CPSC413 Assignment 3

Yifan Chen 30073072

October 2020

1 Part1.

a.

1. 1,4,6,5,7,8,3,2
2. 1,2,3,4,6,5,7,8
3. Yes G is connected, for the graph to no longer be connected only the edge (4,6) needs to be removed.
4. Yes G contains a cycle. By definition, a cycle is a path with length greater than zero from some vertex to itself. An example cycle in G would be the path 1,2,3,4,1.
5. By definition, a tree is a connected acyclic graph, and since we have proved that G contains a cycle in q4, we can ascertain that G is not a tree.

b.

1. In a complete graph, any vertex would connected to any other vertex, so a depth first search would be able to scan through each node without ever hitting a dead end, resulting in a tree that would look like a linked list, each node connected as a single leaf to another node.
2. The breadth-first search tree would look like a dandelion, with the first vertex will be the root of the tree, with every other vertex connected as a leaf.

c. Suppose any arbitrary data structure A, which can be accessed in constant time, and contains all n vertices of G. Let A be an Array with length n. Each element n in Array A could then hold a sorted Array B of length b, which contains all b vertices that are adjacent to the n^{th} vertex. In order to check if 2 vertices are adjacent, the first vertex would be the nth element of array A which could be accessed in constant time. To find if the second vertex were adjacent, a binary search could be performed on sorted array B for the second vertex, which has a worse time performance of O(log n). The time complexity for the entire data structure would then be O(log n)

d. *Claim. let G be a graph on n nodes, where n is an even number. If every node of G has a degree of at least n/2, then G is connected.*
In order for graph G to not be connected, G must be able to be split into 2 separate groups of nodes. We prove the claim by contradiction, suppose G is able to be split into 2 groups, n - i, and n - j, where $i + j = n$, and $(n - i) + (n - j) = n$. Recall that each node of Gs must have a degree of at least n/2. Since the size of each group is lower bounded by how many connections each node must have, In this case the smallest possible groups for which G could be split into, while still maintaining each node's minimal degree of is $n/2 + 1$. The size of the 2 groups must then be $\geq (n/2 + 1) * 2 \geq n + 2, \neq n$, and by contradiction, G is not able to be split into two separate groups, and the claim is proven to be true.

2 Algorithm to detect whether a given undirected graph contains a cycle

1. Precondition: An undirected Graph G.
   Postcondition: If G should have a cycle, a single Cycle of G, in O(m+n) time, where n = nodes and m = edges, otherwise return an empty path.

2. A modified DFS algorithm, please see next page.

3. Suppose the precondition has been satisfied, and exemplar DFS search algorithm is correct. Pseudo code for depth has been removed. Assuming the original code terminates, the modified line 5 of algorithm 2 only changes the value of what is returned to be the correct postcondition of either a cycle or an empty path. Lines 4, 5, 8, 9 of algorithm 2 have been added. Line 4 assigns the parent of v to u, and line 5 keeps track of the edges between 2 nodes. These 2 lines have no effect on termination. Line 8 returns the post condition cycle, if, a path from u to v, where u is not the parent of v has been found. Since there is a path from u to v, and there exists an edge v to u, a cycle has been found and returned. If there does not exist any cycles in the graph, the execution returns to line 5 of function one, where the empty path is returned, fulfilling the post condition.

4. There are 2 cases we must prove
   case 1. A cycle is found
   In this case the worse case is if all nodes are part of the cycle. The algorithm must then visit each node and edge once, which would have the same running time as a regular DFS algorithm of O(m+n)
   case 2. A cycle is not found
   In this case the algorithm must also visit each node and edge once, resulting in the same runtime of O(m+n).

Algorithm 1 DFS (G = (V,E), s), Initialization

```
1: for each vertex u ∈ V do
2:     u = unvisited
3: end for
4: DFS − Visit(s)
5: return empty path
```

Algorithm 2 DFS-visit(u)

```
1: u = visited
2: for each v ∈ Adj[u] do
3:     if v == unvisted then
4:         Parent[v] = u
5:         path.add(u, v)
6:         DFS − visit(v)
7:     end if
8:     if v == visited and v! = parent[u] then
9:         return path
10:    end if
11: end for
12: u = done
```