Shortest Unweighted Path

Remember that the (unweighted) length of a path in a graph is the number of edges the path contains.

Consider a directed graph and let S be any specific node in this graph. We now give an algorithm for finding the shortest path from S to every other node in the graph.

The algorithm maintains a **queue** of nodes, which initially contains only S.

It gives each node a *value*, which ultimately will be the length of the shortest path to it.

Finally, it gives each node a predecessor node in its path from S. This allows us to find the path from S to the node the way we found the path from the entrance to the exit of a maze in Lab 3.

Initially make the value of each node except S be "INFINITY". If you are thinking of a Java implementation, INFINITY can be either Integer.MAX_VALUE or Double.MAX_VALUE, depending on how you want to think of the values.

If you change the logic a bit you could also initialize the distance to -1. What is important is being able to determine if a node has an assigned distance or the default distance.

Make the value of S be 0.

Now perform the following steps until the queue is empty.

a) Remove the head of the queue. Call this node X.

b) For each outgoing edge from X to another node Y, if Y's value is INFINITY make the new value of Y be the (value of X) + 1, make the predecessor of Y be X, and add Y to the queue.

   If Y's value is less than INFINITY then it will be no more than the (value of X)+1, so we ignore Y, we don't change its value and we don't add it to the queue.

Note that, because we are using a queue, the algorithm first finds all of the paths of length 0, then all of the shortest paths of length 1, then all of the shortest paths of length 2, and so forth.

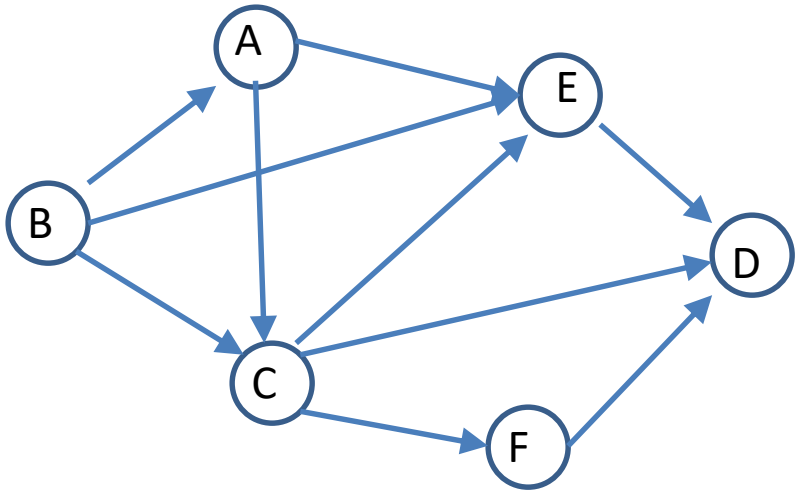
Now, how do we know this algorithm gives each node its correct distance from S?

I claim that if node X has distance n from S then the value this algorithm assigns to X is n. This is certainly true when n is 0 or 1. For other nodes let $S=X_0->X_1->X_2->...->X_n=X$ be a path of length n to X.

For each node X_t on this path, the algorithm couldn't give X_t a value greater than t because it assigns values in increasing order. If it gave X_t a value less than t, that would give us a path from S to X_t of length less than t. Substituting that for the start of our path would give a path to X of length less than n which contradicts the assumption that the real distance from S to X is n. So the algorithm must give each X_t the value t. In particular, it gives X, which is also X_n the correct value n.

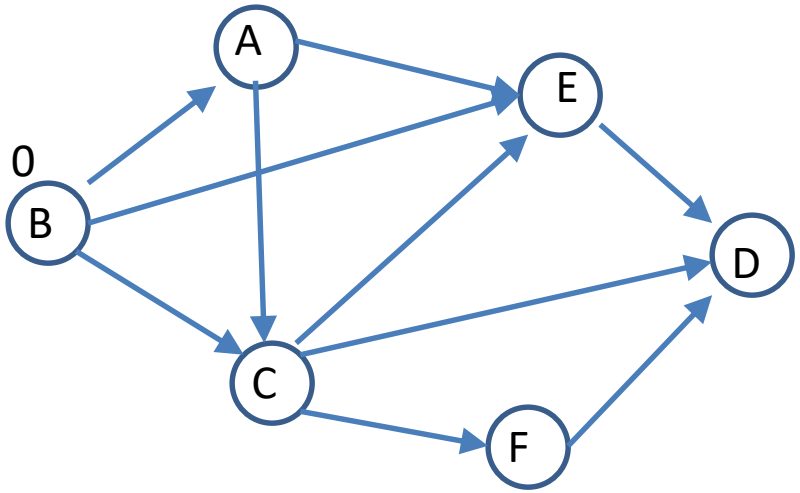
Note that this algorithm visits every edge in the graph and so it runs in time O(|E|).

Example: let's run the algorithm on the following graph, where we take B as our source node.



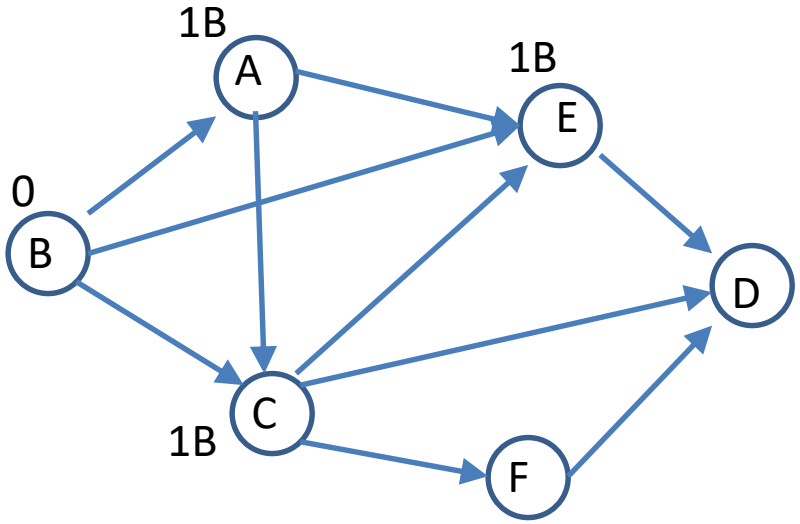
I will write next to each node its value (distance from S) and its predecessor when we compute them. Unmarked nodes still have value INFINITY.

Initially we make the source node B have value 0 and put it in the queue.



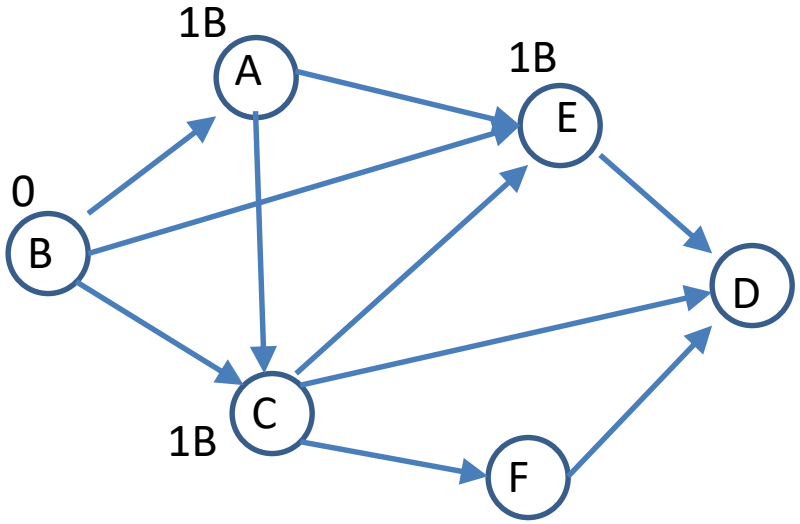
Queue: [B]

For the next step we remove B from the queue. Its outgoing edges all go to nodes with value INFINITY. We give them value 1 and add them to the queue:



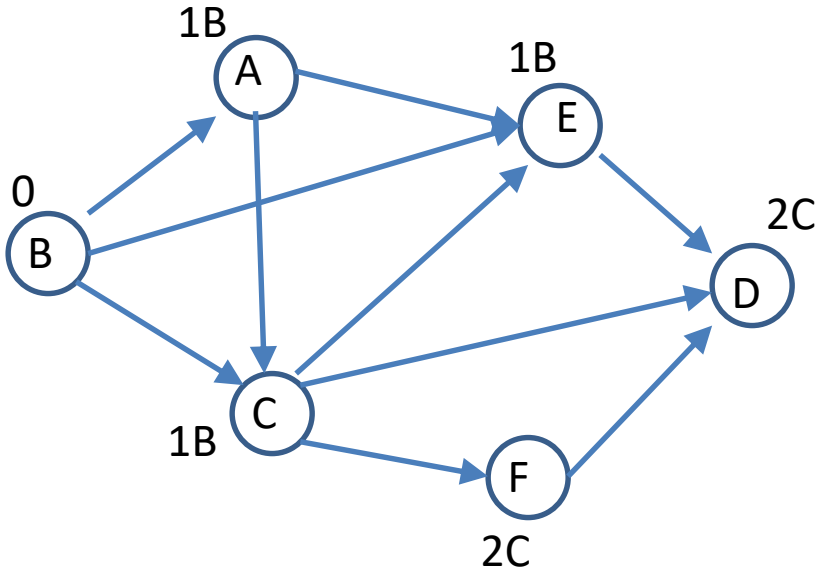
Queue: [A C E]

For the next step we remove the head of the queue, which is A with value 1. It has outgoing edges to C and E, both of which have known values, so we ignore them.



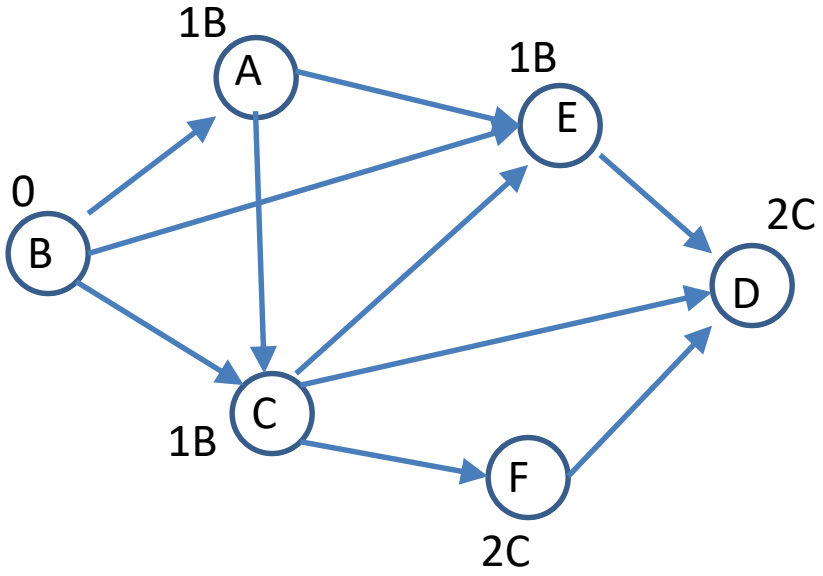
Queue: [C E]

Now the head of the queue is C. It has outgoing edges to E, which has a known value, and also to D and F. We give D and F value 2 and add them to the queue.



Queue: [E D F]

At this point each node in the graph has a value and a predecessor, so for the last three steps we will remove E, D, and F from the queue and nothing will be added.

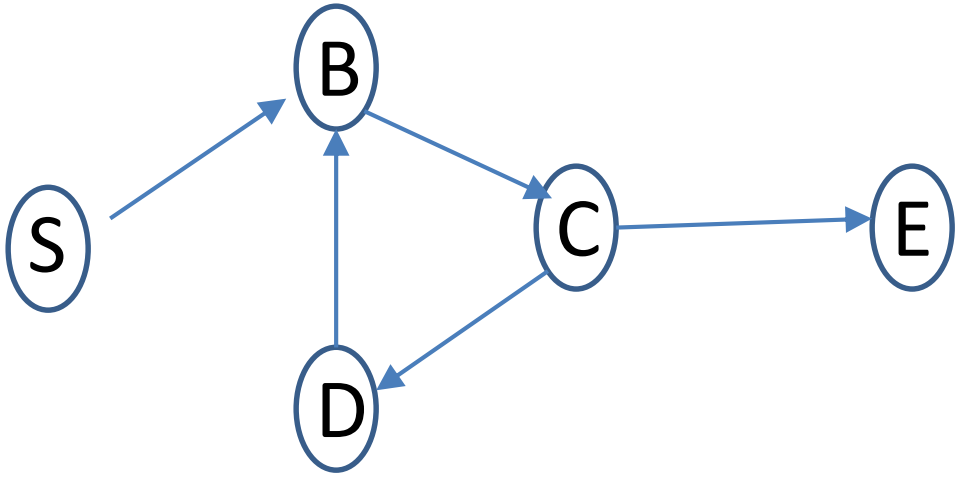


Note that we can find the path from B to any node X by walking backwards along the predecessors from X to B. This is the same way we found paths from the entrance to the exit of a maze in Lab 3.

Question: Would this algorithm still work if we replaced our queue with a stack?

A. Yes, it will still find shortest paths
B. It will find paths, but not necessarily shortest paths.
C. It will not necessarily find the path from S to every other node.

Question: Does this algorithm work if the graph has a cycle?



A. Yes
B. No