COMP 2611 – Data Structures

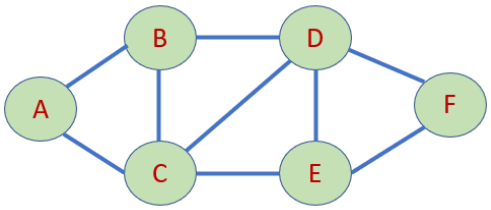
2023/2024 Semester 1

Lab #9

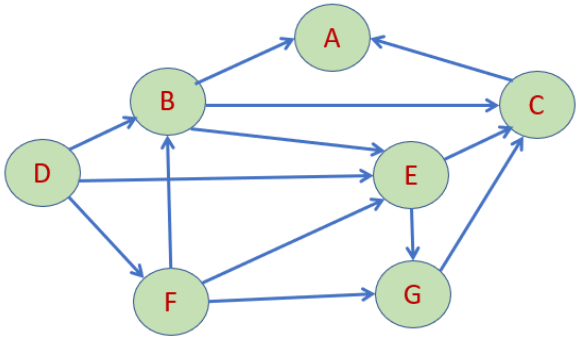
Part 1: Graphs

(1) Draw the adjacency list and adjacency matrix representations of the following graphs:

a) (Undirected)



b) (Directed)



(2) A graph is stored in a text file as follows:

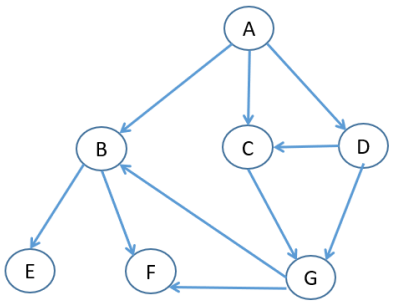
```
7
A E G J N R T
A 2 N T
E 1 R
G 2 E R
J 0
N 3 E G T
R 1 J
T 3 G J R
```

The first line of the file indicates that the graph has 7 vertices. The second line is the name of each vertex. Each subsequent line is for one of the 7 vertices (in alphabetical order). It lists the vertex and then indicates how many edges leave that vertex. For example, there are two edges out of *A* and one out of *E*. The remaining data on the line are the vertices to which the given vertex is connected. For example, *A* is connected to *N* and *T*.

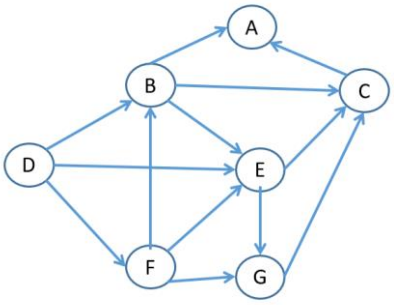
Draw the graph.

(3) Give the depth-first and breadth-first traversal of the following graphs, assuming that vertices are visited based on the alphabetic ordering of the names of the vertices.

(a) Starting from *A*:



(b) Starting from *D*:



(4) A graph is stored as follows in a file:

```
6
A B C D E F
A 2 B C
B 2 D F
C 2 D F
D 1 E
E 2 A B
F 1 E
```

Give the depth-first and breadth-first traversal of the graph, starting from *A*, assuming that vertices are visited based on the alphabetic ordering of the names of the vertices.

Part 2: Some Exam-Type Questions

1. a) Write a **recursive** function to copy the elements of one linked list to another linked list. The function should return the top of the new linked list. Its prototype is:

```
LLNode * copyLL (LLNode * top);
```

   b) Write a recursive function, *isSortedRec*, which returns *true* if the elements of the linked list passed as a parameter are sorted in ascending order, and *false* otherwise. Its prototype is:

```
bool isSortedRec (LLNode * top);
```

2. (a) Draw the two binary search trees (BSTs) produced when the following keys are inserted:

   (i) 40 20 10 50 25 60 15 5 75 55 45

   (ii) 10 20 30 40 50 60 70 80 90

   (b) Delete the node with 50 from the BST in (i).

   (c) Write a function, *isBST*, with the following prototype, which returns *true* if the binary tree rooted at *bt* is a BST and *false*, otherwise.

```
bool isBST (BTNode * bt);
```

   HINT: An inorder traversal of a BST produces the keys in increasing order.

   (d) Write a **recursive** function with the following prototype to find the rank of a key in a BST. Assume that the key exists. The *rank* of a key in a BST is the number of nodes in the BST which have keys that are strictly less than the given key. You are not allowed to use the *inorderSuccessor* function, but you can use the usual binary tree and BST functions.

```
int keyRankBST (BTNode * root, int key);
```

3. An integer array, *A*, contains the following values:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
50	40	72	74	75	70	95	85	60	63	81

   (a) Draw the max-heap after the following statements are executed:

```
MaxHeap * maxHeap = initMaxHeapFromArray (A, 11);
buildMaxHeap (maxHeap);
```

   (b) Draw the max-heap after Node 2 is deleted.

   (c) Draw the max-heap after 92 is inserted.

   (d) Draw the max-heap after the biggest element is removed.

End of Lab #9