Homework 5

Walker Bagley

April 4, 2024

1 Connected Graph

A simple graph has no self loops and only one edge between two nodes, therefore the total number of edges in a graph is equal to the sum of node degrees divided by 2. In this case we have $4 \cdot 2 + 4 \cdot 1 = 8 + 4 = 12$ total degrees and therefore 6 unique edges in the graph. A connected graph must contain a path between every pair of nodes in the graph, therefore requiring a minimum of $n - 1$ edges for n nodes if one were to line up all the nodes and connect each consecutive pair. In this graph containing 8 nodes, we must have at least 7 edges in a connected graph and because we have shown there are only 6 edges, this graph cannot be connected.

2 Exercise 20.1-5

Adjacency list:

```
function square(G):
    G2 = new dictionary to hold adjacency list
    for node in G:
        paths = G[node]
        for node2 in G[node]:
            paths += G[node2]
        G2[node] = paths
    return G2
```

This algorithm creates a new adjacency list $G2$ and iterates through all vertices. On each iteration, it adds the existing nodes from G (paths of length 1) in addition to adding the nodes from each of the lists for the existing connections (paths of length 2). We can see that the outer for loop will run V times. In an adjacency list, we can have a maximum of V nodes in a given list, so the inner for loop can run at most V times if every vertex is connected to the current one. To combine the lists requires at most V operations (if the node being inspected is also connected to every vertex), giving us a final time complexity of $O(V^3)$.

Adjacency matrix:

For this, we can simply multiply the adjacency matrix by itself to yield the squared adjacency matrix representing G^2. In this matrix multiplication, we do V multiplications for each cell of the resulting $V \times V$ matrix, giving us a time complexity of $O(V^3)$, though more complex algorithms exist that can reduce the complexity of matrix multiplication.

3 Exercise 21.1-6

Proof:

We can show this via a contradiction. Consider two distinct minimum spanning trees of a graph, called M and M'. This means there is at least one edge in M but not in M' and vice versa. Consider one of these edges, (u, v) which exists in M but not M'. Remove (u, v) from M, then the tree is cut into two sections, and thus must have a light edge crossing it. Let's call this light edge (y, z) and thus we arrive at two cases: either (y, z) has a lower weight than (u, v) or $(y, z) = (u, v)$ since there is a unique light edge crossing any

cut. In the first case, (y, z) should be in the MST instead of (u, v), so M cannot be a MST. In the second case, (u, v) is a unique light edge crossing the cut and therefore whichever edge crosses this cut in M' must not be a light edge. So, M' cannot be a MST since it does not contain (u, v). Thus, the spanning tree of a graph under these conditions must be unique.

Converse Counterexample:

The converse is as follows: if a graph has a unique minimum spanning tree, it must have a unique light edge crossing any cut in the graph. Consider a three node graph with nodes a, b, c. Suppose (a, b) and (b, c) both have a weight of 5 while (a, c) has a weight of 10. Then the MST obviously consists of (a, b) and (b, c) and is unique. However, if a cut is made to isolate b from a, c, then there are two edges crossing it, both of weight 5 and in the MST. Since they have the same weight, they are both light edges and thus there is not a unique light edge across this cut. So the converse cannot be true.

4 Exercise 21.1-11

Suppose the edge with a decreased weight is e. Start by adding it to T. Then there must be a cycle within T since all nodes of T were already connected, so iterate through each edge of the cycle. Keep track of the highest weight seen, edges seen, and current edge to remove while iterating so that each edge is only checked once. After all edges in this cycle have been seen, remove the one with the highest weight. Now T is once again the MST of the modified graph G.