Experiment 3.2

Student Name: SANJIV GUPTA
UID: 21BCS3478
Branch:BE-CSE
Section/Group: IoT-602/B
Semester:5
Date of Performance:3-11-2023
Subject Name: Advance Programming Lab
Subject Code: 21CSP-314

1. Aim:

Implement the problems based on backtracking.

2. Objective:

1. You are given a number . In one operation, you can either increase the value of by 1 or decrease the value of by 1. Determine the minimum number of operations required (possibly zero) to convert number to a number such that binary representation of is a palindrome. Note: A binary representation is said to be a palindrome if it reads the same from left-right and right-left.

2.A 10 X 10 Crossword grid is provided to you, along with a set of words (or names of places) which need to be filled into the grid. Cells are marked either + or -. Cells marked with a - are to befilled with the word list.

3. Script and Output:

Program 1:-

```
#include <bits/stdc++.h>

using namespace std;

typedef long long int ll;

#define mp make_pair

#define pb push_back

#define pob pop_back()

#define mod 1000000007

#define max INT_MAX

#define min INT_MIN

#define fi first
```

```
#define se second
#define fast_cin() ios_base::sync_with_stdio(false);
cin.tie(NULL);
cout.tie(NULL)
set < int > v;
void binarypalindrome(int s, int e, int x) {
  if (s > e) {
    v.insert(x);
    return;
  }
  binarypalindrome(s + 1, e - 1, x);
  if (s == e)
    binarypalindrome(s + 1, e - 1, x + pow(2, s));
  else
    binarypalindrome(s + 1, e - 1, x + pow(2, s) + pow(2, e));
  return;
}
int main() {
  fast_cin();
  int n, t;
  v.insert(0);
  v.insert(1);
  v.insert(3);
  for (int i = 3; i < 32; i++)
    { int c = pow(2, i - 1) +
    1;
    binarypalindrome(1, i - 2, c);
  }
  cin >> t;
```

```
  while (t--) {
    cin >> n;
    auto ptr = v.lower_bound(n);
    auto ptr2 = ptr--;
    if (abs(n - * ptr) < abs(n - *
    ptr2)) cout << abs(n - * ptr) <<
    endl; else
      cout << abs(n - * ptr2) << endl;
  }
  return 0;
}
```

Output:-

Program 2:-

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    class Point {
        boolean isVertical;
        int length;
        int x;
        int y;
        Point(boolean v, int l, int i, int j) {
            isVertical = v;
              length = l;
```

```
        x = i;
        y = j;
    }
    public String toString() {
        return "v="+isVertical+",l="+length+",x="+x+",y="+y;
    }
};
public static boolean isBorder(char board[][], int i, int j) {
    if (i<0 || i >= 10)
        return true;
    if (j<0 || j >= 10)
        return true;
    char c = board[i][j];
    if(c == '+')
        return true;
    else return false;
}
public boolean canInsert(char board[][], Point p, String word, boolean insert) {
    if (p.length != word.length())
        return false;
    for(int i=0; i < word.length(); i++) {
        char c = word.charAt(i);
        int x = p.x;
        int y = p.y;
        if(p.isVertical)
            x = x+i;
        else
            y = y+i;
        if(board[x][y] != '-' && board[x][y] != c)
            return false;
        else {
            if(insert)
                board[x][y] = c;
        }
    }
    return true;
}

public void showBoard(char board[][]) {
    for (int i=0; i < 10; i++) {
        for (int j=0; j < 10; j++) {
            System.out.print(board[i][j]);
        }
        System.out.println();
    }
}
public char[][] copyBoard(char board[][]) {
    char[][] newBoard = new char[10][10];
    for(int i=0; i<10; i++) {
        for (int j=0; j<10; j++) {
            newBoard[i][j] = board[i][j];
        }
    }
```

```
    return newBoard;
}
public boolean solve(char board[][], LinkedList<Point> points, LinkedList<String> wordList) {

    // if no points, and no words then we are successful so print board
    if(points.size() == 0 && wordList.size() == 0) {
        showBoard(board);
        return true;
    }
    if(points.size() == 0 && wordList.size() > 0) {
        return false;
    }
    LinkedList<String> triedWords = new LinkedList<String>();
    Point p = points.removeFirst();
    Iterator<String> iter = wordList.iterator();
    while(iter.hasNext()) {
        String word = iter.next();
        if(canInsert(board, p, word, false)) {
            char[][] newBoard = copyBoard(board);
            canInsert(newBoard, p, word, true);
            iter.remove();
            LinkedList<String> both = new LinkedList<String>();
            both.addAll(wordList);
            both.addAll(triedWords);
            boolean sts = solve(newBoard, points, both);
            if (sts)
                return true;
            else {
                //System.out.println("Reverse insert " + word + " at p" + p);
                //showBoard(board);
                triedWords.push(word);
            }
        } else {
            //System.out.println("Fail insert " + word + " at p" + p);
        }

    }
    points.addFirst(p);
    return false;
}
public LinkedList<Point> getStarts(char board[][]) {
    LinkedList<Point> plist = new LinkedList<Point>();
    for(int i=0; i<10; i++) {
        for (int j=0; j<10; j++) {
            char c = board[i][j];
            if(c == '-') {
                if(isBorder(board, i-1, j) && !isBorder(board, i+1, j)) {
                    int l=0;
                    while(!isBorder(board, i+l, j))
                        l++;
                    //System.out.println(l + " long vertical at " + i + "," + j);
                    Point p = new Point(true, l, i, j);
                    plist.add(p);
                }
```

```
                if(isBorder(board, i, j-1) && !isBorder(board, i, j+1)) {
                    int l=0;
                    while(!isBorder(board, i, j+l))
                        l++;
                    //System.out.println(l + " long horizontal at " + i + "," + j);
                    Point p = new Point(false, l, i, j);
                    plist.add(p);
                }
            }
        }
    }
    return plist;
}
public void myMain(String[] args) {
    Scanner scanner = new Scanner(System.in);
    char board[][] = new char[10][10];
    for (int i=0; i < 10; i++) {
        String line = scanner.nextLine();
        for (int j=0; j < 10; j++) {
            board[i][j] = line.charAt(j);
        }
    }
    String wordLine = scanner.nextLine();
    String words[] = wordLine.split(";");
    LinkedList<String> wordList = new LinkedList<String>();
    for (int i=0; i < words.length; i++) {
        wordList.add(words[i]);
    }
    LinkedList<Point> starts = getStarts(board);
    solve(board, starts, wordList);
}
public static void main(String[] args) {
    Solution s = new Solution();
    s.myMain(args);
}
}
```

Output:-