Math 320 Homework 2.5

Chris Rytting

October 5, 2015

2.24

One implements BFS, but once the node is found, we count the depth of the node (i.e. the distance from the root node) and retrace our steps. If the shortest path is from A to B to C, then once we find C, we find that we got to C from B and to B from A, and we have found the shortest path.

2.26

$$S = A \quad Q = D, C, E, F, B$$

Starting at node A, we have that $d(A) = 0, w(A, B) = 3, w(A, F) = 6$, and $w(A, B) = 3 < 6 = w(A, F)$, so we add the edge (A, B).

$$S = A, B \quad Q = D, C, E, F$$

Then we have at node B, with neighbors A, F, C, that $A \in S, \quad F, C \in Q$, and that $w(B, F) = 4 < w(B, C) = 5$, so we add the edge (B, F).

$$S = A, B, F \quad Q = D, C, E$$

Then we have at node F, with neighbors A, B, C, E, that $A, B \in S, \quad E, C \in Q$, and that $w(F, E) = 4 < w(F, C) = 5$, so we add the edge (F, E).

$$S = A, B, F, E \quad Q = D, C$$

Then we have at node E, with neighbors F, C, D, that $F \in S, \quad C, D \in Q$, and that $w(E, C) = 1 < w(E, D) = 2$, so we add the edge (E, C).

$$S = A, B, F, E, C \quad Q = D$$

The only node left now is D, so we add it to the MST to complete the algorithm.

$$S = A, B, F, E, C, D \quad Q = \emptyset$$

2.27

Let G be our graph, S be the set containing all nodes to which we have already found a path and a priority queue Q contain all the nodes not in S, prioritized by the length of the current shortest path to those nodes. Let

$$S = \emptyset, \quad d(v) = \infty \forall v \in Q, \quad Q = A, B, F, C, E, D$$

Starting at arbitrary node A,

$$S = A, \quad d(A) = 0, \quad d(v) = \infty \forall v \in Q, \quad Q = B, F, C, E, D$$

For the first stage, let $u = B$ and then that $u = F$, then we have that

$$S = A, B, F \quad d(A) = 0, d(B) = 3, d(F) = 6, \quad d(v) = \infty \forall v \in Q, \quad Q = C, E, D$$

Now, the next nodes to look at from F are the neighbors $v = E$, where $d(F) + w(F, E) = 6 + 4 < \infty \implies d(E) = 10$ and $v = C$, where $d(F) + w(F, C) = 6 + 5 < \infty \implies d(C) = 11$

$$S = A, B, F, C, E \quad d(A) = 0, d(B) = 3, d(F) = 6, d(C) = 11, d(E) = 10, d(v) = \infty \forall v \in Q, \quad Q = D$$

Now, the next nodes to look at from B are the neighbors $v = F$, where $d(B) + w(B, F) = 3 + 4 \not< 6 \implies d(F) = 6$ and $v = C$, where $d(B) + w(B, C) = 3 + 5 < 8 \implies d(C) = 8$

$$S = A, B, F, C, E \quad d(A) = 0 d(B) = 3, d(F) = 6, d(C) = 8, d(E) = 10, d(v) = \infty \forall v \in Q, \quad Q = D$$

Now, the next nodes to look at from C are the neighbors $v = F$, where $d(C) + w(C, F) = 8 + 5 \not< 6 \implies d(F) = 6$,
$v = E$, where $d(C) + w(C, E) = 8 + 1 < 10 \implies d(E) = 9$,
and $v = D$, where $d(C) + w(C, D) = 8 + 1 < \infty \implies d(D) = 9$

$$S = A, B, F, C, D, E \quad d(A) = 0, d(B) = 3, d(F) = 6, d(C) = 8, d(E) = 9, d(D) = 9, \quad Q = \emptyset$$

Now, the next nodes to look at from E are the neighbors $v = F$, where $d(E) + w(E, F) = 8 + 4 \not< 6 \implies d(F) = 6$,
$v = C$, where $d(E) + w(E, C) = 8 + 1 \not< 8 \implies d(C) = 8$,
and $v = D$, where $d(E) + w(E, D) = 9 + 2 \not< 9 \implies d(D) = 9$

$$S = A, B, F, C, D, E \quad d(A) = 0, d(B) = 3, d(F) = 6, d(C) = 8, d(E) = 9, d(D) = 9, \quad Q = \emptyset$$

Now, the next nodes to look at from D are the neighbors $v = E$, where $d(D) + w(D, E) = 9 + 2 \not< 0 \implies d(E) = 9$,
and $v = C$, where $d(D) + w(D, C) = 9 + 1 \not< 8 \implies d(C) = 8$,

$$S = A, B, F, C, D, E \quad d(A) = 0, d(B) = 3, d(F) = 6, d(C) = 8, d(E) = 9, d(D) = 9, \quad Q = \emptyset$$

2.28

Dijkstra's algorithm chooses the closest nodes to process first. This means that if there is a node v that we haven't visited yet, then we will have to go at least one node deeper to visit it. The contrary would be impossible unless the edges had negative weights. The result is obvious, then, that u's minimum distance is less than or equal to v's.

2.29

Dijkstra's Algorithm simply finds the shortest path to any given node w from an arbitrary initial node v by coming up with the "cost" of every single possible path from w to v and summing the weights (can be thought of as cost or distance to travel from one node to the next) of the edges that comprise that path.
BFS, on the other hand, finds the most direct route to a path from the origin without taking the weights of the edges into account. The only instance in which the most direct route would not be cheaper than another, more roundabout way, is if the cumulative cost of the edges that comprise the most direct route were greater than the cumulative cost of the edges that comprise the more roundabout route. In order for this to be the case, the individual edges of the direct route must be more costly than the individual edges of the roundabout route, on average. If the weights are the same, though (even on average), this is obviously not the case, and Dijkstra's Algorithm will yield the same results as BFS.