Homework 2

Due date: 1/20/2021 right before class

Problem 1

You are given an undirected tree $T = (V, E)$. In this problem we will find the diameter of the tree in 3 different ways.

a.

Write the following:

1. A recursive algorithm to find the *diameter of the graph, based on the leaf pruning method discussed in lecture.
2. An iterative algorithm to find the *diameter of the graph, based on the recursive version you came up with and the leaf pruning method discussed in lecture. Compute the complexity of this version. There exists a $O(|E|)$ iterative implementation of the recursion.

b.

Show how you can find the *diameter of the graph in $O(|E|)$ time by running BFS twice.

c.

Find the diameter of the graph by **rooting the tree, and a recursive call from the root asking for the diameter of each subtree hanging from each of its child.

*Diameter is defined as the longest path (number of edges) between any two nodes $v_i, v_j \in V$.
** Rooting a tree is choosing a vetrex in the tree to be called a root. The neighbors of the root are called the children of the root. Recursively, the child is now a root for the fragment of the tree it resides in, after the removal of the root from the tree.

0.1 Solution 1

a.

```
FindDiameter(V, E):
    if |V| == 1:
        return 0
    else if |V| == 2 and |E| == 1:
        return 1
    E', V' <- Run BFS(V,E) to find leaves and their edges attached
    to them
    return (2 + FindDiameter(V/V', E/E')
```

Listing 1: Recursive algo

```
FindDiameter(V, E):
    diameter = 0
    While |V| > 2:
        V', E' <- Run BFS(V,E) to find leaves and  their edges
        diameter += 2
        V = V/V'
        E = E/E'
    if |V| == 2 and |E| == 1:
        diameter += 1
    return diameter
```

Listing 2: Iterative algo

Complexity in the iterative algorithm entails running O(1) operations and an expensive BFS operation that will cost us O(E) time for a total of O(E).

b.

Attached in the end.

c.

```
FindDiameter(root):
    if root has no children:
        return 1, 1
    if root.left_child:
        left_max, combined_left = FindDiameter(root.left_child)
    else:
        left_max, combined_left = 0,0
    if root.right_child:
        right_max, combined_right = FindDiameter(root.right_child)
    else:
        right_max, combined_right = 0,0

    return max(left_max, right_max), max(combined_left,
    combined_right, left_max+right_max)
max1, max2 = FindDiameter(v_i)
result = max(max1,max2)
```

Listing 3: Recursive algo

For each subtree, we want to find if the max diameter goes through the root between two subtrees, or not. Additionally, we want to send the single path (path that does not go through the two node subtrees) up the recursion tree. So for each subproblem, we have two responses to return.

1. the longest single path, that goes through the children and ends in the current root
2. the longest path that combines the two single paths from the left and right children, OR, the combined path from a lower recursive step if it is longer then the current one.

Problem 2

a.

Given an undirected graph $G = (V, E)$ and a subgraph $T = (V^{'}, E^{'})$ where $V^{'} = V, E^{'} \subseteq E$, that is a tree. Design an algorithm to find whether T could have been the output of running **DFS** on the original graph. from some starting vertex.

b.

As the above. But now, design an algorithm to find whether this subgraph T could have been the output of running **DFS** on the original graph.

0.2 Solution 3

0.2.1 a

1. Pick a leaf node v in $T = (V^{'}, E^{'})$ and find the same node in $G = (V, E)$.
2. Start running DFS on G from node v.
3. At each iteration of DFS, instead of arbitrarily selecting the next unexplored node to visit, select a node that is attached to v_i in T.
4. If the same tree could be produced in the end, then return true.

Complexity includes running DFS which takes O(E) time, and selecting a correct node to explore next in the DFS recursion based on T. For this step we will have to potentially go over the list of neighbors for each node. Notice that this step will cost a total of O(E) for the entire graph.

To prove this we will use the fact that DFS explores an arbitrary node in the case where we have more than one unvisited neighbor for some node. Using the given tree T as a "map", we direct the exploration in the direction dictated by T. If the directed exploration is able to reconstruct T than we have shown that DFS could produce it. If it did not, than there is no way DFS could have produced the tree T.

0.2.2 b

1. For each node v in $T = (V^{'}, E^{'})$ Run BFS starting from that node in G
2. If the BFS run produces neighbors that are not in T, stop and start from the next node.
3. Compare the output tree and T by running BFS from both.
4. Return true if found the same tree.

Complexity will be running BFS from each node which will cost us $O(VE)$

Our proof is simple. We are searching for the root, and therefore brute forcing the search by running multiple BFS searches. If a root node exists that produces the tree T then we will find it in our search.

Problem 3

Given a bipartite graph $G = (M, W, E)$ where $|M| = |W| = n$, Where the node names in M are $\{1, 2, \ldots n\}$, and so are the nodes in W (M stands for Men and W stand for Women).

There are undirected edges between M and W. But, the degree of nodes in M are 1) The whole graph contains exactly n edges).

1. If in G each node in W has degree 1 or more, show that the graph is a perfect match.
2. If the graph is not a perfect match, we want to find the largest subset L from $\{1, 2, \ldots, n\}$ such that the subgrapgh induced by the Men in L and the Women in L constitutes a perfect match in the induced subgraph. Your algorithm should be linear time.

For example: if we have 2 men m_1, m_2 and 2 women w_1, w_2 and edges e_1, e_2 going from $< m_1, w_1 >, < m_2, w_1 >$. A subset $L = \{1\}$, since the only possible solution will be $< m_1, w_1 >$.

0.3 Solution 3

Notice that if a woman w_i in W has no incoming edge, then the man corresponding to her m_i cannot be in our final set L. We can cascade and prune in the following way:

1. Select a woman w_i that has no incoming edges
2. Remove nodes w_i and m_i
3. Remove outgoing edges from m_i
4. Repeat the process
5. L is the set of indices left in W

Our algorithm runs in O(E) time, since it can at most, remove all edges in a graph and no more.

Our proof will explain why our algorithm produces a perfect matching over a set of indices. Its clear that if a certain woman w_i does not have an incoming edge it is not part of the perfect match. It follows that her corresponding man m_i cannot be in the match as well, based on the definition in the question. It is left therefore to show that since m_i cannot be in the match, removing outgoing edges from it will either a) create new women with no incoming edges, and the process continues, or b) will remove an edge from a woman that has more incoming edges which will either be removed later, or, if is kept will be part of the match. once all men and woman have been eliminated based on the criteria in our algorithm, we notice that all woman that are left with incoming edges and their partners are the remaining subgroup that is perfectly matched.