DSA ASSIGNMENT-8(RECURSION)

Solution 1:

```
class Solution {
    public int sum(String s,int v)
    {
        int su=0;
        for(int i=v;i<s.length();i++)
        {
            su+=(int)s.charAt(i);
        }
        return su;
    }
    public int minimumDeleteSum(String s1, String s2,int v1,int v2,int[][]dp)
    {
        if(v1==s1.length()&&v2==s2.length())
        return 0;
        if(v1==s1.length()&&v2!=s2.length())
        {
            return sum(s2,v2);
        }
        if(v1!=s1.length()&&v2==s2.length())
        {
            return sum(s1,v1);
        }
        char ch=s1.charAt(v1);
        char ch1=s2.charAt(v2);

        if(dp[v1][v2]!=-1)
        return dp[v1][v2];
        if(ch==ch1)
        {
           dp[v1][v2]=minimumDeleteSum(s1,s2,v1+1,v2+1,dp);
        }
        else
        {
            int a=minimumDeleteSum(s1,s2,v1+1,v2,dp)+(int)(s1.charAt(v1));
            int b= minimumDeleteSum(s1,s2,v1,v2+1,dp)+(int)(s2.charAt(v2));
           dp[v1][v2]= Math.min(a,b);
        }
        return dp[v1][v2];

    }
    public int minimumDeleteSum(String s1,String s2)
    {
        int dp[][]=new int[s1.length()][s2.length()];
        for(int i=0;i<s1.length();i++)
        {
```

```
            for(int j=0;j<s2.length();j++)
            {
                dp[i][j]=-1;
            }
        }
        return minimumDeleteSum(s1,s2,0,0,dp);
    }
}
```

Solution 2:

```
class Solution {
    public boolean checkValidString(String s) {
        int leftMax=0, leftMin = 0;

        for(int i=0;i<s.length();i++) {

            if(s.charAt(i) == '(') {
                leftMax++;
                leftMin++;
            } else if (s.charAt(i) == ')') {
                leftMin--;
                leftMax--;
            } else {
                leftMin--;
                leftMax++;
            }

            if(leftMin < 0) leftMin = 0;
            if(leftMax < 0) return false;
        }

        return leftMin == 0;
    }
}
```

Solution 3:

```
class Solution {
    Integer dp[][]=new Integer[501][501];
    public int minDistance(String w1, String w2) {
      return solve(w1,w2,0,0)  ;
    }
    int solve(String w1,String w2,int i,int j){
        if(i==w1.length() && j==w2.length())return 0;  //both reached end,no
deltetion
        if(i==w1.length()){ //w1 is at end
            return w2.length()-j;
        }
        if(dp[i][j]!=null)return dp[i][j];
```

```
        if(j==w2.length())return w1.length()-i;//w2 at end
        if(w1.charAt(i)==w2.charAt(j)){//no deletions required
            return dp[i][j]=solve(w1,w2,i+1,j+1);
        }else{            //choices for deletions
            return dp[i][j]=Math.min(1+solve(w1,w2,i+1,j),1+solve(w1,w2,i,j+1));
        }
    }
}
```

Solution 4:

```
class Solution {
    public String tree2str(TreeNode root) {
         if(root==null) return "()";
        String res = "";
        res = res+root.val;
        if(root.left!=null && root.right!=null){
            res=res+"("+tree2str(root.left)+")"+"("+tree2str(root.right)+")";
        }
        else if(root.left!=null){
            res=res+"("+tree2str(root.left)+")";
        }
        if(root.left==null && root.right!=null){
            res=res+"()"+"("+tree2str(root.right)+")";
        }
        return res;
    }
}
```

Solution 5:

```
class Solution {
    public int compress(char[] chars) {
        StringBuilder output= new StringBuilder();
        int count=1;
        if(chars.length==1){
            return 1;
        }

        for(int i=0;i<chars.length;i++) {
                if (i+1<chars.length && chars[i] == chars[i + 1]) {
                    count++;
                }
                else{
                    if(count==1){
                        output.append(chars[i]);
                    }
                    else {
                        output.append(chars[i]).append(count);
                    }
```

```
                    count=1;
                }

        }


        for(int i=0;i<output.length();i++){
            chars[i]= output.charAt(i);
        }

        return output.length();
    }
}
```

Solution 6:

```
class Solution {
    public List<Integer> findAnagrams(String s, String p) {
        ArrayList<Integer>ans=new ArrayList<>();
        HashMap<Character,Integer>map=new HashMap<>();
        for(int i=0;i<p.length();i++){
            map.put(p.charAt(i),map.getOrDefault(p.charAt(i),0)+1);
        }
        int i=0,j=0,k=p.length(),count=map.size(),n=s.length();

        while(j<n){
            if(map.containsKey(s.charAt(j))){
                map.put(s.charAt(j),map.get(s.charAt(j))-1);

            if(map.get(s.charAt(j))==0){
                count--;
            }}
            if(j+1-i<k){
                j++;
            }
            else{
                if(count==0){
                    ans.add(i);
                }
                if(map.containsKey(s.charAt(i))){
                    map.put(s.charAt(i),map.get(s.charAt(i))+1);

              if(map.get(s.charAt(i))==1) {

                    count++;
                }}
                i++;
                j++;
            }
        }
        return ans;
```

```
    }
}
```

Solution 7:

```
class Solution {
    public String decodeString(String s) {
        Stack <Integer> counts = new Stack<>();// stores digit
        Stack <String> result = new Stack<>();//stores result
        String res ="";
        int idx=0;

      while(idx<s.length()){
        if(Character.isDigit(s.charAt(idx))){// for digits
            int count = 0;
            while(Character.isDigit(s.charAt(idx))){// for digits more than 1
              count = 10*count +(s.charAt(idx)-'0');
                idx+=1;
            }
            counts.push(count);
        }else if(s.charAt(idx) == '['){// when [ starts
            result.push(res);
            res="";
            idx+=1;
        }else if(s.charAt(idx) == ']'){// when ] ends
            StringBuilder temp = new StringBuilder(result.pop());
            int count = counts.pop();
            for(int i=0;i<count;i++){
                temp.append(res);// putting values back in res
            }
            res= temp.toString();
            idx+=1;
        }else{
            res+= s.charAt(idx);// for letters
            idx+=1;
        }
      }
      return res;
    }
}
```

Solution 8:

```
class Solution {
    public boolean buddyStrings(String s, String goal) {
        // Check if lengths of s and goal are the same
        if (s.length() != goal.length()) {
            return false;
        }
```

```
        // If s and goal are equal, check if there are any repeated characters
        if (s.equals(goal)) {
            int[] charCount = new int[26];
            for (char c : s.toCharArray()) {
                charCount[c - 'a']++;
                if (charCount[c - 'a'] > 1) {
                    return true;
                }
            }
            return false;
        }

        // Find positions of exactly two mismatches
        int firstMismatch = -1;
        int secondMismatch = -1;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) != goal.charAt(i)) {
                if (firstMismatch == -1) {
                    firstMismatch = i;
                } else if (secondMismatch == -1) {
                    secondMismatch = i;
                } else {
                    // More than two mismatches found, return false
                    return false;
                }
            }
        }

        // Check if exactly two mismatches were found and if they are swappable
        return (secondMismatch != -1 && s.charAt(firstMismatch) ==
goal.charAt(secondMismatch)
                && s.charAt(secondMismatch) == goal.charAt(firstMismatch));
    }
}
```