CS344: Design and Analysis of Computer Algorithms

Homework 6

Group Members: Stephen Kuo, Derek Mui

4.1) Suppose Dijkstra's algorithm, is run on the graph, starting t node A.

(a) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm
(b) Show the final shortest path tree

Solution:
a)

A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
0	1	∞	∞	4	8	∞	∞
0	1	3	∞	4	7	7	∞
0	1	3	4	4	7	5	∞
0	1	3	4	4	7	5	8
0	1	3	4	4	7	5	8
0	1	3	4	4	6	5	6
0	1	3	4	4	6	5	6
0	1	3	4	4	6	5	6

b)

$$
\begin{array}{ccccccc}
 & 1 & & 2 & & 1 & \\
A & \rightarrow & B & \rightarrow & C & \rightarrow & D \\
\downarrow 4 & & & & \downarrow 2 & & \\
E & & F & \leftarrow & G & \rightarrow & H \\
 & & & 1 & & 1 &
\end{array}
$$

4.5) Often there are multiple shortest paths between two nodes of a graph. Give a linear-time algorithm for the following task.

Input: Undirected graph G = (V, E) with unit edge lengths nodes $u, v \in V$
Output: The number of distinct shortest paths from u to v

Solution:

Breadth-first search gives us one shortest path from u to v or to any other vertex reachable from u. To compute the number of shortest paths, we need to

make an addition to the algorithm.
Assume that the distance from u to some vertex x is 10. Then x has at least one neighbor y_1 whose distance from u is 9. Let us say that there are two more neighbors y_2, y_3 at distance 9 from u. Any shortest path from u to x can be constructed by choosing $y \in \{y_1, y_2, y_3\}$ as the second last vertex, taking a shortest path from u to y, and adding the final edge (y, x). Each choice y_1, y_2, or y_3 gives a distinct set of paths. The total number of shortest paths from u to x is then the sum of the number of shortest paths from u to y_1, y_2, and y_3. Generally, once we know the number of shortest paths from u to all vertices at distance d, we can compute the number of shortest paths to the vertices at distance d+1 without actually enumerating the paths.

We use BFS to go through the vertices in the order of increasing distance from the start vertex. The algorithm count-shortest-paths below performs a BFS starting from u. The number of shortest paths from u to any vertex x, denoted by paths [x], is initialized to 0, except that there is 1 shortest path of length 0 from u to u. On line 14, we accumulate paths [x] by summing paths [y] for all neighbors y of x such that y is on a shortest path between u and x. In the end, paths [v] contains the number of shortest paths from u to v, or zero if v is unreachable from u.

As a side effect, the algorithm finds the number of shortest paths from u to all vertices and not just to v , but this does not affect the worst-case run time. The run time of BFS is $O(|V| + |e|)$, and count-shortest-paths stays in the same bound, assuming that integer addition is constant-time. Note that we might need to break this assumption in practice because the number of shortest paths can grow exponentially in the number of vertices.

Computing number shortest paths using BFS

```
1 function count-shortest-paths(G, u, v);
2 for all x ∈ V do
3     dist[x] ← ∞;
4     paths[x] ← 0
5 end
6 ;
7 dist[u] ← 0;
8 paths[u] ← 1;
9 Q ← [u]
10 while Q is not empty do
11     x ← EJECT(Q);
12     for all edges (x, y) ∈ E do
13         if dist[y] = dist[x] - 1 then
14             paths[x] ← paths[x] + paths[y];
15         end
16         if dist[y] = ∞ then
17             INJECT(Q, y);
```

```
            dist[y] ← dist[x] + 1;
        end
    end
end
;
return paths[v]
```