ESO207 Programming Assignment-3 Solutions

Algorithm 1: Bipartite(G)

Data: $G = (V, E)$, given in adjacency lists form.
Result: If G is bipartite then array **mark** labels vertices in one part as 1 and vertices in the other part as 2.

```
for v ∈ V do
    mark[v] ← 0 ;                                   //Initialization
end
Q ← new Queue();
for v ∈ V do
    if mark[v] == 0 then
        if bipartite(G, v) == −1 then
            print ("Component of vertex v is Not Bipartite");
            break
        end
    end
end
```

Algorithm 2: bipartite(G, v)

Data: $v \in V$
Result: Returns -1 if graph is not bipartite otherwise returns 0.

```
mark[v] ← 1;
Enqueue(Q, v);
while notEmpty(Q) do
    x ← dequeue(Q);
    for (x, y) ∈ E do
        if mark[y] == mark[x] then
            return −1
        end
        if mark[y] == 0 then
            mark[y] ← 3 − mark[x] ;                 //Put vertex y in a part
                                                    //different from the part of x
            Enqueue(Q, y)
        end
    end
end
return 0
```

The idea of algorithm bipartite(G, v) is simple. We just put the first vertex v to be arbitrarily in one part. We now repeatedly use the simple fact that whenever a vertex has been put in one part (that is, it is marked 1 or 2) its adjacent vertices are forced to be in different part. If we ever find that a vertex is forced to be both in first and second part then we conclude that bipartite partition we are looking for is not possible.

Every vertex in G can be reached by this process if G is connected. Otherwise, we repeat the process for different connected components. This is what algorithm Bipartite(G) does.

Remarks

(i) For the assignment problem, G is assumed to be connected. So full marks for just bipartite(G, v) (or something equivalent). [Bipartite(G) is not needed].

(ii) Our code is very similar to BFS code. Marking vertex y by $1, 2$ is the same thing as finding an even/odd length path respectively from root vertex v to y.

(iii) Time complexity $O(|V| + |E|)$ follows by the same argument as for time complexity of BFS.

(iv) Instead of a queue one may also use a stack. As we only want all vertices which have been marked to be considered later (to mark vertices adjacent to them) but the order in which they are considered is not important.

(b) If G is bipartite then the partition is unique. One may write it is as (V_1, V_2) or as (V_2, V_1). This follows from rationale of the algorithm described above. There, fixing the part for v, fixes part for all the other vertices uniquely.

[Full marks, even if someone answers 2, considering (V_1, V_2) and (V_2, V_1) as different partitions]

If G is not connected, let it have k connected components. Let $\{U_i, V_i\}$ be the partition of the i^{th} connected component. An easy induction on i shows that there are 2^{k-1} different partitions.

Alternatively, one may observe that in forming a part of the bipartite partition, for each i, one has two choices either to include U_i or V_i. This however counts each partition (X, Y) twice, once as (X, Y) and once as (Y, X).

[Full marks, even if someone answers 2^k, considering 2 choices for each i, $1 \leq i \leq k$.]

——— End of Assignment ———