EASY :

MEDIUM :

1. Coin change problem

```
class Solution {
  public:
    long long int count(int S[], int m, int n) {

       long long int arr2[m+1][n+1];
        for(int i=0;i<m+1;i++)
        {
           for(int j=0;j<n+1;j++)
           {
              if(i==0&&j==0||j==0)
              arr2[i][j]=1;
              else
              arr2[i][j]=0;
           }
        }
        for(int i=1;i<m+1;i++)
        {
           for(int j=1;j<n+1;j++)
           {
              if(S[i-1]<=j)
              arr2[i][j]=(arr2[i][j-S[i-1]]+arr2[i-1][j]);
              else
              arr2[i][j]=arr2[i-1][j];
           }
        }
       return arr2[m][n];
    }
};
```

2. 0-1 knapsack problem

```
class Solution
{
    public:
    int knapSack(int x, int arr[], int arr1[], int n)
    {
     int arr2[n+1][x+1];
     for(int i=0;i<=n;i++)
     {
```

```
        for(int j=0;j<=x;j++)
        arr2[i][j]=0;
    }
    for(int i=1;i<n+1;i++)
    {
        for(int j=1;j<x+1;j++)
        {
        if(arr[i-1]<=j)
        arr2[i][j]=max(arr1[i-1]+arr2[i-1][j-arr[i-1]],arr2[i-1][j]);
        else
        arr2[i][j]=arr2[i-1][j];
        }
    }
    return arr2[n][x];
   }
};
```

3. Edit distance

```
class Solution {
  public:
    int editDistance(string s, string t) {

        int n=s.size();
        int m=t.size();
        int dp[n+1][m+1];

        for(int i=0;i<=n;i++)
        {
            for(int j=0;j<=m;j++)
            {
                if(i==0)
                dp[i][j]=j;

                else if(j==0)
                dp[i][j]=i;

                else if(s[i-1]==t[j-1])
                dp[i][j]=dp[i-1][j-1];

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

```
    }
};
```

4. Subset sum problem

```
class Solution{
public:
    bool solve(int arr[],int n,int sum)
    {
        if(sum==0)
        return 1;

        if(n==0 && sum!=0)
        return 0;

        if(arr[n]>sum)
        return solve(arr,n-1,sum);

        return solve(arr,n-1,sum) || solve(arr,n-1,sum-arr[n]);
    }
    int equalPartition(int N, int arr[])
    {
        int sum=0;

        for(int i=0;i<N;i++)
        sum+=arr[i];

        if(sum%2!=0)
        return 0;

        if(solve(arr,N-1,sum/2))
        return 1;

        return 0;
    }
};
```

5. Longest common subsequence

```
class Solution
{
    public:
    int lcs(int x, int y, string s1, string s2)
    {
        vector<vector<int>>dp(x+1,vector<int>(y+1));
```

```
        for(int i=1;i<=x;i++)
        for(int j=1;j<=y;j++)
        dp[i][j]=(s1[i-1]==s2[j-1])?dp[i-1][j-1]+1:max(dp[i-1][j],dp[i][j-1]);

        return dp[x][y];
    }
};
```

6. Longest increasing subsequence

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums)
    {
        set<int>st;
        int n=nums.size();
        for(int i=0;i<n;i++)
        {
            auto it=st.upper_bound(nums[i]);
            auto it1=st.find(nums[i]);
            if(it!=st.end()&&it1==st.end())
            {
                st.erase(it);
                st.insert(nums[i]);
            }
            else
            st.insert(nums[i]);
        }
        return st.size();
    }
};
```

7. Maximum sum increasing subsequence

```
class Solution{
public:
        int maxSumIS(int arr[], int n)
        {
            vector<int>dp(n,0);
            dp[0]=arr[0];
            int ans=arr[0];
             for(int i=1;i<n;i++)
             {
                 int maxx=INT_MIN;
                 for(int j=0;j<i;j++)
```

```
            {
                if(arr[j]<arr[i])
                maxx=max(maxx,dp[j]);
            }

            if(maxx!=INT_MIN)
            dp[i]=arr[i]+maxx;
            else
            dp[i]=arr[i];

            ans=max(ans,dp[i]);
        }
        return ans;
    }
};
```

8. Egg dropping problem

```
class Solution
{
  public:
  int eggDrop(int n, int k)
  {
     swap(n,k);
     int dp[k+1][n+1];

     for(int i=0; i<=n;i++)
     dp[0][i]=INT_MAX;
     for(int i=0;i<=n;i++)
     dp[1][i]=i;
     for(int i=0;i<=k;i++)
     dp[i][0]=0;
     for(int i=0;i<=k;i++)
     dp[i][1]=1;

     for(int i = 2; i <= k; i++)
        for(int j = 2; j <= n; j++)
        {
           int l = 1;
           int r = j;
           int temp = 0;
           int ans = INT_MAX;
            while(l<=r)
            {
              int mid=(l+r)/2;
```

```
                int left=dp[i-1][mid-1];
                int right=dp[i][j-mid];
                temp=1+max(left,right);
                if(left<right)
                l=mid+1;
                else
                r=mid-1;
                ans=min(ans,temp);
              }
              dp[i][j]=ans;
          }
       return dp[k][n];
    }
};
```

HARD :

1. Word break problem

```
class Solution
{
public:
    int wordBreak(string A, vector<string> &B)
    {

        unordered_set<string>st(B.begin(), B.end());
        int n = A.length();
        vector<bool>dp(n+1, false);
        dp[n] = 1;

       for(int i=n-1;i>=0;i--)
       {
          string res = "";
          for(int j=i;j<n;j++)
          {
             res+=A[j];
             if(st.find(res)!=st.end())
             dp[i]=dp[i] || dp[j+1];
           }
       }
       return dp[0];
    }
};
```

2. Palindrome partitioning problem

```
class Solution{
public:
int dp[585][585];
   bool isPalindrome(string String, int i, int j)
   {
         while(i<j)
         if(String[i++] != String[j--])
         return false;
         return true;
   }
   int min_Partion(string &String, int i, int j)
   {
      cout<<i<<j<<endl;
      if(dp[i][j]!=-1)
      return dp[i][j];

      if(i>=j || isPalindrome(String, i, j))
      return 0;

      int ans = INT_MAX;
      int c=0;

      for(int k=i;k<j;k++)
      {
        c = 1 + min_Partion(String, i, k) + min_Partion(String, k + 1, j);
        ans=min(ans, c);
      }

      return dp[i][j]=ans;
   }
   int palindromicPartition(string str)
   {
      memset(dp,-1,sizeof(dp));
      return min_Partion(str,0,str.size()-1);
      // code here
   }
};
```