ESO207A: Data Structures and Algorithms
Homework 3 Solutions

Problem 1. Given a directed graph $G = (V, E)$ defined the graph $G^R = (V, E^R)$ with all edge directions reversed, that is, $E^R = \{(v, u) | (u, v) \in E\}$.

1. Suppose G is represented as an adjacency matrix. Give an $O(|V|^2)$ algorithm for finding the adjacency matrix representation of G^R. (4)

   Solution. Suppose G is a directed graph given by adjacency matrix A, that is, $A[u, v] = 1$ iff $(u, v) \in E$ and is 0 otherwise. Then, $A^R[v, u] = A[u, v]$. So the adjacency matrix of G^R is the transpose of A, denoted A^T. A^T can be computed in time $O(n^2)$. For an undirected graph, $G^R = G$ and hence, $A^T = A$; nothing new is to be computed.

2. Suppose G is represented as an adjacency list. Give an $O(|V| + |E|)$ algorithm for finding the adjacency matrix representation of G^R. (4)

   Solution. We initialize an array for V and set the adjacency list for G^R to NIL each– this takes $O(|V|)$ time. In G, for every $u \in V$, if $v \in Adj[u]$, then, to construct Adj^R, we insert u in the head of the adjacency list $Adj^R[v]$. This takes $O(|E|)$ time. The formal algorithm is as under.

   ```
   1.  for every u ∈ V
   2.      Adj^R[u] = NIL
   3.  for every u ∈ V
   4.      node = Adj[u]
   5.      while node ≠ NIL
   6.          v = node.vertex
   7.          insert u in the head of the list Adj^R[v]
   ```

   Lines 1-2 take $O(|V|)$ time. Every vertex incident from u is visited exactly once in line 5-7, this takes time $O(deg(u))$. Thus total time taken in lines 4-7 is $\sum_{u \in V} O(deg(u)) = O(|E|)$. Lines 3 to 7 take $O(|V| + |E|)$. This is the complexity.

3. A source vertex in G is a vertex s such that there is no vertex $v \in G$ such that $(v, s) \in E$. A sink vertex in G is a vertex u such that there is no vertex $v \in G$ such that $(u, v) \in G$. Relate the source and sink vertices of G and G^R. Relate the strong components of G with the strong components of G^R. (2+2)

   Solution. A source vertex s in G has no incoming edge. So when G^R is constructed, it will have no outgoing edge, and becomes a sink vertex. Conversely, a sink vertex in G becomes a source vertex in G^R. The strong components of G are strong components of G^R as well. That is because, if u and v are vertices in a strong component of G, then in G, there is a path p from u to v and a path q form v to u. So p^R now becomes a path from v to u in G^R and q^R becomes a path from u to v in G^R.

Problem 2. A connected graph $G = (V, E)$ is said to be bi-partite if V can be partitioned into two non-empty and disjoint partitions $V = V_1 \cup V_2$ such that any edge $\{u, v\}$ in E has $u \in V_1$

and $v \in V_2$. That is, there are no edges among vertices in V_1 or among vertices in V_2. Consider the following test for bi-partition. Run BFS on G starting from vertex s. Let $u.d$ be the shortest distance (as per BFS) from s to u.

1. Show that G is bipartite iff for every edge $\{u, v\} \in E$, $|u.d - v.d| = 1$. (12)
2. Extend the BFS algorithm slightly to give an $O(|V| + |E|)$ algorithm to test if G is bi-partite. (12)

Solution. Both parts will be solved by proving the following. Let $L_0, L_1, L_2, \ldots$ be the layers produced by BFS starting at some fixed node s, where, $L_0 = \{s\}$. Then exactly one of the following is true.

1. There is no edge of G joining two nodes of the same layer. In this case the graph is bi-partite, with a bipartition consists of giving red color to all even level vertices and blue color to all odd level vertices.
2. There is an edge joining two vertices of the same layer. In this case G contains an odd length cycle and is not bi-partite.

Case 1 is the bi-partite graph condition. For undirected graphs, all edges proceed between layers (no edges between vertices of the same layer) and hence, for every edge, $\{u, v\}$, $|u.d - v.d| = 1$.

Run a BFS that generates layers of vertices. Exactly one of two possibilities happens. Either there is a pair of vertices in the same level or no pairs of vertices are at the same level. In the latter case, the graph is bi-partitite, by putting vertices in even numbered layers into one red partition and vertices in odd numbered layers into the other blue partition.

In the other case, there are two vertices say x and y that are in the same layer L_i and have an edge. Consider the BFS tree T produced by the BFS. Now let z be the least common ancestor of x and y and is at layer L_j. Then, the cycle produced by the tree edges from z to x, followed by the edge $\{x, y\}$, followed by the tree edges from y back to z is of size $2(i - j) + 1$ and is odd. It is clear that an odd cycle cannot be bi-partite.

Part 2 asks to extend BFS with a test to check if for any edge $\{u, v\}$, whether u and v are in the same layer.

```
BFS(G, s)
1.   for  each u ∈ V
2.        u.d = ∞
3.        u.visited = 0
1.   Enqueue(Q, s)
2.   s.d = 0; s.parent = NIL
3.   s.visited = 1
4.   while  Q is not empty {
5.        u = Dequeue(Q)
6.        for every vertex v ∈ Adj[u]
7.             if v.visited
8.                  if  u.d == v.d
```

9. **return odd cycle: not bipartite**
10. **else** {
11. $v.d = u.d + 1$;
12. $Enqueue(Q, v)$
13. $v.parent = u$
14. $v.\ visited = 1$ }
15. return **bi-partite**

Problem 3. Give an $O(|V|+|E|)$ algorithm that takes a directed graph and a given edge $e = (u, v)$ and tests if there is a cycle involving e in the graph. (20)

Solution. Outline: Start a DFS with u and consider the edge (u, v) as the tree edge to place v just above u on stack. Now continue to run DFS from this state. If there is a back edge to u then there is a cycle involving (u, v). If no back edge to u is found before DFSVisit(v) terminates, then, there is no cycle involving (u, v).

Problem 4. Given a graph $G = (V, E)$, consider the following alternative outline for finding a topological order that repeatedly removes source nodes from G.

Repeat until the graph is empty
Find a source, output it and delete it.

Design an $O(|V| + |E|)$ time algorithm to implement this (alternative) outline and prove your complexity. (22)

Solution. Reverse the graph to get G^R. Run a DFS on G^R. Output vertices in the order their color turns black during the DFS.

A source vertex in a graph G is a sink vertex in G^R. Do a DFS on G^R. Consider the sequence of vertices as they turn black. The first black vertex is a sink vertex. Making a vertex v black essentially removes it from the graph, because, for every edge $(u, v) \in E$, the edge will not be processed any further. This gives the correctness of the algorithm.

Problem 5. Given a directed graph $G = (V, E)$, design an algorithm to find a vertex u that has paths to all vertices in V in G. (22)

Solution.

First let us assume that the graph is a DAG. Find all source vertices in G. If there is only one source vertex, then this has paths to all vertices in V. If there are multiple source vertices then this is not true.

We will prove the correctness below. Source vertices are detected by reversing G and checking for sink vertices in G^R. A vertex with nil adjacency list is a sink. All sinks are found by going over all the adjacency lists and testing for emptiness. Reversal takes time $O(|V| + |E|)$ and testing if a vertex is a sink takes time $O(1)$, total time is $O(|V| + |E| + |V| \cdot 1) = O(|V| + |E|)$.

If there are multiple source vertices u and v, then there cannot be a vertex w with paths to u or v, since, each u, v are sources. Also u can't reach v and vice-versa.

Now suppose there is a single source vertex s and say that it can't reach v. Then, consider a predecessor vertex v_1 of v that is, $(v_1, v) \in E$. Consider the predecessor v_2 of v_1, and so on, till

you get a vertex say v_k which has no predecessor, that is, it is a source. Now, v_k is not s since s is assumed to not have a path to v. So we have at least two sources, a contradiction.

Now assume that G is a general directed graph. Let $G^{scc} = (V^{scc}, E^{scc})$ be the strong components abstracted graph obtained by running the SCC algorithm on G. Run the above algorithm on G^{scc}. If there is exactly one source vertex W in V^{scc}, then, each vertex $v \in W$ reaches all vertices in G. If there are multiple source vertices, then, this is not possible.