McGill University COMP360 Winter 2011 **Instructor: Phuong Nguyen**

Assignment 1 Solution

QUESTION 1

The idea of the algorithm is as follows. We start with an arbitrary vertex s, color it Red, and run BFS from there in order to color all vertices in the same connected component with s. It is necessary that all neighbors of s are colored Blue, and all neighbors of these neighbors are colored Red, etc. If at some point we detect a vertex that is colored by two different colors, then the graph is not 2-colorable.

After doing BFS at s, if there are uncolored vertices in G we choose one of them and repeat the same process.

The algorithm We will use the adjacency list representation for the graph. In this representation, each vertex v is associated with a linked list $Adj[v]$ that contains all the neighbors of v. We will use an array $Color$, where $Color[v]$ is the color of vertex v. Initially $Color[v] = null$ for all vertices v.

```
% main for-loop: do BFS while there are unvisited vertices
for v in V do
    if Color[v] = null do    % vertex v has not been visited
        % now do BFS at v
        initialize an empty queue Q
        Color[v] ← Red
        Enqueue(Q, v)
        while Q is not empty do
            u ← Dequeue(Q)    % take an element from the queue
            for each w in Adj[u] do
                if Color[w] = null do    % w has not been visited
                    if Color[u] = Red
                        Color[w] ← Blue
                    else
                        Color[w] ← Red
                    end if
                    Enqueue(Q, w)
```

```
18.                else if Color[w] = Color[u]
19.                    output NO
20.                end if
21.            end for
22.        end while
23.    end if
24. end for
25. output YES
```

Running time The total time for all "for" loops (line 10) that are executed is $\mathcal{O}(|E|)$, because each loop corresponds to an edge of G. Thus the total running time is $\mathcal{O}(|V| + |E|)$.

Proof of correctness: There are two parts.

Part I. First we show that if the algorithm rejects (output NO) then the graph is not 2-colorable. Because every two BFS trees are not connected, the coloring for each tree is independent of each other. Suppose that the algorithm outputs NO during the BFS tree T that starts at a vertex v, we will show that this tree is not 2-colorable, and hence the whole graph is also not 2-colorable. Suppose for a contradiction that T is 2-colorable, then any 2-coloring of T is completely determined by the color of v, and we can assume without loss of generality that v has color Red. We prove the following Claim by induction on the distance from a vertex u to v.

Claim: Let u be a vertex in T. Any color that u gets from the program is the color that u must get in a 2-coloring of T where v is colored Red.

Proof of Claim For the base case, the distance from u to v is 0, i.e., u is v itself. The Claim holds in this case because v is colored Red.

For the induction step, suppose that the distance from u to v is $d+1$ for some $d \geq 0$. Therefore there is a neighbor u' of u such that the distance from u' to v is d. By the induction hypothesis the color assigned to u' by the program is the color u' must get in a 2-coloring of T where v is Red. Given the color of u', u has only once choice and it's clear that this is the choice chosen by the program. This completes the induction step, and hence the proof of the Claim.

We reach a contradiction because some vertex u in T is colored both Red and Blue by the program.

Part II. Now we show that if the program outputs YES, then indeed the graph is 2-colorable. This is so because the program actually provides a 2-coloring of the graph. This is because the BFS algorithm colors every vertex of the graph, and for any edge (u, v) the two endpoints must have different colors, for otherwise the program would outputs NO.

QUESTION 2

The nondeterministic algorithm The certificate describes a mapping f such that $(f(v_1), f(v_2)) \in E_2$ whenever $(v_1, v_2) \in E_1$. Such a mapping can be given as a list of pairs of preimages and images. For example, if $v_1, v_2, \ldots, v_n$ are all vertices of G_1, then f can be given as a list of the form

$$(v_1, u_1), (v_2, u_2), \ldots, (v_n, u_n)$$

where $u_1, u_2, \ldots, u_n$ are vertices in G_2.

On input G_1, G_2 and a mapping f as the certificate, the verifier works as follows. It goes over all edges in G_1, and for each edge e verifies that $(f(v_1), f(v_2))$ is an edge in G_2, where v_1, v_2 are the two endpoints of e.

Running time For each edge $e = (v_1, v_2)$ in G_1, to find the images $f(v_1), f(v_2)$ in the worst case the verifier has to go through the list of $n = |V_1|$ pairs. Once the images are found, verifying that they form an edge in G_2 takes time at most $|E_2|$. Therefore the total running time is at most

$$|E_1| \times |V_1| \times |E_2|$$

This is a polynomial in the size of the input (G_1, G_2).

Correctness of the algorithm The verifier accepts precisely when the certificate is a mapping f as in the definition of the problem. Therefore (G_1, G_2) is a YES instance if and only if there is a certificate that makes the verifier accept.