

## Graphs Part 4

### Dijkstra's shortest path algorithm

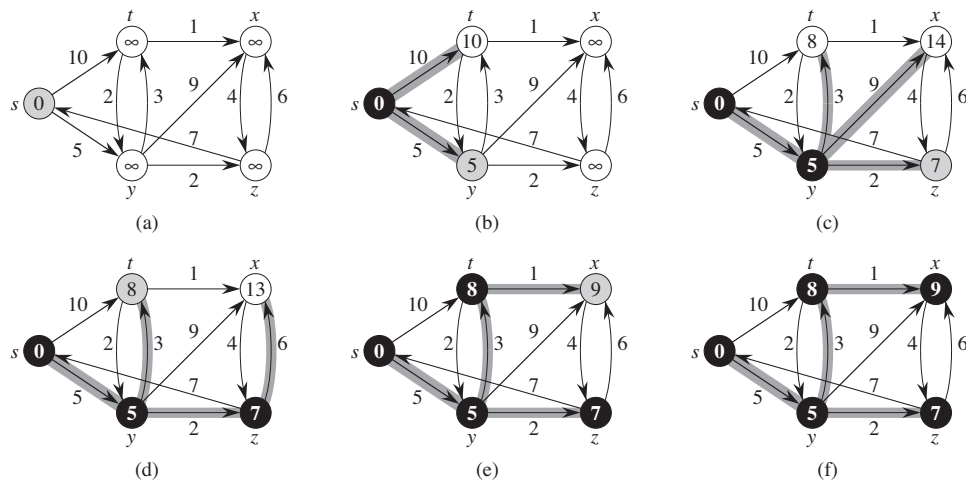
1. This algorithm finds a single source shortest path on a directed graph, with weights for the edges (for instance, weights could be driving distances between locations).
2. This is a greedy algorithm, and the weights must be *non negative* (otherwise, for instance if we want to represent positive profit and negative loss, we can use a “cousin” algorithm, Bellman Ford, which we do not discuss here, and uses dynamic programming).
3. Remember that Breadth First Search (BFS) also finds a shortest path, but in the case of unweighted edges. Dijkstra can be seen as a generalization of BFS.

Approach: We first describe the main approach, and then look at an example from the text-book, which will make it more concrete.

The main idea is that we maintain a set of vertices  $S$  whose final shortest path lengths have already been determined. Each time we consider the not yet discovered vertices in the graph, and all edges going from a discovered vertex ( $u$ ) to an undiscovered vertex ( $v$ ). We choose an undiscovered vertex with an edge from  $u$  to  $v$ , that gives the shortest path length. The length from  $u$  to  $v$  for each vertex  $v$ , is given by the length of  $u$ , plus the weight between  $u$  and  $v$ .

In the initialization, we just include source node  $s$  in the set of discovered nodes, and set its length to 0. All other lengths are initially infinity. Then we keep expanding set  $S$  of discovered nodes in a greedy manner, as in the example figure below.

In the figure, the black vertices at each step are those vertices added to set  $S$ . Initially, only  $s$  is in set  $S$ .  $s$  can go to  $t$  (length 10) or  $y$  (length 5), and  $y$  yields the shortest path. Now both  $s$  and  $y$  are in set  $S$ . We consider all possibilities from set  $S$  (vertices  $s$  and  $y$ ) to other vertices. We already have the  $s$  to  $t$  (length 10) stored and we also look at  $y$  to  $t$  ( $5 + 3 = 8$ );  $y$  to  $z$  ( $5 + 2 = 7$ ); and  $y$  to  $x$  ( $5 + 9 = 14$ ). The shortest greedy choice is  $y$  to  $z$  (length 7), so now  $z$  is added to our set of discovered nodes  $S$ . And so on; see figure below.



**Figure 24.6** The execution of Dijkstra's algorithm. The source  $s$  is the leftmost vertex. The shortest-path estimates appear within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set  $S$ , and white vertices are in the min-priority queue  $Q = V - S$ . (a) The situation just before the first iteration of the **while** loop of lines 4–8. The shaded vertex has the minimum  $d$  value and is chosen as vertex  $u$  in line 5. (b)–(f) The situation after each successive iteration of the **while** loop. The shaded vertex in each part is chosen as vertex  $u$  in line 5 of the next iteration. The  $d$  values and predecessors shown in part (f) are the final values.

Run time summary: We noted in class that we go through each vertex once, and then for each vertex we need to look at its adjacency list. If there are  $n$  vertices and  $m$  edges, we have the usual  $(n + m)$  number of operations. However, each operation takes time, since we need to find the minimum amongst all possible edges. This can be done efficiently by extracting the minimum from a Queue, with each such minimum taking  $O(\log n)$  time. Overall we have  $O((n+m) \log n)$