Q1. (10 points) An undirected 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. Design a linear time, i.e., O(|V| + |E|), time algorithm to check if a graph is bipartite or not.

Answer: Just perform a DFS from any vertex v. Label vertex v as 'X', and in the DFS, label all its neighbors as 'Y'. Likewise, if a vertex has label 'Y', its neighbors should be labeled 'X'. If at any point we ever see a neighbor with the same label, then the graph cannot be bipartite. Here is the algo:

```
Bipartite(G = (V,E)):
Set label(v) = 0 for all v in V
For v in V with label(v) = 0:
        res = BipartiteDFS(v, 'X')
        if res = False: Return res
Return res

BipartiteDFS(v, L):
label(v) = L
if L = 'X': newL = 'Y'
else: newL = 'X'
For all x in neighbors(v):
        if label(v) = 0: return BipartiteDFS(v, newL)
        elseif label(v) = L: return False
return True
```

Since it is a DFS, the time is O(|V|+|E|)

Q2. (15 points) Answer the following questions:
a. Prove that a non-empty DAG must have at least one source.
b. What is the time complexity of finding a source in a directed graph or to determine such a source does not exist if the graph is represented by its adjacency matrix? Describe the algorithm.
c. What is the time complexity of finding a source in a directed graph or to determine such a source does not exist if the graph is represented by its adjacency list? Describe the algorithm.

Answer:

a) Consider a vertex v in V. Let S(v) be the set of all vertices that can reach v in the DAG. Now if S(v) is empty, then v is a source vertex. On the other hand, if S(v) is not empty, then it can be at most of size n-1, since v does not belong to S(v). Now, take any vertex x in S(v), and compute S(x) – the set of all vertices in S(v) that can reach x. Again either S(x) is empty in which case x is a source, or it is not empty. But this time S(x) has size at most n-2, since we have excluded both v and x. If we continue in this manner, by induction, we will be left with some vertex y for which S(y) is empty, since the size of the 'S' sets decreases by at least 1 at each step. This last vertex y must be the source.

b) Given the adjacency matrix of a directed graph, we have to traverse all $O(|V|^2)$ entries to add the number of incoming edges for each vertex v, which gives the in-degree of that node. So, if entry (v,x) is 1 in the matrix, we increment the in-degree of x. If there exists a node with in-degree 0, we have found a source. And if all in-degrees are more than 0, then there is no source. Total time $O(|V|^2)$.

c) For an adjacency list, we can scan all entries in the adjacency list and update the in-degree of each vertex. For each vertex v, is x belongs to the adjacency list of v, we increment the in-degree of x. This takes time O(|V|+|E|).

Q3. (10 points) Describe a linear time algorithm to compute the neighbor degree for each vertex in an undirected graph. The neighbor degree of a node x is defined as the sum of the degree of all of its neighbors.

Answer: We make two passes. In the first pass, we compute the degree of each vertex v, by looking at the length of its adjacency list. This takes time O(|V|) if the length of the adjacency list is already available, or in the worst case O(|V|+|E|) if we have to traverse the list to compute its length.

In the second pass, we initialize ND(v) = 0, and then simply add the degrees of all of v's neighbors into ND(v).

```
For all v in V:
        ND(v)  = 0
        For all x in neighbors(v):
                ND(v) = ND(v) + degree(x)
```

Total time is O(|V|+|E|). We can also use DFS search.

Q4. (20 points) Consider a directed graph that has a weight w(v) on each vertex v. Define the reachability weight of vertex v as follows:

$$r(v) = \max\{w(u) \mid u \text{ is reachable from } v\}$$

That is, the reachability weight of v is the largest weight that can be reached from v. Answer the following questions:

a. Assume the graph is a DAG. Describe a linear time algorithm to compute the reachability weight for all vertices.

Answer:

First, perform a topological sort of the DAG. Let v1, v2, v3, …, vn be the list of vertices in sorted order. Now we just process each vertex in reverse order and update its reachability weight as follows:

```
For i = n to 1:
        r(vi) = w(vi)
        for all neighbors x of vi:
                if r(vi) < r(x): r(vi) = r(x)
```

Topological sort takes O(|V|+|E|) time and so does the code snippet above.

b. Assume that the graph is a general directed graph (with possible cycles). Describe a linear time algorithm to find the reachability weight for all vertices.

Answer:

The solution is to first compute the set of strongly connected components of the directed graph. Next, set the weight of each SCC as the maximum w(v) value for vertices in that SCC. Next run algorithm a) on the SCC. Total time is O(|V|+|E|).