EC 4070 DATA STRUCTURES AND ALGORITHMS

LAB 07

NAME : AARTHY.V

REGISTRATION NO. : 2019/E/001

DATE ASSIGNED : 30 MAR 2020

Q1.

```
import java.util.*;
import java.util.Scanner;

public class JungleRun_2019_E_001_L7
{
	public static int startR;
	public static  int startC;
	public static int shortDis=30*30;

	// creating node class
	static class Node
	{
		int row;
		int column;
		int distance;
		String key;

		// constructor
		Node(String key, int row, int column)
		{
	    this.key = key;
	    this.row = row;
	    this.column = column;
			distance = 1000;
	  }

		public void setDistance(int distance)
		{
			this.distance = distance;
		}
		// getting distance function
		public int getDistance()
		{
			return distance;
		}
		// getting key function
		public String getKey()
		{
			return key;
		}
		// getting row function
		public int getRow()
		{
			return row;
		}
		// getting column function
```

```
                public int getColumn()
                {
                        return column;
                }

        }

        // function for finding the starting position of matrix
        public static void findStart(LinkedList<LinkedList<Node>> graphADT, int size)
        {
    for (int i = 0; i < size; i++)
          {
       for (int j = 0; j < size; j++)
                  {
         if (graphADT.get(i).get(j).getKey().equals("S")) // if S is found in the matrix
                          {
            startR = i;
            startC = j;
            graphADT.get(startR).get(startC).setDistance(0); // setting the distance in that place as 0
            break;
          }
       }
    }
  }

        // function for finding shortest path
   public static void findDistance(LinkedList<LinkedList<Node>> graphADT, int length, int row, int
column)
        {

    if ((row < graphADT.size()) && (row > -1) && (column < graphADT.size()) && (column > -1))
                {

        switch (graphADT.get(row).get(column).getKey())
                        {
          case "E": // if found E, shortest distance was found
            if (shortDis > length + 1)
                                {
              shortDis = length + 1;
              graphADT.get(row).get(column).setDistance(shortDis);
            }
            break;
                                case "S":
            findDistance(graphADT, length, row + 1, column);
            findDistance(graphADT, length, row - 1, column);
            findDistance(graphADT, length, row, column + 1);
            findDistance(graphADT, length, row, column - 1);
            break;
          case "P":
```

```
            if (graphADT.get(row).get(column).getDistance() == 1000)
                                        {
                length = length + 1;
                graphADT.get(row).get(column).setDistance(length);
                findDistance(graphADT, length, row + 1, column);
                findDistance(graphADT, length, row - 1, column);
                findDistance(graphADT, length, row, column + 1);
                findDistance(graphADT, length, row, column - 1);
            }
            break;
          case "T": // if found T escape
            break;
        }
      }
  }


        // print rules()
        static void printRules()
        {
                System.out.println();
                System.out.println("Enter the matrix using these key words");
                System.out.println("START \t S");
                System.out.println("END \t E");
                System.out.println("PATH \t P");
                System.out.println("TREE \t T");
        }

        // main function
        public static void main (String args[])
        {
                System.out.println("\n\tJUNGLE RUN PROBLEM");
                System.out.println("\t--------------------");

                // initializing graphADT as linked list
                LinkedList<LinkedList<Node>> graphADT = new LinkedList<>();

                // take size of the matrix as input from user (square matrix)
                Scanner sc=new Scanner(System.in);
                System.out.print("Enter the rows and columns of the matrix: ");
                int m_size=sc.nextInt();
                System.out.println("Matrix size is: " + m_size + "*" + m_size);

                printRules(); // printing rules for entering matrix
                System.out.println("\nEnter the Matrix (rows and columns) according to the size with tab:
");
                // take matrix as input from the user
                for (int i=0; i<m_size; i++)
                {
```

```
                graphADT.add(new LinkedList<>()); // adding new linked list
                for (int j=0; j<m_size; j++)
                {
                        graphADT.get(i).add(new Node(sc.next(),i,j)); // adding new node to the
linked list created
                }
        }

        // finding size of linkedlist
        int size=graphADT.size();
        // findind the place which we should start
        findStart(graphADT, size);
        findDistance(graphADT, 0, startR, startC);


        if (shortDis != 30*30)
        {
   System.out.println("Shortest way to exit from jungle: " + shortDis );
  }
  else
        {
   System.out.println("Sorry, No way to escape from the jungle");
        }


   }
}
```

```
F:\EC 4070\2019E001_Lab07\2019_E_001_L7>javac JungleRun_2019_E_001_L7.java

F:\EC 4070\2019E001_Lab07\2019_E_001_L7>java JungleRun_2019_E_001_L7

        JUNGLE RUN PROBLEM
        --------------------
Enter the rows and columns of the matrix: 5
Matrix size is: 5*5

Enter the matrix using these key words
START    S
END      E
PATH     P
TREE     T

Enter the Matrix (rows and columns) according to the size with tab:
S       P       P       P       P
T       P       T       P       P
T       P       P       P       P
P       T       E       T       T
P       T       P       T       T
Shortest way to exit from jungle: 5
```