Morgan Baccus
CptS 350
Homework # 8

Problem #1

Graph G has no loops and is therefore a tree.

There is a unique path from every vertex to another.

We will use a BFS traversal from u until we reach V.

① Start from vertex u and add all unvisited neighbors to the queue.

② Mark u as visited and pop the next element in the queue.

③ If the new element is V, then we will break.

④ Repeat 2 and 3 until V is reached or the queue is empty. If V is not reached that means u and V are not connected. If u and V are connected, then the number of unique paths is $\boxed{1.}$

Problem #2

We will use a bfs to look for paths between u and V. Determine if the edge is red or yellow during the calculation of edge distance. The color variables will be attached to each node so that the number of red and yellow edges can be counted for any path.

Let L be an empty list that stores all possible paths.
Let Q be an empty queue that stores the nodes to be examined.
For each node in G, mark as undiscovered, distance = ∞, parent = null, and counters = 0.

Start at v
Initialize distance of v as 0
add v to Q

```
While Q is not empty
    dequeue Q and call this node u (current node)
    for each vertex x adjacent to u
        If x is undiscovered
            mark x as discovered
            x's distance = parent u's distance + 1
            set x's parent to u
            If x is yellow
                x's yellow count = u's +1
            If x is green
                x's green count = u's +1
        add x to Q
    mark as explored
    add u to L
```

```
for each element in L
    If element == v'
        If v'yellow count <
            v'green count
            count ++
```

Problem #3

```
#define green 0
#define yellow 1
#define V 5

graphcolor() {

    int result[V];
    result[0] = green;
    for(int i = 1 to V){
        result[i] = -1;
    }

    for (u = 1 to V) {
        list<int>:: iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); i++)
            if (result [*i] != -1)
                available [result [*i]] = true;

        int cr;        #find first available color
        for (cr = 0; cr < V; cr++)
            if ( available [cr] == false)
                break;

        result[u] = cr;   #set to color found
        for (i=adj[u].begin(); i != adj[u].end(); ++i)
            if (result [*i] != -1)
                available [result[*i]] = false;
        if (result [u] == green) count_green ++;
        if (result [u] == yellow) count_yellow++;
    }
    if (count_green > count_yellow
        goodpath ++;

}
```

Problem #4

For G_1 use $|S'n|$ to denote number of walks in G_1 with length n. Similarly, use $|S^2 n|$ to denote the same in G_2. Then, we look at the growth rate of the two to see which one is larger.

$$\limsup_{n \to \infty} \frac{\log |S' n|}{n} = \log \lambda_1$$

$$\limsup_{n \to \infty} \frac{\log |S^2 n|}{n} = \log \lambda_2$$

Now, we compare the two:

① $m^n = m_1 \times m_2 \times \ldots \times m_n$

② $m^n[i,j]$ = total number of walks from node i to j in G with length n.

③ m^n can be approximated by when n is large

$$\lambda^n v u^T$$

λ^n ← Perron # of M

v ← left eigen vector

u ← right eigenvector

T ← Transpose

④ $M^n[i,j]$ can be approximated by $\frac{v_i u_j}{\|u\|} \cdot \lambda^n \cdot v u^T$

⑤ Total number of walks from node j in G with length n can be approximated by $\frac{u_j}{\|u\|} \cdot \lambda_i^n \cdot v u^T$ $\quad \limsup_{n \to \infty} \frac{\log |S_n|}{n} = \log \lambda$

Problem #5

For each node in U, we have to come up with a Perron number (λ_u) that satisfies:

$$\limsup_{n \to \infty} \frac{\log |C_n(u)|}{n} = \log \lambda_u$$

where $C_n(u)$ is the set of all paths in $C(u)$ with length n.

To compute λ_u, we can look at problem #4.

Problem #6

count = 0

Start with either Ø, 11, or 11Ø1. Then we either have 1101 to go to the final state or we repeat while count++