Competitive Programming Lab - 3

Academic year: 2020-2021 **Semester:** Long Sem

Faculty Name: Dr. Ajith Jublison sir **Date:** 18 / 6 / 2022

Student name: Taran Mamidala **Reg. no.:** 19BCE7346

Q1.) Word Problem

Get a matrix from the user. Pick a minimum of one character from one row and create the word. If the guessed word abides to the rule return 1 else 0.

CODE:

```
import java.util.*;

public class WordProblem{
    public static int exist(ArrayList<String> a, String b) {
        if(a == null || a.size() == 0 || b == null || b.length() == 0)
            return 0;
        int result = 0;
        char[][] marked = new char[a.size()][a.get(0).length()];
        for(int i = 0; i < a.size(); i++){
            for(int j = 0; j < a.get(i).length(); j++){
                marked[i][j] = a.get(i).charAt(j);
            }
        }
        for(int i = 0; i < a.size(); i++){
            for(int j = 0; j < a.get(i).length(); j++){

                if(wordSearching(a, b, i, j, 0, marked));
                return 1;

            }
        }
        return 0;
    }

    public static boolean wordSearching(ArrayList<String> a, String b,
int row, int column, int index, char[][] marked){

        if(row < 0 ||
           row >= a.size() ||
```

```
            column < 0||
            column >= a.get(0).length())

             return false;

        if(marked[row][column] == b.charAt(index)){
            char temp = marked[row][column];
            marked[row][column] = '!';

            if(index == b.length() -1)
                return true;
            else if(wordSearching(a, b, row, column + 1, index + 1,
marked) ||
                    wordSearching(a, b, row, column - 1, index + 1,
marked) ||
                    wordSearching(a, b, row -1, column, index + 1,
marked)  ||
                    wordSearching(a, b, row + 1, column, index + 1,
marked))
                return true;
            marked[row][column] = temp;
        }
        return false;

    }


    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        ArrayList<String> a = new ArrayList<String>();
        a.add("ABCDE");
        a.add("ESFC");
        a.add("XDEE");

        for(String s : a){
            for(int i = 0; i < s.length(); i++)
                System.out.print(s.charAt(i) + " ");
            System.out.println();
        }

        for(int i=0;i<5;i++){

        System.out.println("Enter the word : ");
        String b = sc.nextLine();
        String t1 = "STOP";
```

```
            if(b.equals(t1)){
                System.out.println("--Ended--");
                break;
            }
            System.out.println("b : " + b);
            System.out.print("Output : ");
            System.out.println(exist(a, b));
            }
            sc.close();
        }
```

Output:

Result
compiled and executed in 28.382 sec(s)

```
A B C E
E S F C
X D E E
Enter the word :
AXE
b : AXE
Output : 1
Enter the word :
BED
b : BED
Output : 1
Enter the word :
DOG
b : DOG
Output : 0
Enter the word :
STOP
--Ended--
```

Result
compiled and executed in 51.386 sec(s)

```
A B C E
E S F C
X D E E
Enter the word :
CAT
b : CAT
Output : 0
Enter the word :
BASE
b : BASE
Output : 1
Enter the word :
ABSF
b : ABSF
Output : 0
Enter the word :
AFX
b : AFX
Output : 1
Enter the word :
STOP
--Ended--
```

Q2.) Given a singly linked list

$$L: L0 \rightarrow L1 \rightarrow \ldots \rightarrow Ln\text{-}1 \rightarrow Ln,$$

reorder it to:

$$L0 \rightarrow Ln \rightarrow L1 \rightarrow Ln\text{-}1 \rightarrow L2 \rightarrow Ln\text{-}2 \rightarrow \ldots$$

You must do this in-place without altering the nodes' values.

CODE:

```
class ReorderLL {

    static Node head;
```

```
static class Node {

    int data;
    Node next;
    Node(int d)
    {
        data = d;
        next = null;
    }
}

void printingLL(Node node)
{
    if (node == null) {
        return;
    }
    while (node != null) {
        System.out.print(node.data + " -> ");
        node = node.next;
    }
}
Node reverseLL(Node node)
{
    Node prev = null, curr = node, next;
    while (curr != null) {
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    node = prev;
    return node;
}

void rearrange(Node node)
{


    Node slow = node, fast = slow.next;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    Node node1 = node;
```

```
        Node node2 = slow.next;
        slow.next = null;
        node2 = reverseLL(node2);
        node = new Node(0);
        Node curr = node;
        while (node1 != null || node2 != null) {
            if (node1 != null) {
                curr.next = node1;
                curr = curr.next;
                node1 = node1.next;
            }
            if (node2 != null) {
                curr.next = node2;
                curr = curr.next;
                node2 = node2.next;
            }
        }
        node = node.next;
    }

    public static void main(String[] args)
    {
        ReorderLL list = new ReorderLL();
        list.head = new Node(1);
        list.head.next = new Node(2);
        list.head.next.next = new Node(3);
        list.head.next.next.next = new Node(4);
        list.head.next.next.next.next = new Node(5);

        list.printingLL(head);
        list.rearrange(head);
        System.out.println("");
        list.printingLL(head);
        }
}
```

Output:

Result

CPU Time: 0.12 sec(s), Memory: 33576 kilobyte(s)

```
1 -> 2 -> 3 -> 4 -> 5

After Reordering Linked List :
1 -> 5 -> 2 -> 4 -> 3
```