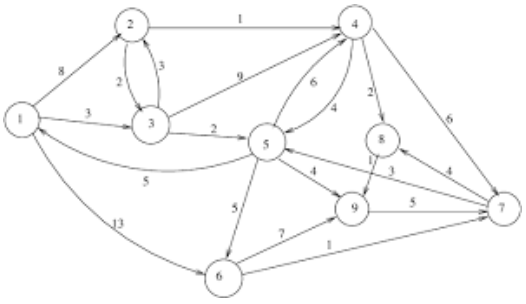
Exercises related to the chapter on Graph Algorithms



1. Consider Figure 1 above.

   (a) Give the adjacency list representation of this directed graph
   Solutions: We can represent this as a list A[current node] of pairs with the form (target node, weight)
   A[1] = {(2, 8), (3, 3), (6, 13)}
   A[2] = {(3, 2), (4, 1)}
   A[3] = {(2, 3), (4, 9), (5, 2)}
   A[4] = {(5, 4), (7, 6), (8, 2)}
   A[5] = {(1, 5), (4, 6), (6, 5), (9, 4)}
   A[6] = {(7, 1), (9, 7)}
   A[7] = {(5, 3), (8, 4)}
   A[8] = {(9, 1)}
   A[9] = {(5, 5)}
   In Python, we can have a quick implementation as shown in the figures

```python
# "Dirty" implementation for building an adjacency list for a directed weighted graph
class GraphList:
    # Creating new node
    def CreateNode(target_node, weight):
        return (target_node, weight)

    # Insert node into an arry
    def InsertNode(arr, cur_node, tar_node):
        arr[cur_node].append(tar_node)
        return arr

    # Print the Adjacency List
    def PrintAdjacencyList(arr, length):
        for row in range(1, length):
            num_cols = len(arr[row])
            for col in range(num_cols):
                if arr[row][col]!=None:
                    print(row, "->", arr[row][col])
        return

# Driver code
# Define an empty matrix to store the node.
arr = [[] for row in range(10)]

# Create some nodes and add to the matrix

node_1_2 = GraphList.CreateNode(2, 8)
GraphList.InsertNode(arr=arr, cur_node=1, tar_node=node_1_2)
node_1_3 = GraphList.CreateNode(3, 3)
GraphList.InsertNode(arr=arr, cur_node=1, tar_node=node_1_3)
node_1_6 = GraphList.CreateNode(6, 13)
GraphList.InsertNode(arr=arr, cur_node=1, tar_node=node_1_6)

GraphList.PrintAdjacencyList(arr=arr, length=10)
```

```
1 -> (2, 8)
1 -> (3, 3)
1 -> (6, 13)

Process finished with exit code 0
```

(b) Which node(s) have the largest in-degree value.
Solutions: nodes 4, 5, 7, 9 have in-degree 3

(c) Which node(s) have the largest out-degree value.
Solutions: nodes 1, 3, 4, 5 have out-degree 3

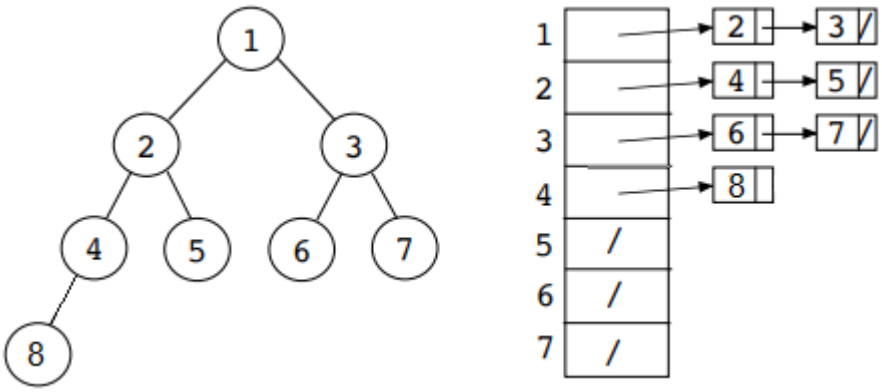
2. Is the graph in Figure 1 a multi-graph? Explain briefly your answer.
Solutions: No, there is at most one directed edge between each pair of nodes.

3. How long does it take to compute the out-degree of every node?
Solutions: Using adjacency list graph representation (an array of size n of pointers to link lists), an algorithm will visit the adjacency list of

each entry of the array and for each of them count the number of nodes in the adjacency list. The worst case is when the graph is complete, cost is $O(n^2)$, or $O(|E|)$ as we have to visit n link lists the length of each adjacency list is $O(n)$.

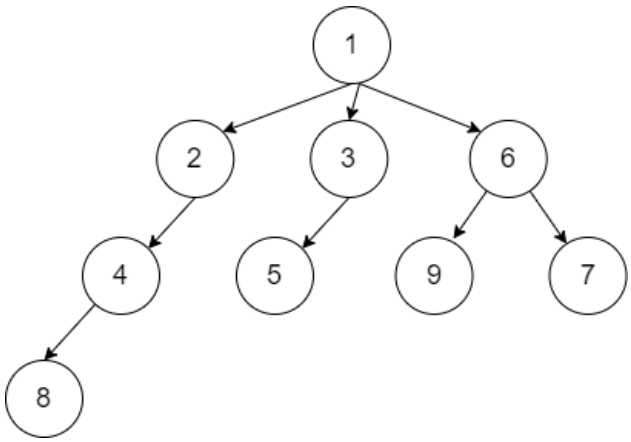
4. How long does it take to compute the in-degrees?
   Solutions: This can be done in n^2 or $|E|$ also. For each node we create an in-degree counter. Each time a node is found in an adjacency list, its counter is incremented by one.

5. Give an adjacency-list representation for a complete binary tree on 7 nodes. Give an equivalent adjacency-matrix representation. Assume that nodes are numbered from 1 to 7 as in a binary heap



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	0	0	0	1	1	0	0	0
3	0	0	0	0	0	1	1	0
4	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

6. Run the breadth-first search algorithm on slide 13 on the directed graph of Figure 1, using node 1 as the source. Draw the resulting tree, as indicated in the BFS algo give for each node i its predecessor π (parent of i in the tree) and its level d in the tree

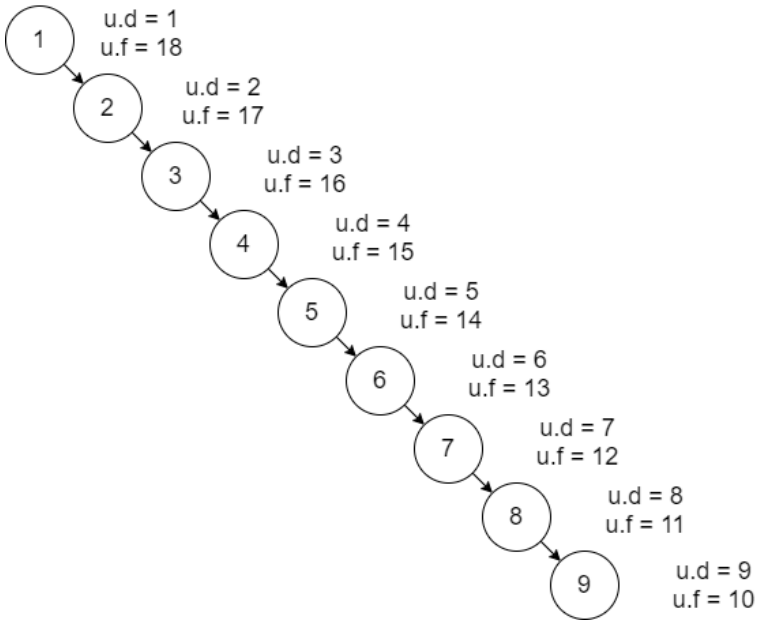


7. The BFS algorithm on slide 13 assumes that the graph of Figure 1 is represented using adjacency lists.

   (a) If in one instance (a) the nodes are always stored in increasing order in the adjacency lists while in another instances (b) the nodes are always stored in decreasing order, will the trees be the same in each instance?
   Solutions: Yes, different. BFS is implemented using a queue. Each time the ENQUEUE() operation is executed for a given node v, the adjacency list will be copied in the queue in the same order as it appears in the adjacency list. In instance (a) the left most child of node v in the generated tree will be the node with the smallest value in the adjacency list while on instance (b) the left most child of node v in the generated will be the one with the largest value in the adjacency list.

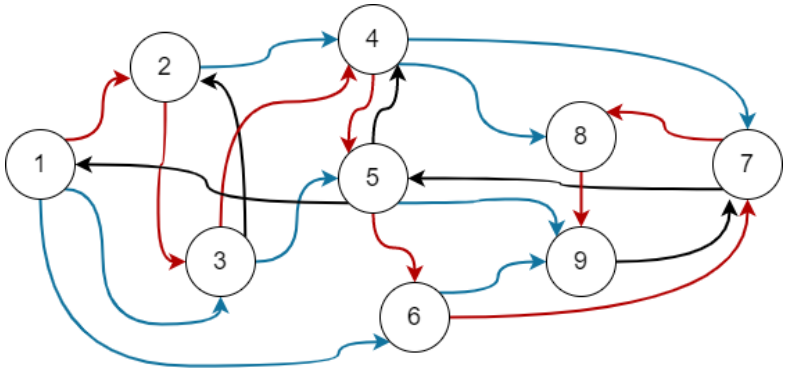
   (b) Will the depth of each node change if nodes are stored in increasing versus decreasing orders in the adjacency lists?
   Solutions: No, the depth does not change

8. Using the DFS algorithm on slide 30, show how depth-first search works on the graph of Figure 1. Assume that the second **for** loop of the algorithm considers the nodes in increasing order order, and assume that nodes in each adjacency list are stored in increasing order. Write the u.d and u.f time of each node in the tree.



9. Using the graph of Figure 1 and the tree you obtained in the previous question, identify in Figure 1 the edges that are tree edges, back edges and forward edges

   Solution: Red - Tree edges, Black - Backward edges, Blue - forward edges



10. Does the graph in Figure 1 is an acyclic graph? Explain briefly your answer.
    Solutions: It is not. In the previous question you will have identified back edges in the tree obtained using the DFS algorithm. We can also determine directly whether this graph is acyclic by searching for circular paths (cycles) in the graph. There is a cycle 2-3-2; 4-5-4; 1-2-5-1; 7-8-9-7 and several others.

11. Rewrite the DFS algorithm on slide 30, using a stack to eliminate recursion

```
DFS-stack(G,s){
  for (v ∈ G.V)
    visited[v] = false;
  S = EmptyStack;
  Push(S,s);
  while (S not empty){
    u = Pop(S);
    if (visited[u] == false) {
      visited[u] == true;
      for (w ∈ u.adj) {
        if (visited[w] == false)
          Push(S,w);
      }
    }
  }
}
```

12. Show the ordering of nodes produced by the topological-sort algorithm (slide 57) when it is run on the dag of Figure 2 below
Solution: Starting from 140, we have: 142, 140, 143, 374, 373, 414, 417, 413, 415, 410, 351, 352, 333, 341, 331, 403, 311, 332, 344, 312, 154. Note that the answer may vary depending on the way choosing node to visit (in increasing order, from right to left, etc.)