Assignment -08

1. 859. Buddy Strings

Given two strings s and goal, return true if you can swap two letters in s so the result is equal to goal, otherwise, return false.

Swapping letters is defined as taking two indices i and j (0-indexed) such that i != j and swapping the characters at s[i] and s[j].

```
//Time Complexity:O(n)

//Space Complexity: O(n)

class Solution {

public:

    bool checkFreq(string s)

    {

        map<int,int>mp;

       for(auto x:s)

       {

           mp[x]++;

           if(mp[x]>1)

           {

               return true;

           }

       }

       return false;

    }

    bool buddyStrings(string s, string goal) {
```

```
        if(s.size() != goal.size()) return false;


        if(s==goal)
        {
            return checkFreq(s);
        }
         vector<int> ans;
        for(int i=0;i<s.size();i++)
        {
            if(s[i]!=goal[i])
            {
                ans.push_back(i);
            }
        }
        if(ans.size()!=2)
        {
            return false;
        }
        swap(s[ans[0]],s[ans[1]]);
        return s==goal;
    }
};
```

2. 583. Delete Operation for Two Strings

Given two strings word1 and word2, return the minimum number of steps required to make word1 and word2 the same.

In one step, you can delete exactly one character in either string.

```
 //Time Complexity: O(m*n)

//Space Complexity: O(m*n)


class Solution {

public:

    int minDistance(string word1, string word2) {

        int m=word1.size();//peace->5

        int n=word2.size();//ease//4

        vector<vector<int>>dp(m+1,vector<int>(n+1,0));//6*4

        for(int i=1;i<m+1;i++)

        {

            for(int j=1;j<n+1;j++)

            {

                if(word1[i-1]==word2[j-1])

                {

                    dp[i][j]=1+dp[i-1][j-1];

                }

                else

                {

                    dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
```

```
                    }

                }

            }

            return m-dp[m][n] + n-dp[m][n];

        }

    };
```

3. 678. Valid Parenthesis String

Given a string s containing only three types of characters: '(', ')' and '*', return true if s is valid.

The following rules define a valid string:

- Any left parenthesis '(' must have a corresponding right parenthesis ')'.
- Any right parenthesis ')' must have a corresponding left parenthesis '('.
- Left parenthesis '(' must go before the corresponding right parenthesis ')'.
- '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string
- "".

```
//Time Complexity: O(n)

//Space Complexity: O(n)

class Solution {

public:

    bool checkValidString(string s) {

        stack<int>open,star;

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

        {

            if(s[i]=='(')

            {

                open.push(i);

            }

            else if(s[i]=='*')

            {

                star.push(i);

            }

            else
```

```
        {
            if(!open.empty())
            {
                open.pop();
            }
            else if(!star.empty())
            {
                star.pop();
            }
            else
            {
                return false;
            }
        }
    }
    //We have some opening brackets still left
    while(!open.empty())
    {
        if(star.empty())
        {
            return false;
        }
        else if(open.top()<star.top())
        {
```

```
                star.pop();
                open.pop();
            }
            else
            {
                return false;
            }
        }
        return true;
    }
};
```

4. 712. Minimum ASCII Delete Sum for Two Strings

Given two strings s1 and s2, return the lowest ASCII sum of deleted characters to make two strings equal.

```
//Time Complexity: O(m*n)

//Space Complexity: O(m*n)

class Solution {

public:

    int minimumDeleteSum(string s1, string s2) {

        int m=s1.size()+1;

        int n=s2.size()+1;



        vector<vector<int>>dp(m+1,vector<int>(n+1));

        for(int i=1;i<m+1;i++)

        {

            dp[i][0]=dp[i-1][0]+s1[i-1];

        }

         for(int j=1;j<n+1;j++)

        {

            dp[0][j]=dp[0][j-1]+s2[j-1];

        }



        for(int i=1;i<m+1;i++)

        {

            for(int j=1;j<n+1;j++)
```

```
                {
                    if(s1[i-1]==s2[j-1])
                    {
                        dp[i][j]=dp[i-1][j-1];
                    }
                    else
                    {
                        dp[i][j]=min(dp[i-1][j]+s1[i-1],dp[i][j-1]+s2[j-1]);
                    }
                }
            }
            return dp[m][n];
        }
};
```