National Institute of Technology, Calicut
Department of Computer Science and Engineering
CS2094D – Data Structures Lab (MCA)

Assignment-4

Policies for Submission and Evaluation:

You must submit your assignment in the moodle (Eduserver) course page, on or before the submission deadline. Also, ensure that your programs in the assignment must compile and execute without errors in Andromeda server. During evaluation your uploaded programs will be checked in Andromeda server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program. Detection of ANY malpractice regarding the lab course will also lead to awarding an F grade.

Naming Conventions for Submission:

Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz).

The name of this file must be ASSG<NUMBER>_<ROLLNO>_<FIRSTNAME>.zip (For example: ASSG3_MxxyyyyCA_LAXMAN.zip). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

The source codes must be named as ASSG <NUMBER> _<ROLLNO> _<FIRSTNAME> _ <PROGRAM-NUMBER> .<extension> (For example: ASSG3_MxxyyyyCA_LAXMAN_1.c). If there is a part 'a ' and a part 'b' for a particular question, then name the source files for each part separately as in ASSG3_MxxyyyyCA_LAXMAN_1b.c. Make sure that you follow the naming conventions.

Standard of Conduct:

Violations of academic integrity will be severely penalized.

Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course.

The department policy on academic integrity can be found at:

http://cse.nitc.ac.in/sites/default/files/Academic-Integrity.pdf

Assignment Questions

NOTE: The input should be read from the file **input.txt** and output should be written to the file **output.txt.**

Q1. Depth first search (DFS) is an algorithm for traversing or searching graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Write a C program to implement the Depth first search (DFS) in a directed graph. Vertex ordering in a graph follows the natural number sequence starting from 0.

Input
The first line contains two integers denoting the number of vertices, ***n,*** and the number of edges, ***m***
The next ***m*** lines denote the pair of vertices representing edge
The last line contains the source vertex

Output
The DFS traversal of the graph

Sample Input:

```
4 6
0 1
0 2
1 2
2 0
2 3
3 3
0
```

Output:

```
0 1 2 3
```

Q2. Write a program in C that receives as input a Boolean 2D matrix, and returns the number of *islands* in the matrix. A group of connected 1s forms an island.

Input
The first line will read ***m***, the number of rows, in the matrix
The second line will read ***n***, the number of columns in the matrix
The following ***m*** lines will read each row of the matrix (***n*** elements each, separated by a single space)

Output
The number of islands in the matrix

Sample Input 1:

```
5
5
1 1 0 0 0
0 1 0 0 1
```

```
1 0 0 1 1
0 0 0 0 0
1 0 1 0 1
```

Output:

```
5
```

Sample Input 2:

```
3
7
1 0 0 1 1 1 1
0 0 1 1 0 0 0
1 0 0 0 0 0 0
```

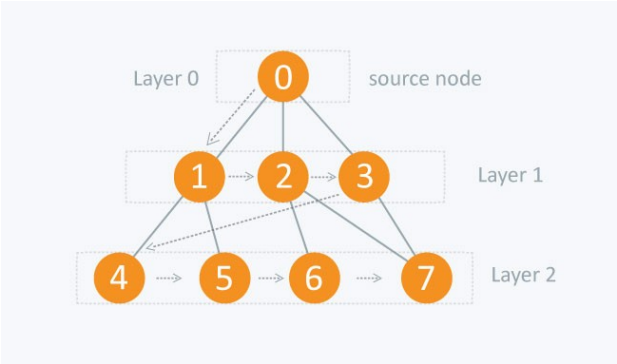
Output:

```
3
```

Q3. BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer-wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.
As the name BFS suggests, you are required to traverse the graph breadth-wise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

Consider the following diagram.



The distance between the nodes in layer 1 is comparatively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

Write a C program to implement the Breadth First Search (BFS) in a directed graph. Vertex ordering in a graph follows the natural number sequence starting from 0.

Input
The first line contains two integers denoting the number of vertices, ***n,*** and the number of edges, ***m***
The next ***m*** lines denote the pair of vertices representing edge
The last line contains the source vertex

Output
The BFS traversal of the graph

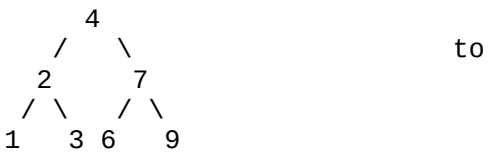
Sample Input:

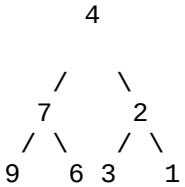
```
4 6
0 1
0 2
1 2
2 0
2 3
3 3
0
```

Output:

```
0 1 2 3
```

Q4. Write a program to invert a given binary tree using Breadth First Search.





Input
The first line will read ***n***, the number of nodes in the binary tree
The next line will read all of the ***n*** nodes, each separated by a single space

Output
The inverted binary tree in one line

Sample Input 1:

```
7
1 2 3 4 5 6 7
```

Output:

```
1 3 2 7 6 5 4
```

Sample Input 2:

```
3
3 4 5
```

Output:

Q5. Write a C program to implement hash table data structure using open addressing to store student information with roll number as key. Your program should contain the following functions:

- hashTable(int m) - create a hash table of size **m**
- insert(int k) - insert element into hash table having key value as **k**
- search(int k) - find whether element with key **'k'** is present in hash table or not
- delete(int k) - delete the element with key **'k'**

Input:
The first line contains a character from {'a', 'b', 'c'} denoting the options as below:

a - Collision resolution by *linear probing* with hash function
$h(k, i) = (h_1(k) + i) \bmod m$ where $h_1(k) = k \bmod m$
b - Collision resolution by *quadratic probing* with hash function
$h(k, i) = (h_1(k) + c_1 i + c_2 i^2) \bmod m$
where $h_1(k) = k \bmod m$, c_1 and c_2 are positive auxiliary constants, $i = 0,1,...m-1$
c - Collision resolution by *double hashing* with hash functions
$h(k, i) = (h_1(k) + i * h_2(k)) \bmod m$
where, $h_1(k) = k \bmod m$, $h_2(k) = R - (k \bmod R)$ {R = Prime number just smaller than the size of table}

The next line contains an integer ***m***, denoting the size of hash table
(Note: In case of quadratic probing only(option b), the next line contains the constants c_1 , c_2 separated by a single space)
The next lines contains a character from { 'i', 's', 'd', 'p', 't' } followed by zero or one integer:
i x - insert the element with key x into hash table
s x - search the element with key x in hash table. Print 1 if present otherwise print -1
d x - delete the element with key x from hash table.
p - print the hash table in "index (key values)" pattern.(See sample output for explanation)
t - terminate the program

(Note :

In case of linear probing, quadratic probing and double hashing, total elements (n) to be inserted into hash table will be lesser than or equal to the size of the hash table (m) i.e., n<=m
a. deletion operation will always be a valid operation
b. While printing the hash, multiple key values must be separated by a single white space.)

Output:
The output (if any) of each command should be printed on a separate line

Sample Input 1:

```
a
7
i 76
i 93
i 40
i 47
i 10
i 55
p
s 35
```

```
s 47
d 47
s 55
t
```

Output:

```
0 (47)
1 (55)
2 (93)
3 (10)
4 ()
5 (40)
6 (76)
-1
1
1
```

Sample Input 2:

```
b
7
0 1
i 76
i 40
i 47
i 5
s 5
i 55
p
s 62
d 55
t
```

Output:

```
1
0 (47)
1 ()
2 (5)
3 (55)
4 ()
5 (40)
6 (76)
-1
```

Sample Input 3:

```
c
7
i 76
i 93
i 40
i 47
i 10
i 55
p
d 40
```

s 47
s 76
s 40
t

Output:
0 ()
1 (47)
2 (93)
3 (10)4 (55)
5 (40)
6 (76)
1
1
-1

Q6. Write a program to get the union and intersection of two linked lists using hashing.

Note: Order of elements need not be maintained while printing the output.

Input
The first line will read all elements of the first linked list
The next line will read all elements of the second linked list

Output
The first line will have the union list of elements
The second line will have the intersection list

Sample Input 1:

10 15 4 20
8 4 2 10

Output:

4 10
2 8 20 4 15 10

Sample Input 2:

81 24 35 17 10 9 1 3
1 2 3 4 5 6 7 8 9 10

Output:

10 9 1 3
1 2 3 5 6 7 8 20 9 4 15 10 35 17 81 24