DYNAMIC PROGRAMMING

1. FIBONACCI NUMBERS

MEMOIZATION

```
#include <iostream>
#include <string.h>
using namespace std;
int memo[1000000];
int fib(int n)
{
    if(memo[n]==-1)
    {
        int res;
        if(n==0 || n==1)
            return n;
        else
           { res = fib(n-1)+fib(n-2);

           }
           memo[n]=res;
    }
    return memo[n];
}
int main() {
         int n = 5;
         memset(memo, -1, sizeof(memo));
         cout<<fib(5);
}
```

TABULATION

```
int fib(int n)
{
    int f[n+1];

    f[0]=0;
    f[1]=1;

    for(int i=2;i<=n;i++)
    {
        f[i] = f[i-1] + f[i-2];
    }


return f[n];

}
```

2. LONGEST COMMON SUBSEQUENCE

MEMOIZATION   T:theta(m*n)

```
#include <iostream>
#include <string.h>
using namespace std;

int memo[1000][1000];

int lcs(string s1, string s2, int n, int m)
{
    if(memo[n][m]!=-1)
        return memo[n][m];
    if(n==0 || m==0)
        memo[n][m]=0;

    else
    {
        if(s1[n-1]==s2[m-1])
            memo[n][m] = 1 + lcs(s1,s2,n-1,m-1);
        else
            memo[n][m] = max(lcs(s1,s2,n-1,m),lcs(s1,s2,n,m-1));
    }

    return memo[n][m];

}

int main() {


         string s1="AXYZ", s2="BAZ";

         int n = 4, m = 3;

         memset(memo,-1,sizeof(memo));

         cout<<lcs(s1,s2,n,m);


}
```

TABULATION

```
#include <iostream>
#include <string.h>
using namespace std;
```

```
int lcs(string s1, string s2)
{
    int m = s1.length(), n = s2.length();

    int dp[m+1][n+1];

    for(int i=0;i<=m;i++)
        dp[i][0]=0;

    for(int j=0;j<=n;j++)
        dp[0][j]=0;

    for(int i=1; i<=m; i++)
    {
        for(int j=1; j<=n; j++)
        {
            if(s1[i-1]==s2[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 dp[m][n];

}

int main() {


         string s1="AXYZ", s2="BAZ";
        cout<<lcs(s1,s2);


}
```

3. **COIN CHANGE COUNT COMBINATIONS**

```
#include <iostream>
#include <string.h>
using namespace std;


int getCount(int coins[], int n, int sum)
{
    int dp[sum+1][n+1];

    for(int i=0;i<=n;i++)
    {
        dp[0][i]=1;
    }

    for(int j=0;j<=sum;j++)
    {
        dp[j][0]=0;
    }

    for(int i=1;i<=sum;i++)
    {
        for(int j=1;j<=n;j++)
        {
            dp[i][j] = dp[i][j-1];

            if(coins[j-1]<=i)
            dp[i][j]+=dp[i-coins[j-1]][j];
        }
    }

    return dp[sum][n];

}

int main() {


int coins[]={1, 2, 3}, sum=4, n=3;

cout<<getCount(coins, n, sum);


}
```

4. EDIT DISTANCE PROBLEM

```
#include <iostream>
#include <string.h>
using namespace std;


int eD(string s1, string s2, int m, int n)
{
    if(m==0)
        return n;
    if(n==0)
        return m;

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

}

int main() {


string s1="CAT", s2="CUT";
int n=3, m=3;

cout<<eD(s1,s2,m,n);


}
```

DP BASED SOLUTION (TABULATION)

```
#include <iostream>
#include <string.h>
using namespace std;


int eD(string s1, string s2, int m, int n)
{
    int dp[m+1][n+1];

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

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

            }
        }
    }

    return dp[m][n];

}

int main() {


string s1="CAT", s2="CUT";
int n=3, m=3;

cout<<eD(s1,s2,m,n);



}
```

5. LONGEST INCREASING SUBSEQUENCE

```
#include <iostream>
#include <string.h>
using namespace std;


int LIS( int arr[], int n )
{
    int lis[n];

    lis[0] = 1;


    for (int i = 1; i < n; i++ )
    {
        lis[i] = 1;
        for (int j = 0; j < i; j++ )
            if ( arr[i] > arr[j])
                lis[i] = max(lis[i], lis[j] + 1);
    }

    int res = lis[0];

    for(int i=0;i<n;i++)
    {
        res = max(lis[i], res);
    }

    return res;

}
```

```
int main() {
int arr[] ={3, 4, 2, 8, 10, 5, 1};
int n = 7;
cout<<LIS(arr, n);
}
```

O(NLOGN) SOLUTION:

```
#include <iostream>
#include <string.h>
using namespace std;


        int ceilIdx(int tail[], int l, int r, int key)
        {
            while (r > l) {
                int m = l + (r - l) / 2;
                if (tail[m] >= key)
                    r = m;
                else
                    l = m+1;
            }

            return r;
        }

    int LIS(int arr[], int n)
    {


        int tail[n];
        int len =1;

        tail[0] = arr[0];
        for (int i = 1; i < n; i++) {

            if(arr[i] > tail[len - 1])
            {
                tail[len] = arr[i];
                len++;
            }
            else{
                int c = ceilIdx(tail, 0, len - 1, arr[i]);

                tail[c] = arr[i];
            }
        }

        return len;
    }


int main() {


 int arr[] ={3, 10, 20, 4, 6, 7};
  int n = 6;

cout<<LIS(arr, n);


}
```

6. MAXIMUM SUM INCREASING SUBSEQUENCE

```
#include <iostream>
```

```
using namespace std;

int MSIS(int arr[], int n)
{
        int msis[n];


        for(int i=0; i<n; i++)
        {
                msis[i]  = arr[i];


                for(int j=0; j<i; j++)
                {
                        if(arr[j] < arr[i])
                        {
                                msis[i] = max(msis[i], arr[i] + msis[j]);
                        }
                }
        }

        int res = msis[0];

        for(int i=0; i<n; i++)
        {
                res = max(res, msis[i]);
        }

        return res;
}


int main() {

        int n = 3;
        int arr[] = {5, 10, 20};

    cout<<MSIS(arr, n);

    return 0;
}
```

7. MAXIMUM CUTS

```
#include <iostream>
#include <string.h>
using namespace std;




    int maxCuts(int n, int a, int b, int c)
    {

     int dp[n+1];

     dp[0] =0;

     for(int i = 1; i<=n; i++)
     {
         dp[i] = -1;

         if(i-a >=0) dp[i] = max(dp[i],dp[i-a]);

         if(i-b >=0) dp[i] = max(dp[i],dp[i-b]);

         if(i-c >=0) dp[i] = max(dp[i],dp[i-c]);
         if(dp[i]!=-1)
            dp[i]++;
     }

    return dp[n];

    }


int main() {

int n = 5, a = 1, b = 2, c = 3;

cout<<maxCuts(n, a, b, c);


}
```

8. MINIMUM COINS TO MAKE A VALUE

```
#include <iostream>
#include <string.h>
#include <limits.h>
using namespace std;



  int minCoins(int arr[], int m, int value)
    {

        int dp[value + 1];


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


        for (int i = 1; i <= value; i++)
        {

            for (int j = 0; j < m; j++)
            if (arr[j] <= i)
            {
                int sub_res = dp[i - arr[j]];
                if (sub_res != INT_MAX
                       && sub_res + 1 < dp[i])
                       dp[i] = sub_res + 1;


            }

        }
        return dp[value];

    }


int main() {

int arr[] = {3, 4, 1}, val =5, n =3;

cout<<minCoins(arr, n, val);


}
```

9. MINIMUM JUMPS TO REACH END

```
#include <iostream>
#include <string.h>
#include <limits.h>
using namespace std;



 int minJumps(int arr[], int n)
  {

    int dp[n];
    int i, j;


    dp[0] = 0;


    for (i = 1; i < n; i++) {
      dp[i] = INT_MAX;
      for (j = 0; j < i; j++) {
        if (i <= j + arr[j] && dp[j] != INT_MAX) {
          dp[i] = min(dp[i], dp[j] + 1);
          break;
        }
      }
    }
    return dp[n - 1];
  }
int main() {
int arr[] = {3, 4, 2, 1, 2, 1}, n =6;
cout<<minJumps(arr, n);
}
```

10. 0-1 KNAPSACK PROBLEM

```
#include <iostream>
#include <string.h>
#include <limits.h>
using namespace std;
  int knapSack(int W, int wt[], int val[], int n)
{
    if (n == 0 || W == 0)
        return 0;


    if (wt[n-1] > W)
        return knapSack(W, wt, val, n - 1);


    else
        return max(val[n-1] + knapSack(W - wt[n-1],  wt, val, n - 1),
                            knapSack(W, wt, val, n - 1));
}

int main() {
int val[] = { 10, 40, 30, 50 };
int wt[] = { 5, 4, 6, 3 };
int W = 10;
int n = 4;
cout<<knapSack(W, wt, val, n);
}
```

DP SOLUTION

```
#include <iostream>
#include <string.h>
#include <limits.h>
```

```
using namespace std;



  int knapSack(int W, int wt[], int val[], int n)
{



    int i, j;
    int dp[n + 1][W + 1];

   for(int i=0; i<=W; i++)
   {
       dp[0][i] = 0;
   }

   for(int i=0; i<=n; i++)
   {
       dp[i][0] = 0;
   }

    for (i = 1; i <= n; i++) {
        for (j = 1; j <= W; j++) {
           if (wt[i - 1] > j)
                dp[i][j] = dp[i-1][j];
            else
                dp[i][j] = max(val[i - 1] + dp[i - 1][j - wt[i - 1]], dp[i - 1][j]);
        }
    }

    return dp[n][W];
}


int main() {
```

```
int val[] = { 10, 40, 30, 50 };
int wt[] = { 5, 4, 6, 3 };
int W = 10;
int n = 4;
cout<<knapSack(W, wt, val, n);


}
```

11. OPTIMAL STRATEGY FOR A GAME

```
#include <iostream>
using namespace std;

int sol(int arr[], int n)
{
    int dp[n][n];

    for(int i=0;i<n-1;i++)
    {
        dp[i][i+1]= max(arr[i],arr[i+1]);
    }

    for(int gap =3; gap<n; gap = gap + 2)
    {
        for(int i=0; i+gap<n; i++)
        {
            int j = gap + i;

            dp[i][j] = max((arr[i] + min(dp[i+1][j], dp[i+1][j-1])),
                                            (arr[i] +
min(dp[i+1][j-1], dp[i][j-2])));
        }
    }

    return dp[0][n-1];
}


int main() {

    int n = 4;

        int arr[] = {20, 5, 4, 6};

    cout<<sol(arr, n);

    return 0;
}
```

12. EGG DROPPING PUZZLE

```
#include <bits/stdc++.h>
using namespace std;

// A utility function to get
// maximum of two integers
int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Function to get minimum
// number of trials needed in worst
// case with n eggs and k floors
int eggDrop(int n, int k)
{
    // If there are no floors,
    // then no trials needed.
    // OR if there is one floor,
    // one trial needed.
    if (k == 1 || k == 0)
        return k;

    // We need k trials for one
    // egg and k floors
    if (n == 1)
        return k;

    int min = INT_MAX, x, res;

    // Consider all droppings from
    // 1st floor to kth floor and
    // return the minimum of these
    // values plus 1.
    for (x = 1; x <= k; x++) {
        res = max(
            eggDrop(n - 1, x - 1),
            eggDrop(n, k - x));
        if (res < min)
            min = res;
    }

    return min + 1;
}

// Driver program to test
// to pront printDups
int main()
{
    int n = 2, k = 10;
    cout << "Minimum number of trials "
```

```
                           "in worst case with "
                  << n << " eggs and " << k
                  << " floors is "
                  << eggDrop(n, k) << endl;
         return 0;
}
```

DP SOLUTION

```
#include <iostream>
#include <limits.h>
using namespace std;

int res(int e, int f)
{

    int dp[f+1][e+1];

    for(int i = 1; i <= e ;i++){
        dp[1][i] = 1;
        dp[0][i] = 0;
    }

    for(int j = 1; j <= f; j++){
        dp[j][1] = j;
    }

    for(int i = 2; i <= f; i++){
        for(int j = 2; j <= e; j++){
            dp[i][j] =INT_MAX;
            for(int x = 1; x <= i; x++){
                dp[i][j] = min(dp[i][j], 1 + max(dp[x-1][j-1], dp[i-x][j]));
            }
        }
    }
```

```
  return dp[f][e];

}


int main() {

    int n = 2;

        int f = 10;
    cout<<res(n, f);

    return 0;
}
```

13. COUNT BSTs WITH N KEYS

```
#include <iostream>
#include <limits.h>
using namespace std;

int countBSTs(int n)
{
    int dp[n+1];

    dp[0] = 1;

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

        for(int j=0; j<i; j++)
        {
            dp[i] += dp[j] * dp[i-j-1];
        }
    }

    return dp[n];
}


int main() {

    int n = 4;

    cout<<countBSTs(n);

    return 0;
}
```

14.   MAXIMUM SUM WITH NO 2 CONSECUTIVES

O(1) SPACE

```
#include <iostream>
#include <limits.h>
using namespace std;

int maxSum(int arr[], int n)
{
    if(n==0)
        return arr[0];

    if(n==0)
            return arr[0];
        int prev_prev = arr[0];
        int prev = max(arr[0], arr[1]);
        int res = prev;

        for(int i=3; i<=n; i++)
        {
            res = max(prev, prev_prev + arr[i-1]);

            prev_prev = prev;

            prev = res;
        }

        return res;
}


int main() {

    int n = 5, arr[]= {10, 20, 30, 40, 50};

    cout<<maxSum(arr, n);

    return 0;
}
```

O(N) SPACE

```
#include <iostream>
#include <limits.h>
using namespace std;

int maxSum(int arr[], int n)
{
    if(n==0)
```

```
        return arr[0];

    int dp[n+1];

    dp[1] = arr[0];
    dp[2] = max(arr[0], arr[1]);

    for(int i=3; i<=n; i++)
    {
        dp[i] = max(dp[i-1], dp[i-2] + arr[i-1]);
    }

    return dp[n];
}


int main() {

    int n = 5, arr[]= {10, 20, 30, 40, 50};

    cout<<maxSum(arr, n);

    return 0;
}
```

15. SUBSET SUM PROBLEM

```
#include <iostream>
#include <limits.h>
using namespace std;

int countSubsets(int arr[], int n, int sum)
{
    if(n==0)
```

```
        return sum==0? 1 : 0;

    return countSubsets(arr, n-1, sum) + countSubsets(arr, n-1,
sum - arr[n-1]);
}


int main() {

    int n = 3, arr[]= {10, 20, 15}, sum = 25;

    cout<<countSubsets(arr, n, sum);

    return 0;
}
```

DP SOLUTION

```
#include <iostream>
#include <limits.h>
using namespace std;

int countSubsets(int arr[], int n, int sum)
{
    int dp[n+1][sum+1];

    for(int i=0; i<=n; i++) dp[i][0] = 1;
    for(int j=1; j<=sum; j++) dp[0][j] = 0;


    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=sum; j++)
        {
                if(j < arr[i-1])
                    dp[i][j] = dp[i-1][j];
                else
                    dp[i][j] = dp[i-1][j] + dp[i][j - arr[i-1]];
            }
        }

        return dp[n][sum];
    }


    int main() {

        int n = 3, arr[]= {2, 5, 3}, sum = 5;

        cout<<countSubsets(arr, n, sum);

        return 0;
    }
```

16. MATRIX CHAIN MULTIPLICATION

```
/* A naive recursive implementation that simply
follows the above optimal substructure property */
#include <bits/stdc++.h>
using namespace std;

// Matrix Ai has dimension p[i-1] x p[i]
// for i = 1..n
int MatrixChainOrder(int p[], int i, int j)
{
    if (i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;

    // place parenthesis at different places
    // between first and last matrix, recursively
    // calculate count of multiplications for
    // each parenthesis placement and return the
    // minimum count
    for (k = i; k < j; k++) {
        count = MatrixChainOrder(p, i, k) + MatrixChainOrder(p, k
+ 1, j) + p[i - 1] * p[k] * p[j];

        if (count < min)
            min = count;
    }

    // Return minimum count
    return min;
}

// Driver Code
int main()
{
    int arr[] = {40, 20, 30, 10, 30};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Minimum number of multiplications is "
        << MatrixChainOrder(arr, 1, n - 1);
}
```

DP SOLUTION

```
#include <iostream>
#include <limits.h>
```

```
using namespace std;

int mChain(int p[], int n)
{

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

    for (int gap = 2; gap < n; gap++)
    {
        for (int i=0; i+gap < n; i++)
        {
            int j = i + gap;
            dp[i][j] = INT_MAX;
            for (int k=i+1; k<j; k++)
            {
               dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j] + p[i]*p[k]*p[j]);
            }
        }
    }

    return dp[0][n-1];
}

int main() {

         int n = 4, arr[]= {2, 1, 3, 4};

         cout<<mChain(arr, n);

         return 0;
}
```

17. PALINDROME PARTITIONING

```
#include <iostream>
#include <limits.h>
using namespace std;

    bool isPalindrome(string input, int start, int end)
         {

             while (start < end) {
                 if (input[start] != input[end])
                     return false;
                 start++;
                 end--;
             }
             return true;
         }

    int palPart(string str)
         {
                  int n = str.length();

                  int dp[n][n];

                  for(int i=0; i<n; i++)
                  {
                           dp[i][i] =0;
                  }

                  for(int gap = 1; gap<n; gap++)
                  {
                           for(int i=0; i+gap<n; i++)
                           {
                                    int j = i + gap;
```

```
                if(isPalindrome(str, i, j))
                {
                    dp[i][j] = 0;
                }
                else
                {
                    dp[i][j] = INT_MAX;

                    for(int k=i; k<j; k++)
                    {
                        dp[i][j] = min(dp[i][j], 1 + dp[i][k] + dp[k+1][j]);
                    }
                }
            }
        }

        return dp[0][n-1];
    }

int main() {

    string s = "geek";

    cout<<palPart(s);

    return 0;
}
```

18. ALLOCATE MINIMUM PAGES

```
#include <bits/stdc++.h>
using namespace std;
```

```
int sum(int arr[],int b, int e){
    int s=0;
    for(int i=b;i<=e;i++)
        s+=arr[i];
    return s;
}

int minPages(int arr[],int n, int k){
    int dp[k+1][n+1];
    for(int i=1;i<=n;i++)
        dp[1][i]=sum(arr,0,i-1);
    for(int i=1;i<=k;i++)
        dp[i][1]=arr[0];

    for(int i=2;i<=k;i++){
        for(int j=2;j<=n;j++){
            int res=INT_MAX;
            for(int p=1;p<j;p++){
                res=min(res,max(dp[i-1][p],sum(arr,p,j-1)));
            }
            dp[i][j]=res;
        }
    }
    return dp[k][n];
}

int main()
{
    int arr[]={10,20,10,30};
    int n=sizeof(arr)/sizeof(arr[0]);
    int k=2;

    cout<<minPages(arr,n,k);
}
```