CMPT 307 : Solutions to Practice Problems (Chapter 3)

3.2 (a) A (1:16)
B (2:15)
C (3:14)
D (4:13)
H (5:12)
G (6:11)
F (7:10)
E (8:9)

Tree Edges: (A,B), (B,C), (C,D), (D,H), (F,E), (G,F), (H,G)
Back Edges: (D,B), (E,D), (E,G), (F,G)
Forward Edges: (A,F)
Cross Edges: none

(b) A (1:16)
B (2:11)
F (3:10)
C (4:5)
D (6:9)
E (7:8)
H (12:15)
G (13:16)

Tree Edges: (A,B), (A,H), (B,F), (D,E), (F,C), (F,D), (H,G)
Back Edges: (C,B), (G,A)
Forward Edges: (F,E)
Cross Edges: (D,C), (G,B), (G,F)

3.3 (a) The pre and post values after DFS is:
A (1:14)
C (2:13)
D (3:10)
F (4:9)
G (5:6)
H (7:8)

E (11:12)
B (15:16)

(b) A and B are sources. G and H are sinks.

(c) The topological ordering is given by the post values in decreasing order: B, A, C, E, D, F, H, G.

(d) we can switch the order of A and B, D and E, G and H and still have a valid ordering. So there are 8 of them.

3.4 i. The SCCs are found in following order: 1={C,D,F,J}, 2={G,H,I}, 3={A}, 4={E}, 5={B}.
The edge in the metagraph is: (2,1), (3,1), (3,2), (4,2), (4,3), (5,2), (5,3).

ii. The SCCs are found in following order: 1={D,F,G,H,I}, 2={C}, 3={A,B,E}.
The edges in the metagraph are: (3,1), (3,2), (2,3).

3.5 Create an empty adjacent list. For each $(u, v) \in E$, add u to v's neighbour-list.

3.7 (a) Observe that a graph is bipartite if and only if it can be properly colored using only two colors. This is because saying that the vertices can be partitioned into two sets such that no two vertices in the same set share an edge is the same as saying that the vertices can be colored using two colors such that no two vertices with the same color share an edge. Therefore, we can test bipartiteness using the DFS-based 2-coloring algorithm. That is, start at an arbitrary vertex and an arbitrary color; color the vertices in an alternate manner as we visit them; the algorithm fails if it is forced to color a vertex with the color that is already assigned to one of its neighbours.

(b) **2-colorable $\implies$ No odd cycle**
An odd cycle is not 2-colorable.

No odd cycle $\implies$ 2-colorable
Suppose that a graph G is not 2-colorable. Assume without loss of generality that G is connected. The algorithm in (a) fails to color G using 2 colors. This means that at some point during the execution it is forced to color a vertex, say v, with the color that is already assigned to one of its neighbours, say u. Since the algorithm assigns colors to the vertices in an alternative manner, we can conclude that there is a path of even length between v and u. Then this path and the edge (v, u) form an odd cycle.

(c) We need at most 3 colors for such a graph. To see this, first remove an vertex in the odd cycle. The resulted graph has no odd-cycle and hence is bipartite, which means we can color it with 2 colors. Then we add the removed vertex back and assign the third color to it.

3.10
```
explore($G$,$s$)
    push($s$)
    while stack is not empty
        $u$ = top
        visited($u$) = true
        previsit($u$)
        for each neighbour $v$ of $u$
            if $v$ is not visited
                push($v$)
        if all the neighbours of $u$ are visited
            pop($u$)
            postvisit($u$)
```

3.11 First remove $e = (u, v)$ from G. Then check if u and v are still connected. This can be done using DFS starting at u and see if it reaches v.

3.12 The statement is true. Consider the 3 possible cases that satisfy $post(u) < post(v)$:

1. $pre(v) < pre(u) < post(u) < post(v)$
2. $pre(u) < pre(v) < post(u) < post(v)$
3. $pre(u) < post(u) < pre(v) < post(v)$

In case 2, we visit u and then v, but exist u before v, so this case is not possible. In case 3, we exist u before visiting v, which is not possible because v is a neighbour of u. So only case 1 is valid, and it is easy to see that in this case u is an ancestor of v in the DFS tree.

3.14 The algorithm can be implemented as follows: compute the in-degree of all vertices; find all the vertices that have in-degree 0 (the sources); add the sources to the ordering and remove them from the graph; then continue to look for vertices whose in-degree are changed to 0 due to the removals of the sources in previous step and do the same thing for these vertices. Note that we only need to look at in-degrees of the vertices that are neighbours of the removed sources, so the running time is linear in the number of edges.

3.16 Consider the following algorithm:

1 Perform topological sorting on the graph.

2 Divide the vertices into a set of layers; each layer contains the maximal number of consecutive vertices in the topological ordering such that there is no edges between any two vertices in the same layer.

The minimum number of semesters necessary to complete the curriculum is the number of layers given by this algorithm.

3.18 Consider the prepocessing algorithm that performs DFS on the tree starting at r and computes the pre and post values of the nodes. It is easy to see that u is an ancestor of v if and only if $pre(u) < pre(v) < post(v) < post(u)$

3.21 Since an odd cycle must be within a strongly connected component, we only need to consider strongly connected graphs. Then for a general graph, we need to check each SCCs.

The algorithm works as follows: first build the the DFS-tree for the graph; for each level i, color the vertices at level i with the color (i mod 2); then output true if there is an edge between two vertices with the same color, and output false otherwise.

Claim: A strongly connected graph G contains an odd cycle if and only if above algorithm outputs true on G.

Proof:

Has odd cycle $\implies$ Algorithm outputs true
An odd cycle is not 2-colorable.

Algorithm outputs true $\implies$ Has odd cycle
Let (u, v) be the edge that causes the algorithm to output true. Let w be the lowest common ancestor of u and v in the DFS-tree. Then we have two path A and B both of which are from w to v, and one of them has odd length and the other has odd length (this is because u and v have the same color). Assume without loss of generality that A has odd length and B has even length. Since G is strongly connected, there is also a simple path C from v to w. If C has odd length, then B and C form an odd cycle; otherwise A and C also form an odd cycle. ■

3.22 First find out the strongly connected components (SCCs). Note that if there exist a vertex from which all other vertex are reachable, then this vertex must be in a source SCC; moreover, there can be only one source SCC in this case because vertices in different source SCCs are not reachable to each other. Therefore, after verifying that there is only one source SCC, we only need to pick one vertex (any of them) from the source SCC and check if it can reach all other vertices, which can be can be done using DFS.

3.24 Consider the following algorithm:

1 Perform topological sorting on the graph.

2 For every pair of consecutive vertices in the topological ordering, check whether there is an edge between the two vertices.

3.26 (a) **Has Eulerian tour $\implies$ All vertices have even degree**
An Eulerian tour is a cycle, so it must enter and exist each vertex the same number of times. Also, the tour use each edge once so it must enter and exist each vertex using different edges.
All vertices have even degree $\implies$ Has Eulerian tour
We prove by induction. The base case with only two vertices is trivial. Now suppose that every graphs with n vertices in which all vertices have even degree has an Eulerian tour. Let G be a graph with $n+1$ vertices and each of them has even degree. Let v be an arbitrary vertex in G, say v has degree $2i$ and its neighbours are $u_1, u_2, ..., u_{2i}$. We then obtain a graph G' from G as follows:

remove v and add the edges $(u_1, u_2), (u_3, u_4), ..., (u_{2i-1}, u_{2i})$. Note that G' has n vertices and each of them has even degree. By the induction hypothesis, G' has an Eulerian tour T. Now from T we obtain an Eulerian tour for G as follows: for each edge in T of the form (u_j, u_{j+1}), replace it with two edges $(u_j, v), (v, u_{j+1})$.

(b) A undirected graph has an Eulerian path if and only if exact two of its vertices have odd degree and all other vertices have even degree.

(c) A directed graph has an Eulerian tour if and only if each of its vertices have the same in-degree and out-degree.

3.29 First represent the equivalence relation R on the set S by an undirected graph. That is, the vertices are elements in S and two vertices x and y share an edge if and only if $(x, y) \in R$. The set of connected components of this graph yields an partition that satisfies the two conditions. The second condition is trivial. For the first condition, transitivity property of R ensures that any two vertices in the same connected component must share an edge.