BACK TRACKING ALGORITHMS

Content

12.1. Rat in a Maze Problem

Statement : Consider a rat placed at **(0, 0)** in a square **matrix of order N*N**. It has to reach the destination at **(n-1, n-1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right).

```
class GfG{
   static boolean isSafe(int x,int y,int m[][],boolean vis[][],int n){
      if(x==-1 || y==-1 || x==n || y==n || vis[x][y] || m[x][y]==0)
         return false;
      return true;
   }
   static void utilFun(int m[][],int n,int x,int y,ArrayList<String> res,String path,boolean
vis[][]){
      if(x==n-1 && y==n-1){
         res.add(path);
         return;
      }
      vis[x][y] = true;
      //down
      if(isSafe(x+1,y,m,vis,n)){
         path += 'D';
         utilFun(m,n,x+1,y,res,path,vis);
         path = path.substring(0,path.length()-1);
      }
      //left
      if(isSafe(x,y-1,m,vis,n)){
         path += 'L';
         utilFun(m,n,x,y-1,res,path,vis);
         path = path.substring(0,path.length()-1);
      }
      //right
      if(isSafe(x,y+1,m,vis,n)){
         path += 'R';
         utilFun(m,n,x,y+1,res,path,vis);
         path = path.substring(0,path.length()-1);
      }
      //upper
      if(isSafe(x-1,y,m,vis,n)){
         path += 'U';
```

```
            utilFun(m,n,x-1,y,res,path,vis);
            path = path.substring(0,path.length()-1);
        }

        vis[x][y] = false;
    }
    public static ArrayList<String> printPath(int[][] m, int n)
    {
        ArrayList<String> res = new ArrayList<>();
        boolean vis[][] = new boolean[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                vis[i][j] = false;
            }
        }
        String path="";
        utilFun(m,n,0,0,res,path,vis);
        return res;
    }
}
```

12.2. Word Boggle

Statement : Given a dictionary, a method to do lookup in dictionary and a M x N board where every cell has one character. Find all possible words that can be formed by a sequence of adjacent characters. Note that we can move to any of 8 adjacent characters, but a word should not have multiple instances of same cell.

Example:

```
Input: dictionary[] = {"GEEKS", "FOR", "QUIZ", "GO"};
      boggle[][]   = {{'G','I','Z'},
                      {'U','E','K'},
                      {'Q','S','E'}};

Output:  Following words of dictionary are present
        GEEKS, QUIZ
.
class GFG {
    static int n,m;
```

```
    private static void findWord(char boggle[][],boolean visited[][],HashSet<String>
dic,HashSet<String> ans,int i,int j,String str){
        visited[i][j] = true;
        str += boggle[i][j];
        if(dic.contains(str)){
            ans.add(str);
        }
        //traverse all surrounded character
        for(int row=i-1;row<=i+1 && row<n;row++){
            for(int col=j-1;col<=j+1 && col<m;col++){
                if(row>=0 && col>=0 && !visited[row][col]){
                    findWord(boggle, visited, dic, ans, row, col, str);
                }
            }
        }
        str = "" + str.charAt(str.length()-1);
        visited[i][j] = false;
    }
    public static void main (String[] args) {
                Scanner sc = new Scanner(System.in);
                int test = sc.nextInt();
                for(int t=0;t<test;t++){
                    int len = sc.nextInt();
                    HashSet<String> dic = new HashSet<>();
                    for(int i=0;i<len;i++){
                        dic.add(sc.next());
                    }
                    n = sc.nextInt();
                    m = sc.nextInt();
                    char boggle[][] = new char[n][m];
                    for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
```

```
                boggle[i][j] = sc.next().charAt(0);
            }
        }
        //processing
        HashSet<String> ans = new HashSet<>();
        boolean visited[][] = new boolean[n][m];
        String str = "";
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                findWord(boggle, visited, dic, ans, i, j, str);
            }
        }
        if(ans.size()==0){
            System.out.println(-1);
        }else{
            Iterator iter = ans.iterator();
            while(iter.hasNext()){
                System.out.print(iter.next()+" ");
            }
        }
        System.out.println();
    }
  }
}
```

12.3. Generate IP Addresses

Statement : Given a string **S** containing only digits, Your task is to complete the function **genIp()** which returns a vector containing all possible combination of valid IPv4 ip address and takes only a string **S** as its only argument.

A valid IP address must be in the form of A.B.C.D, where A, B, C, and D are numbers from 0-255. The numbers cannot be 0 prefixed unless they are 0
Note : Order doesn't matter.

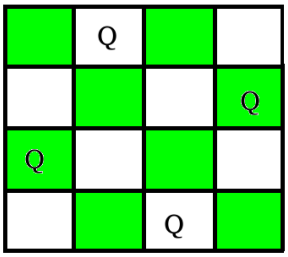
For string 11211 the ip address possible are
1.1.2.11
1.1.21.1
1.12.1.1
11.2.1.1

```
    private boolean isValid(String str){
        String a[] = str.split("[.]");
        for(String s:a){
            int i = Integer.parseInt(s);
            if(s.length()>3 || i<0 || i>255)
                return false;
            if(s.length()>1 && i==0)
                return false;
            if(s.length()>1 && i!=0 && s.charAt(0)=='0')
                return false;
        }
        return true;
    }
    public ArrayList<String> genIp(String s) {
        ArrayList<String> list = new ArrayList<>();
        if(s.length()<0 || s.length()>12)
            return list;
        else{
            int size = s.length();
            String snew = s;
            for(int i=1;i<size-2;i++){  //for placing first dot
                for(int j=i+1;j<size-1;j++){  //for placing second dot
                    for(int k=j+1;k<size;k++){  //for placing third dot
                        snew = snew.substring(0,k)+"."+snew.substring(k);
                        snew = snew.substring(0,j)+"."+snew.substring(j);
                        snew = snew.substring(0,i)+"."+snew.substring(i);
                        if(isValid(snew)){
```

```
                        list.add(snew);
                    }
                    snew = s;
                }
            }
        }
        return list;
    }
```

12.4. N-Queen Problem

Statement : The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other. Given an integer n, print all distinct solutions to the n-queens puzzle. Each solution contains distinct board configurations of the n-queens' placement, where the solutions are a permutation of [1,2,3..n] in increasing order, here the number in the *ith* place denotes that the *ith*-column queen is placed in the row with that number. For eg below figure represents a chessboard [3 1 4 2].




```
class GFG {
    static class Flag{
        int data;
        Flag(int data){
            this.data = data;
        }
    }
    static boolean isSafe(int n,int row,int col,boolean vis[][]){
        //for same row
        for(int i=col-1;i>=0;i--){
            if(vis[row][i])
```

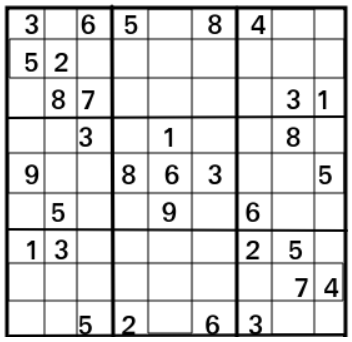
```
            return false;
        }
        //upper diagnol
        for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--){
            if(vis[i][j])
                return false;
        }
        //lower diagnol
        for(int i=row+1,j=col-1;i<n && j>=0;i++,j--){
            if(vis[i][j])
                return false;
        }
        return true;
    }
    static void nQueen(Flag flag,int n,int col,boolean vis[][],int arr[]){
        if(col==n){
            flag.data = 1;
            System.out.print("[");
            for(int i=0;i<n;i++){
                System.out.print(arr[i]+" ");
            }
            System.out.print("]"+" ");
            return;
        }
        for(int i=0;i<n;i++){
            if(isSafe(n,i,col,vis)){
                arr[col] = i+1;
                vis[i][col] = true;
                nQueen(flag,n,col+1,vis,arr);
                vis[i][col] = false;
            }
        }
        if(col==0 && flag.data==0){
            System.out.print(-1);
        }
    }
      public static void main (String[] args) {
            Scanner sc = new Scanner(System.in);
            int test = sc.nextInt();
            for(int t=0;t<test;t++){
                int n = sc.nextInt();
                boolean vis[][] = new boolean[n][n];
```

```
                int arr[] = new int[n];
                Flag flag = new Flag(0);
                nQueen(flag,n,0,vis,arr);
                System.out.println();
            }
        }
}
```

12.5. Solve the Sudoku

Statement : Given an incomplete Sudoku configuration in terms of a 9 x 9 2-D square matrix (mat[][]). The task to print a solved Sudoku. For simplicity you may assume that there will be only one unique solution.

Sample Sudoku:



```
class GFG {
   static boolean isSafe(int row,int col,int no,int arr[][]){
      //same row
      for(int i=0;i<9;i++){
         if(arr[row][i]==no)
            return false;
      }
      //same column
      for(int i=0;i<9;i++){
         if(arr[i][col]==no)
            return false;
      }
      //same cube
      int startRow = row - (row % 3);
      int startCol = col - (col % 3);
      for(int i=startRow;i<startRow+3;i++){
         for(int j=startCol;j<startCol+3;j++){
            if(arr[i][j]==no)
```

```
                return false;
            }
        }
        return true;
    }
    static boolean solveSudoku(int arr[][]){
        int row = -1;
        int col = -1;
        boolean isEmpty = true;
        for(int i=0;i<9;i++){
            for(int j=0;j<9;j++){
                if(arr[i][j]==0){
                    row = i;
                    col = j;
                    isEmpty = false;
                    break;
                }
            }
            if(!isEmpty){
                break;
            }
        }
        if(isEmpty){
            return true;
        }
        //try for 1 to 9 no.
        for(int no=1;no<=9;no++){
            if(isSafe(row,col,no,arr)){
                arr[row][col] = no;
                if(solveSudoku(arr))
                    return true;
                arr[row][col] = 0;
            }
        }
        return false;
    }
    static void print(int arr[][]){
        for(int i=0;i<arr.length;i++){
            for(int j=0;j<arr.length;j++)
                System.out.print(arr[i][j]+" ");
        }
    }
```

```
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
            int arr[][] = new int[9][9];
            for(int i=0;i<9;i++){
                for(int j=0;j<9;j++){
                    arr[i][j] = sc.nextInt();
                }
            }
            solveSudoku(arr);
            print(arr);
            System.out.println();
        }
    }
}
```