Mahitha Valluru

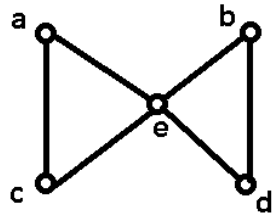
CS4800: Algorithms and Data

Professor Or Beit Aharon

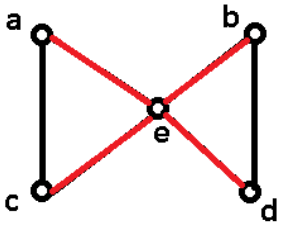
Homework 6:

1.

a. The graph is below:

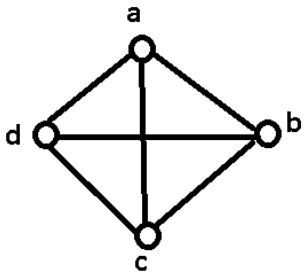


The spanning tree T would be:

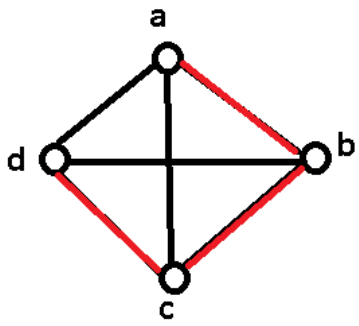


With DFS, we will not be able to reach this spanning tree. If we started at vertex a, the spanning tree would go to vertices c, e, b, and d and this pattern would follow with either b, c, or d. If it started at vertex e, it would not be able to span the whole tree. Therefore, the spanning tree T shown above in red will not be able to be constructed by DFS.

b. The graph is below:



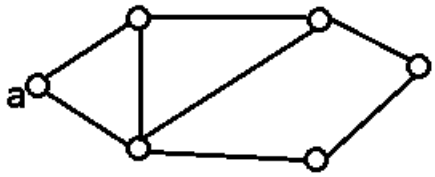
The spanning tree T would be:



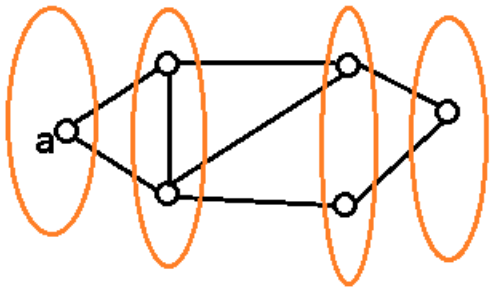
With BFS, you will not be able to construct the spanning tree T shown above. This is because, no matter which vertex a, b, c, or d that you start on, you will always take the three edges that are connected to you and that would be the spanning tree with BFS. It will never be able to make the tree T shown above.

2.

a. Imagine there is a graph that looks like this:

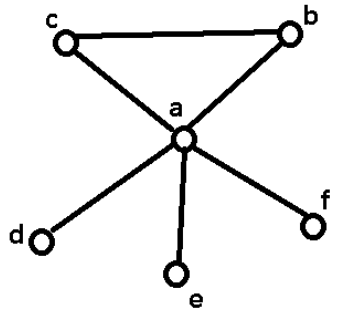


You can split the edges into levels based on how much time it will take for BFS to find it. Like so:



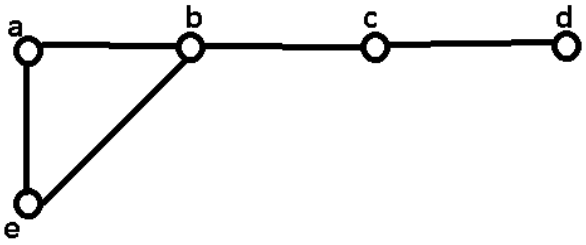
If you start at vertex a and find an edge from vertex u to vertex v, if v is the same distance (meaning in the same circle) or in a distance plus one (the next circle) and vertex v is gray, then the edge from u to v is a cross edge and it is a cycle. If vertex v is gray, that means it has been visited already and therefore any connection between your current vertex and a vertex that has already been visited will be a cross edge and there will be a cycle.
Therefore, if this is not the case, then there will be acyclicity.

b. No, for some graphs, BFS finds a cycle faster and for others DFS finds it faster. Here is an example of a graph where DFS finds the cycle faster:



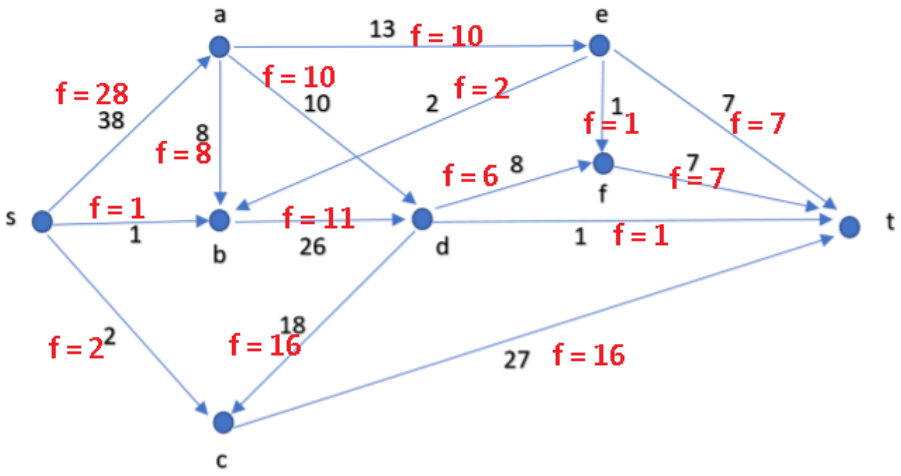
If we start at vertex a, it will go to vertex b, which will then go from b to vertex c and then realize that there is a cycle.
If this was BFS, it would have started at vertex a then go to vertex b, then go from a to vertex c, then from a to d, then from a to e, and finally from a to vertex f. BFS will never be able to find the cycle.

An example of a graph where BFS finds the cycle faster is:



If we start at vertex a, it will go to vertex b. Then it will go from vertex a to vertex e. Then it will go from vertex b to vertex c. Then it will check the edge from vertex e to vertex b, but since vertex b has already been found, it cannot go there and has found the cycle.
If this was DFS then it could go from vertex a to vertex b then to vertex c then to vertex d before stopping. It would not be able to reach vertex e and thus will not be able to find the cycle.

3. Here are the max flows of the graph:



The maximum flow is $28 + 1 + 2 = 31$.
We can find this by going over the low network and systematically taking the minimum cut each step.
At every step, we want to create a residual graph G_f. To do so, you have to first find an augmenting path p_a with a given minimum capacity c_p. That is, c_p is the lowest capacity of all the edges along path p_a.
For each edge with endpoints (u, v) in p_a, increase the flow from u to v by c_p and decrease the flow from v to u by c_p. Sometimes this will make necessary a new edge in the backward direction. This increases the flow from the source to the sink by exactly c_p. This results in the creation of a residual graph.
This is repeated until no augmenting paths are left. After that, we denote all vertices reachable from the source s ad V^c. We will add all flows remaining in V^c and this will be our maximum flow.

4. Step 1 – With all of the vertices V in G, split every vertex into 2 vertices v_{in} and v_{out}.
   Step 2 – Between each of these 2 vertices v_{in} and v_{out}, add one edge $(v_{in},\ v_{out})$ with capacity of the original edge $c(v)$.
   Step 3 – Convert every edge (u, v) to an edge (u, v_{in}) and every edge (v, w) to an edge (v_{out}, w).
   Step 4 – Run the Ford-Fulkerson algorithm and return the maximum flow.

   The capacity of this new edge is equal to the capacity of the vertex, and the flow is unchanged because all incoming edges go into v_{in} and all outgoing edges go out of v_{out}. This implies that the flow is unchanged and therefore the Ford-Fulkerson algorithm will work.