, $next[2] = \underline{3}$, $next[3] = 4 = t$

$0 \xrightarrow{5} 1 \xrightarrow{10} 2 \xrightarrow{5} 3 \xrightarrow{10} 4.$