ESO207: Data Structures and Algorithms	*Time : 90min.*
Quiz: 2	*Max. Marks: 100*

Instructions.

a. The exam is closed-book and closed-notes. No collaboration is permitted. You may not have your cell phone on your person.

b. You may use algorithms done in the class as subroutines and cite their properties, unless explicitly asked to prove them.

c. Describe your algorithms completely and precisely in English, preferably using pseudo-code.

d. Grading will be based not only on the correctness of the answer but also on the justification given and on the clarity of your arguments. Always argue correctness of your algorithms.

Problem 1. State your answer with justifications. $5 \times 3 + 10 = 25$

1. Let $T = (V, E)$ be a rooted undirected tree with root r with adjacency list representation, such that for every node, the entry for the children are in the order of left to right. Then, a typical DFS of T starting from r is most similar to which traversal of the tree: inorder, pre-order or post-order and why?

   Soln. DFS_VISIT(T, r) when called, will in turn call DFS_VISIT(T, v) for the children of v in order. If the node r is processed before all the other vertices, then the processing is the same as pre-order, if the node r is processed after all the other vertices, it is similar to post-order, and if the tree is binary and the node r is processed after the left child but before the right, then the processing is similar to inorder.

2. Give an example of a directed graph and its DFS such that there is a cycle in G involving a cross edge of the DFS. Label each vertex of your graph with its discovery and finish time interval.

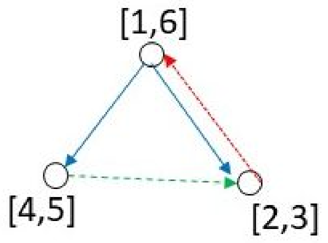

Figure 1: Blue edges are tree edges, green is cross-edge and red is back-edge.

3. Give an example of a weighted directed graph with a designated source vertex with possible negative weights for which Dijkstra's algorithm will not correctly give the shortest path lengths to all vertices. Dijkstra's algorithm starts from s and takes the current shortest path to c as having length 2. Having done that, it then finds the shortest path length to b as 3. What it doesn't realize is that there is a shorter path length $s - b - c$ which is of length $3 - 2 = 1$

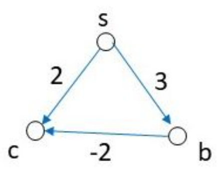

Figure 2: Dijkstra's algorithm fails on this graph. Source vertex is s

4. Design an algorithm (give outline) to solve the single source longest path problem in a DAG in linear time. (10)

   You can solve the single source shortest path problem in a DAG in linear time (as done in the class). So flip the signs of the edge-weights and run this algorithm. This gives the longest paths.

Problem 2. Design a linear-time algorithm that takes a strongly connected directed graph and determines if it is bipartite. A graph is bipartite if the vertex set V can be partitioned into two sets $V = V_1 \cup V_2$, such that V_1 and V_2 are disjoint, and all edges in E have their end points one in V_1 and another in V_2 (i.e., there are no edges with both ends in either V_1 or both ends in V_2).

Soln. Run BFS from any vertex (since G is strongly connected). BFS partitions vertices into levels, $L_0, L_1, \ldots, L_k$. After running BFS, make a pass over edges and check if for every edge (u, v), if $u \in L_i$ and $v \in L_j$, then, i and j have the opposite parity, that is, one of them is odd and the other is even. If all edges satisfy this, then the graph is bipartite, otherwise say that the graph is not bipartite. If the algorithm says bipartite, take the union of all the levels that are odd into one partition, and those that are even as the other partition. That gives the two partitions.

Argument. The algorithm is clearly correct if it says that the graph is bipartite. So suppose that the algorithm reports that the graph is not bipartite. We will show that there is an odd length cycle in G. This means that there is an edge (u, v) such that $u \in L_i, v \in L_j$ and u and v have the same parity. Suppose i, j are odd. Now run BFS or DFS from v and note the distance of s from v given by the search algorithm you have run. If it is odd, then, we have an odd length path from s to u from the first BFS, the edge (u, v) and the odd length path from v to s. This gives an odd length cycle from s to itself. Otherwise, the path length from v to s is even, then we get the following odd length cycle: s to v by original BFS, and v to s, which is odd + even = odd length. Note that the odd length cycle need not be simple (i.e., may have vertex repetitions).

Problem 3. Design an efficient (linear-time) algorithm that takes a directed graph $G = (V, E)$ and determines if there is a vertex $s \in V$ from which all other vertices are reachable. (25)

Soln.

1. Find a source vertex.
2. Do a DFS from that source vertex and at the end check if all vertices are reached (i.e., no white vertices are left).

The algorithm essentially checks if there are no 2 distinct source vertices in the graph. If so, they are not reachable from one another.

Problem 4. You are given a directed graph $G = (V, E)$, where the edges have non-negative weights and a symbol: $+$ or $-$. (The $+$'s and $-$'s are not the signs of the weights, the weights are always non-negative). You are given a source vertex s and a destination vertex t. The problem is to find an efficient algorithm that computes the shortest weight strictly *bitonic* path from s to t. A bitonic path is a sequence of edges which has the initial segment of all $+$s and the latter segment all $-$. A strictly bitonic path is one which has a non-empty initial segment consisting of $+$'s and a non-empty latter segment of $-$'s. (25)

Soln. Let $G^+ = (V, E^+)$ consists of the subgraph consisting of $+$ edges and G^- consists of the subgraph consisting of $-$ edges. Given an adjacency list representation, G^+ and G^- can be computed in linear time. Next we also compute G^{-R}, the reverse graph of G^-. Now do the following.

1. Run Dijkstra's single source shortest path algorithm from s on G^+. Suppose the min weight length from s in G^+ is stored in $u.dist^+$ and the parent vertex is stored in $u.\pi^+$. Assume $s.\pi^+ = \text{NIL}$.
2. Run Dijkstra's single source shortest path algorithm from t on G^{-R}. Assume that the corresponding fields are $u.dist^-$ and the parent vertex is $u.\pi^-$.
3. Combine to find the best strictly bitonic path.

```
1.  $mincost = \infty$
2.  for each vertex $u \in V - \{s, t\}$
3.      if $mincost > u.cost^+ + u.cost^-$
4.          $mincost = u.cost^+ + u.cost^-$
5.          $minvertex = u$
```

4. To print the minimum cost bitonic path, let *minvertex*$= u$. Follow the parent pointers starting from $u.\pi^+$ backward to s and reverse the path. This gives a directed path from s to u. Now follow the parent pointers starting from u starting from $u.\pi^-$ backwards to t, this gives a path in G^- from u to t. The two paths concatenated gives the desired minimum cost bitonic path from s to t. Time complexity is $O((|V| + |E|) \log |V|)$ for Dijkstra plus $O(|V|)$ for the above loop. The time for reconstructing the path is at most $O(|V|)$.