(1) Adjacency Matrix Algorithm

* i represents the y-dimension of the matrix and j represents the x-dimension

Start the algorithm at the Graph G's G[0][0] index. Check to see if that slot contains a 1 or a 0. If it contains a 1, increment the i variable by 1, and if it contains a 0, increment the j variable by 1. Keep doing this until one of 2 things happen:

1) The j variable exceeds the number of vertices, then check if all of row i is 0's and all of column i is 1's except for G[i][i]. If this is the case, we have found the desired vertex and the algorithm is over

2) the i variable exceeds the number of vertices. This means that there cannot be a vertex with the desired conditions and the algorithm is over

(2) Modified DFS Path Pseudo-code

* assume there are node objects with label, adjnode list, and boolean "seen" value

```
path = []
def modDFS(G, s, t):
   alist = get_adjlist(s)
   for v in alist"
         if v.visited = False:
                  if v.value not in path:
                           if v.value != t.value:
                           path.append(v)
                           v.seen = True
                          modDFS(G, v, t)
                  else:
                          s.seen = True
                          return path
   return []
```

(3) If Td and Tb Are The Same Tree, Prove That The Graph They Came From, G, Is Already A Tree

For Td and Tb to be the same tree but the initial graph GA is not a tree, there must be some edge in G not in Td/Tb. This creates a cycle in G. If there is a cycle, DFS will handle both connected nodes on the same path and one will be the other’s ancestor. This is because, when searching, DFS will find all possible paths from one of the nodes before it goes back to start, so it will inevitaby get this edge. In a BFS search, when it accesses the cycle, it would branch at least twice from the initial node, never accessing the backedge. Clearly, both BFS and DFS handle backedges differently, so we have a contradiction and G has to be a tree already.

(4) Prove that for an undirected graph, a DFS tree cannot be smaller than the depth of a BFS tree

Proceed as though a DFS tree IS smaller in depth compared to a BFS tree. For this to be the case, the node that creates the deepest path and therefore sets the depth needs to have a shorter path found only by DFS and not BFS.

However, since BFS takes it’s connections one level at a time, if there is a shorter path in the same tree, BFS will identify it before it identifies the longer path. This longer path is, through our procession by contradiction, the path that is included in BFS and not DFS. It will identify it first by nature of the algorithm. This makes it so that any shorter path DFS could find, BFS would inevitable find it as well, giving us a contradiction. A DFS tree cannot be smaller than the depth of the BFS tree for the same graph.