473 Levitin problems and hints HW 06B

Problem 1: **(6)** 3.5.3 [5.2.3] Independence of properties from specific DFS traversals. Explain your answers.

3. Let G be a graph with n vertices and m edges.

   a. True or false: All its DFS forests (for traversals starting at different vertices) will have the same number of trees?

   b. True or false: All its DFS forests will have the same number of tree edges and the same number of back edges?

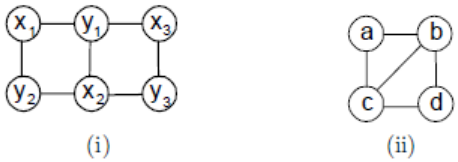
Author's Hints:

3. a. What is the number of such trees equal to?

   b. Answer this question for connected graphs first.

Problem 2: **(10)** 3.5.8a [5.2.8a] Bipartite graph checking using DFS

8. A graph is said to be ***bipartite*** if all its vertices can be partitioned into two disjoint subsets X and Y so that every edge connects a vertex in X with a vertex in Y. (We can also say that a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also called ***2-colorable***). For example, graph (i) is bipartite while graph (ii) is not.



(i) (ii)

a. Design a DFS-based algorithm for checking whether a graph is bipartite.

b. Design a BFS-based algorithm for checking whether a graph is bipartite.

Author's Hints:

8. Use a DFS forest and a BFS forest for parts (a) and (b), respectively.

Problem 3: **(5)** 4.1.1 [5.1.1] Ferrying Soldiers

1. *Ferrying soldiers* A detachment of n soldiers must cross a wide and deep river with no bridge in sight. They notice two 12-year-old boys playing in a rowboat by the shore. The boat is so tiny, however, that it can only hold two boys or one soldier. How can the soldiers get across the river and leave the boys in joint possession of the boat? How many times need the boat pass from shore to shore?

Author's Hints:

1. Solve the problem for $n = 1$.

Problem 4: **(5)** (not in 3rd edition) [5.1.9] binary insertion sort efficiency

Binary insertion sort uses binary search to find the appropriate position to insert A[i] among the previously sorted A[0] ≤ … ≤ A[I - 1]. Determine the worst-case efficiency class of this algorithm. I.e. get big-Θ time for number of comparisons and number of moves.

Author's Hints:

9. The order of growth of the worst-case number of key comparisons made by binary insertion sort can be obtained from formulas in Section 4.3 and Appendix A. For this algorithm, however, a key comparison is not the operation that determines the algorithm's efficiency class. Which operation does?

Problems 5: **(9)** 4.2.6 [5.3.6] dag source

In the 3rd edition, it says "Prove that a **nonempty** dag must have at least one source." That additional word is necessary!

6. a. Prove that a dag must have at least one source.

   b. How would you find a source (or determine that such a vertex does not exist) in a digraph represented by its adjacency matrix? What is the time efficiency of this operation?

   c. How would you find a source (or determine that such a vertex does not exist) in a digraph represented by its adjacency lists? What is the time efficiency of this operation?

Author's Hints:

6. a. Use a proof by contradiction.

   b. If you have difficulty answering the question, consider an example of a digraph with a vertex with no incoming edges and write down its adjacency matrix.

   c. The answer follows from the definitions of the source and adjacency lists.

Problem 6: **(9)** 4.2.9 [5.3.9] Strongly connected components of a digraph

9. A digraph is called ***strongly connected*** if for any pair of two distinct vertices u and v, there exists a directed path from u to v and a directed path from v to u. In general, a digraph's vertices can be partitioned into disjoint maximal subsets of vertices that are mutually accessible via directed paths of the digraph; these subsets are called ***strongly connected components***. There are two DFS-based algorithms for identifying strongly connected components. Here is the simpler (but somewhat less efficient) one of the two:

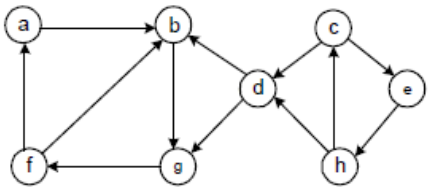
Step 1. Do a DFS traversal of the digraph given and number its vertices in the order that they become dead ends.

Step 2. Reverse the directions of all the edges of the digraph.

Step 3. Do a DFS traversal of the new digraph by starting (and, if necessary, restarting) the traversal at the highest numbered vertex among still unvisited vertices.

The strongly connected components are exactly the subsets of vertices in each DFS tree obtained during the last traversal.

a. Apply this algorithm to the following digraph to determine its strongly connected components.



b. What is the time efficiency class of this algorithm? Give separate answers for the adjacency matrix representation and adjacency list representation of an input graph.

c. How many strongly connected components does a dag have?

Author's Hints:

9. a. Trace the algorithm on the input given by following the steps of the algorithm as indicated.

b. Determine the efficiency for each of the three principal steps of the algorithm and then determine the overall efficiency. Of course, the answers depend on whether a digraph is represented by its adjacency matrix or by its adjacency lists.