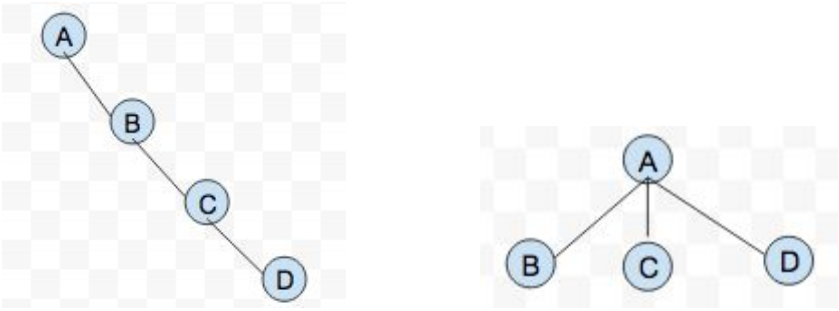
Sheetal Parikh
Foundations of Algorithms
EN.605.621.81.SU19
Homework #5

1. *Suppose we have a connected graph G = (V, E), and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at u and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u and obtain the same tree T. Prove that G = T. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)*

We need to prove that if a BFS and DFS are completed at a specific vertex $u \in V$ from a connected graph $G = (V, E)$ and we obtain the same tree T, than $G = T$. If $G = T$, this also means that G will not contain any edges that do not belong to T. In order for both a BFS and DFS to produce the same tree (same order), T, then $G = (V, E)$ would be a tree that looks similar to the following:



A DFS starts at the root and visits the children nodes before the sibling nodes, traversing down the depth of the graph. A BFS starts at the root and visits the graph one level of sibling nodes before the children nodes, traversing the width of the graph. In order to ensure that both traversals obtain the same tree, T, the starting graph $G = (V, E)$, would be a tree with a width or depth of 1. Because G would be a tree, than the BFS and DFS output T, would contain the same number of nodes/vertices as G, and would both have |V| - 1 edges. If G is not a tree, then it would be a connected graph with a cycle and the output of both the algorithms would differ as the structure of the graph would no longer be able to dictate the order of traversal for both searches.

Now that we know that G must be a tree as shown above, in order to prove that $G = T$, let's make an incorrect assumption that G contains an edge (a, b) that is not in T. In a DFS, the node a would be an ancestor of b. In a BFS, the node a and b would have a difference of one level, as per the definition of BFS. Because we know that the BFS and DFS output the same tree T and must look like the above, we conclude that nodes a and b, would have to be ancestors that have a difference of one level. Therefore, edge (a, b) must also exist in T and so $G = T$.

2. *Design an algorithm that answers questions of this type: given a collection of trace data, the algorithm should decide whether a virus introduced at computer C_a at time x could have infected computer C_b by time y. Prove that the algorithm runs in time O(m). Also, prove the correctness of your algorithm.*

We are given that we have *m* triples total that are ordered in sorted time and each pair of computers communicates at most once. Because we want to prove that C_a infected C_b , we can assume that C_a is the only infected computer. We can represent the trace data as an undirected graph represented by an adjacency list in which the computers would be the vertices and the edges would be the time that the computers communicated. Since it is more likely that the graph is sparse because some computers may not communicate with others, the adjacency list would ensure access to each vertex or computer's adjacency list in constant time. If there is communication between a pair of computers there is an edge and if there is also a possible path to another computer within the range of communication time, this other computer could be infected if the first computer is infected.

We want to only look at the triples that have communication times, *t,* in the range of *x* or *y* meaning that $x \leq t \leq y$. If C_a communicated with C_b in this time range than, C_a had an open connection to infect C_b. Since the data is sorted, we can easily check the list for the first triple in which computer C_a has a communication time of ≥ *x.* This particular triple will be our starting vertex. We can use BFS at this C_a to search through the graph to determine if there is a path to C_b at a time that is ≤ *y* meaning that C_a could have infected C_b by time *y.*

To prove the correctness of the algorithm, we want to prove that the algorithm will be able to determine that C_a , infected at time *x,* would be able infect computer C_b by time *y*. At the start of the algorithm, we first search the triples for which C_a has a communication time of ≥ *x* , and the first triple we find that satisfies this condition, becomes our starting vertex. This ensures that we are only running a BFS through a C_a that satisfies the initial condition. The BFS searches the graph until if finds a path that is within a communication time of ≤ *y.* The adjacency list records whether a path exists from C_a and would also have the weights of the edges or communication time of each pair of computers. If a path isn't found , the BFS will continue to search the graph until it exhausts all nodes ensuring termination of the algorithm. In this situation, the algorithm will return false (meaning no infection). However, if a path is found, then the algorithm will return true meaning that C_a infected at time *x,* would be able infect computer C_b by time *y.*

Because we are implementing the adjacency list to represent the graph of trace data, constructing the graph will take will take constant $O(m)$ time. We can assume the "m" to be the total nodes and edges of the graph. The algorithm uses BFS which will run in $O(m)$ time as it will traverse every node and edge or every element of the graph. Therefore, overall the algorithm will have a running time of $O(m)$.