**0 1 KNAPSACK VARIATIONS**

**01.) 0 1 KNAPSACK**

**RECURSION**

int func(vector<int> &values, vector<int> &weights, int n, int w){

    if(n==0 || w==0)

    return 0;

    if(weights[n-1]<=w){

        return max(values[n-1]+

func(values, weights, n-1, w-weights[n-1]),

func(values, weights, n-1, w));

    }

    return func(values, weights, n-1, w);

}

int maxProfit(vector<int> &values, vector<int> &weights, int n, int w)

{

    return func(values, weights, n, w);

}

**MEMOIZATION**

int func(vector<int> &values, vector<int> &weights, int n, int w,

vector<vector<int>> &dp){

    if(n==0 || w==0)

    return 0;

    if(dp[n][w]!=-1)

    return dp[n][w];

    if(weights[n-1]<=w){

        return dp[n][w]=max(values[n-1]+

func(values, weights, n-1, w-weights[n-1], dp),

func(values, weights, n-1, w, dp));

    }

    return dp[n][w]=func(values, weights, n-1, w, dp);

}

int maxProfit(vector<int> &values, vector<int> &weights, int n, int w)

{

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

    return func(values, weights, n, w, dp);

}

**TABULATION**

int maxProfit(vector<int> &values, vector<int> &weights, int n, int w)

{

    vector<vector<int>> dp(n+1, vector<int> (w+1, 0));

    for(int i=1;i<n+1;i++){

        for(int j=1;j<w+1;j++){

            if(weights[i-1]<=j){

                dp[i][j]=max(values[i-1]+dp[i-1][j-weights[i-1]],

dp[i-1][j]);

            }

            else

            dp[i][j]=dp[i-1][j];

        }

    }

    return dp[n][w];

}

**2.) SUBSET SUM PROBLEM**

**RECURSIVE**

class Solution{

public:

bool func(vector<int> &arr, int n, int sum){

if(n==0 && sum==0)

return true;

if(sum==0)

return true;

if(n==0)

return false;

if(arr[n-1]<=sum){

return func(arr, n-1, sum-arr[n-1]) || func(arr, n-1, sum);

}

return func(arr, n-1, sum);

}

bool isSubsetSum(vector<int>arr, int sum){

int n=arr.size();

return func(arr, n, sum);

}

};

**MEMOIZATION**

class Solution{

public:

    bool func(vector<int> &arr, int n, int sum,

vector<vector<int>> &dp){

        if(n==0 && sum==0)

        return true;

        if(sum==0)

        return true;

        if(n==0)

        return false;

        if(dp[n][sum]!=-1)

        return dp[n][sum];

        if(arr[n-1]<=sum){

            return dp[n][sum]=func(arr, n-1, sum-arr[n-1], dp) ||

func(arr, n-1, sum, dp);

        }

        return dp[n][sum]=func(arr, n-1, sum, dp);

    }

    bool isSubsetSum(vector<int>arr, int sum){

        int n=arr.size();

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

        return func(arr, n, sum, dp);

    }

};

**TABULATION**

class Solution{

public:

bool isSubsetSum(vector<int>arr, int sum){

int n=arr.size();

vector<vector<int>> dp(n+1, vector<int> (sum+1, 0));

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

dp[i][0]=1;

}

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

for(int j=1;j<sum+1;j++){

if(arr[i-1]<=j)

dp[i][j]=dp[i-1][j-arr[i-1]] || dp[i-1][j];

else

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

}

}

return dp[n][sum];

}

};

**03.) PARTITION EQUAL SUBSET SUM**

**MEMOIZATION**

class Solution {

public:

    bool func(vector<int> &arr, int n, int sum,

              vector<vector<int>> &dp){

        if(n==0 && sum==0)

        return true;

        if(sum==0)

        return true;

        if(n==0)

        return false;

        if(dp[n][sum]!=-1)

        return dp[n][sum];

        if(arr[n-1]<=sum){

            return dp[n][sum]=func(arr, n-1, sum-arr[n-1], dp) ||

                                    func(arr, n-1, sum, dp);

        }

        return dp[n][sum]=func(arr, n-1, sum, dp);

    }

    bool isSubsetSum(vector<int>&arr, int sum){

        int n=arr.size();

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

        return func(arr, n, sum, dp);

    }

    bool canPartition(vector<int>& nums) {

        int n=nums.size();

        int sum=accumulate(nums.begin(), nums.end(), 0);

        if(sum&1)

        return false;

        int req\_sum=sum/2;

        return isSubsetSum(nums, req\_sum);

    }

};

**TABULATION**

class Solution {

public:

    bool isSubsetSum(vector<int>arr, int sum){

        int n=arr.size();

        vector<vector<int>> dp(n+1, vector<int> (sum+1, 0));

        for(int i=0;i<n+1;i++){

            dp[i][0]=1;

        }

        for(int i=1;i<n+1;i++){

            for(int j=1;j<sum+1;j++){

                if(arr[i-1]<=j)

                dp[i][j]=dp[i-1][j-arr[i-1]] || dp[i-1][j];

                else

                dp[i][j]=dp[i-1][j];

            }

        }

        return dp[n][sum];

    }

    bool canPartition(vector<int>& nums) {

        int n=nums.size();

        int sum=accumulate(nums.begin(), nums.end(), 0);

        if(sum&1)

        return false;

        int req\_sum=sum/2;

        return isSubsetSum(nums, req\_sum);

    }

};

**04.) COUNT OF A SUBSET WITH A GIVEN SUM**

class Solution {

public:

    int mod = 1e9 + 7;

    int func(int arr[], int n, int sum) {

        vector<vector<int>> dp(n + 1, vector<int>(sum + 1, 0));

        for (int i = 0; i <= n; i++) {

            dp[i][0] = 1;

        }

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

            for (int j = 0; j <= sum; j++) {

                if (arr[i - 1] <= j) {

                    dp[i][j] = (dp[i - 1][j - arr[i - 1]] + dp[i - 1][j])

% mod;

                } else {

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

                }

            }

        }

        return dp[n][sum];

    }

    int perfectSum(int arr[], int n, int sum) {

        return func(arr, n, sum);

    }

};

**05.) TARGET SUM**

class Solution {

public:

    int func(vector<int> &arr, int sum) {

        int n=arr.size();

        vector<vector<int>> dp(n + 1, vector<int>(sum + 1, 0));

        for (int i = 0; i <= n; i++) {

            dp[i][0] = 1;

        }

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

            for (int j = 0; j <= sum; j++) {

                if (arr[i - 1] <= j) {

                    dp[i][j] = (dp[i - 1][j - arr[i - 1]] + dp[i - 1][j]);

                } else {

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

                }

            }

        }

        return dp[n][sum];

    }

    int findTargetSumWays(vector<int>& nums, int target) {

        int sum=accumulate(nums.begin(), nums.end(), 0);

        if(target>sum)

        return 0;

        if((target+sum)%2!=0)

        return 0;

        sum=(sum-target)/2;

        return func(nums, sum);

    }

};

**UNBOUNDED KNAPSACK**

**1.) UNBOUNDED KNAPSACK**

**RECURSION**

int func(vector<int> &values, vector<int> &weights, int n, int w){

    if(n==0 || w==0)

    return 0;

    if(weights[n-1]<=w){

        return max(values[n-1]+

                   func(values, weights, n, w-weights[n-1]),

                   func(values, weights, n-1, w));

    }

    return func(values, weights, n-1, w);

}

int unboundedKnapsack(int n, int w, vector<int> &profit, vector<int> &weight){

    return func(profit, weight, n, w);

}

**MEMOIZATION**

int func(vector<int> &values, vector<int> &weights, int n, int w,

         vector<vector<int>> &dp){

    if(n==0 || w==0)

    return 0;

    if(dp[n][w]!=-1)

    return dp[n][w];

    if(weights[n-1]<=w){

        return dp[n][w]=max(values[n-1]+

                    func(values, weights, n, w-weights[n-1], dp),

                    func(values, weights, n-1, w, dp));

    }

    return dp[n][w]=func(values, weights, n-1, w, dp);

}

int unboundedKnapsack(int n, int w, vector<int> &values, vector<int> &weights){

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

    return func(values, weights, n, w, dp);

}

**TABULATION**

int unboundedKnapsack(int n, int w, vector<int> &values, vector<int> &weights){

    vector<vector<int>> dp(n+1, vector<int> (w+1, 0));

    for(int i=1;i<n+1;i++){

        for(int j=1;j<w+1;j++){

            if(weights[i-1]<=j){

                dp[i][j]=max(values[i-1]+dp[i][j-weights[i-1]],

                             dp[i-1][j]);

            }

            else

            dp[i][j]=dp[i-1][j];

        }

    }

    return dp[n][w];

}

**2.) ROD CUTTING PROBLEM**

int cutRod(vector<int> &price, int n)

{

    vector<vector<int>> dp(n+1, vector<int>(n+1, 0));

    for(int i = 1; i <= n; i++){

        for(int j = 1; j <= n; j++){

            int notTake = dp[i-1][j];

            int take = 0;

            if(j >= i){

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

            }

            dp[i][j] = max(take, notTake);

        }

    }

    return dp[n][n];

}

**3.) COIN CHANGE II**

class Solution {

public:

    int change(int sum, vector<int>& coins) {

        int n=coins.size();

        vector<vector<int>> dp(n+1, vector<int> (sum+1, 0));

        for(int i=0;i<=n;i++){

            dp[i][0]=1;

        }

        for(int i=1;i<=n;i++){

            for(int j=1;j<=sum;j++){

                if(coins[i-1]<=j){

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

                }

                else{

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

                }

            }

        }

        return dp[n][sum];

    }

};

**4.) COIN CHANGE I**

class Solution {

public:

    int coinChange(vector<int>& coins, int sum) {

        int n=coins.size();

        vector<vector<int>> dp(n+1, vector<int> (sum+1, INT\_MAX-1));

        for(int i=0;i<=n;i++){

            if(i!=0)

            dp[i][0]=0;

        }

        for(int i=1;i<=n;i++){

            for(int j=1;j<=sum;j++){

                if(coins[i-1]<=j){

                    dp[i][j]=min(1+dp[i][j-coins[i-1]], dp[i-1][j]);

                }

                else{

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

                }

            }

        }

        if(dp[n][sum]==INT\_MAX-1)

        return -1;

        return dp[n][sum];

    }

};

**LONGEST COMMON SUBSEQUENCE**

**1.) LONGEST COMMON SUBSEQUENCE**

**RECURSIVE**

class Solution {

public:

    int LCS(string a, string b, int n, int m){

        if(n==0 || m==0)

        return 0;

        if(a[n-1]==b[m-1])

        return 1+LCS(a, b, n-1, m-1);

        else

        return max(LCS(a, b, n, m-1), LCS(a, b, n-1, m));

    }

    int longestCommonSubsequence(string text1, string text2) {

        int n=text1.size();

        int m=text2.size();

        return LCS(text1, text2, n, m);

    }

};

**MEMOIZATION (TLE)**

class Solution {

public:

    int LCS(string a, string b, int n, int m, vector<vector<int>> &dp){

        if(n==0 || m==0)

        return 0;

        if(dp[n][m]!=-1)

        return dp[n][m];

        if(a[n-1]==b[m-1])

        return dp[n][m]=1+LCS(a, b, n-1, m-1, dp);

        else

        return dp[n][m]=max(LCS(a, b, n, m-1, dp), LCS(a, b, n-1, m, dp));

    }

    int longestCommonSubsequence(string text1, string text2) {

        int n=text1.size();

        int m=text2.size();

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

        return LCS(text1, text2, n, m, dp);

    }

};

**MEMOIZATION (GOOD)**

class Solution {

    vector<vector<int>> dp;

    int lcs(int i, int j, string& text1, string& text2) {

        if (i >= text1.size() || j >= text2.size()) {

            return 0;

        }

        if (dp[i][j] != -1) {

            return dp[i][j];

        }

        if (text1[i] == text2[j]) {

            return dp[i][j] = 1 + lcs(i + 1, j + 1, text1, text2);

        }

        return dp[i][j] = max(lcs(i, j + 1, text1, text2),

lcs(i + 1, j, text1, text2));

    }

public:

    int longestCommonSubsequence(string text1, string text2) {

        dp.resize(text1.size(), vector<int>(text2.size(), -1));

        return lcs(0, 0, text1, text2);

    }

};

**TABULATION**

class Solution {

public:

    int longestCommonSubsequence(string text1, string text2) {

        int n=text1.size();

        int m=text2.size();

        vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

        for(int i=1;i<n+1;i++){

            for(int j=1;j<m+1;j++){

                if(text1[i-1]==text2[j-1])

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

                else

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

            }

        }

        return dp[n][m];

    }

};

**2.) LONGEST COMMON SUBSTRING**

int LCSubStr(string &text1, string &text2){

    int n=text1.size();

    int m=text2.size();

    vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

    int ans=0;

    for(int i=1;i<n+1;i++){

        for(int j=1;j<m+1;j++){

            if(text1[i-1]==text2[j-1]){

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

                dp[i][j]=val;

                ans=max(ans, val);

            }

            else{

                dp[i][j]=0;

            }

        }

    }

    return ans;

}

**3.) SHORTEST COMMON SUPERSEQUENCE (ONLY LENGTH)**

class Solution

{

public:

int shortestCommonSupersequence(string text1, string text2, int n, int m)

{

vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

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

for(int j=1;j<m+1;j++){

if(text1[i-1]==text2[j-1])

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

else

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

}

}

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

}

};

**4.) MINIMUM NUMBER OF DELETIONS AND INSERTIONS TO MAKE STRING A TO B**

int canYouMake(string &text1, string &text2){

    int n=text1.length();

    int m=text2.length();

    vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

    for(int i=1;i<n+1;i++){

        for(int j=1;j<m+1;j++){

            if(text1[i-1]==text2[j-1])

            dp[i][j]=1+dp[i-1][j-1];

            else

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

        }

    }

    return m+n-(2\*dp[n][m]);

}

**5.) DELETE OPERATIONS FOR TWO STRINGS**

**SAME AS ABOVE**

class Solution {

public:

    int minDistance(string text1, string text2) {

        int n=text1.length();

        int m=text2.length();

        vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

        for(int i=1;i<n+1;i++){

            for(int j=1;j<m+1;j++){

                if(text1[i-1]==text2[j-1])

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

                else

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

            }

        }

        return m+n-(2\*dp[n][m]);

    }

};

**6.) LONGEST PALINDROMIC SUBSEQUENCE**

class Solution {

public:

    int longestPalindromeSubseq(string text1) {

        string text2=text1;

        reverse(text2.begin(), text2.end());

        int n=text1.length();

        int m=text2.length();

        vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

        for(int i=1;i<n+1;i++){

            for(int j=1;j<m+1;j++){

                if(text1[i-1]==text2[j-1])

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

                else

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

            }

        }

        return dp[n][m];

    }

};

**7.) MINIMUM INSERTION STEPS TO MAKE STRING PALINDROME**

class Solution {

public:

    int minInsertions(string text1) {

        string text2=text1;

        reverse(text2.begin(), text2.end());

        int n=text1.length();

        int m=text2.length();

        vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

        for(int i=1;i<n+1;i++){

            for(int j=1;j<m+1;j++){

                if(text1[i-1]==text2[j-1])

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

                else

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

            }

        }

        return n-dp[n][m];

    }

};

**8.) LONGEST REPEATING SUBSEQUENCE**

#include <bits/stdc++.h>

int longestRepeatingSubsequence(string text1, int n)

{

    string text2=text1;

    int m=text2.length();

    vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

    for(int i=1;i<n+1;i++){

        for(int j=1;j<m+1;j++){

            if(text1[i-1]==text2[j-1] && i!=j)

            dp[i][j]=1+dp[i-1][j-1];

            else

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

        }

    }

    return dp[n][m];

}

**9.) IS SUBSEQUENCE**

class Solution {

public:

    bool isSubsequence(string text1, string text2) {

        int n=text1.size();

        int m=text2.size();

        vector<vector<int>> dp(n+1, vector<int> (m+1, 0));

        for(int i=1;i<n+1;i++){

            for(int j=1;j<m+1;j++){

                if(text1[i-1]==text2[j-1])

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

                else

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

            }

        }

        return n==dp[n][m];

    }

};

**10.) NUMBER OF MATCHING SUBSEQUENCES**

class Solution {

public:

    int LCS(string &word, string &s){

        int index=-1;

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

            index=s.find(word[i], index+1);

            if(index==-1){

                return false;

            }

        }

        return true;

    }

    int numMatchingSubseq(string s, vector<string>& words) {

        int count=0;

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

            string s2=words[i];

            if(LCS(s2, s))

            count++;

        }

        return count;

    }

};

**MATRIX CHAIN MULTIPLICATION**

**01.) BASIC MCM**

**RECURSIVE**

#include <bits/stdc++.h>

int mcm(vector<int> &arr, int i, int j){

    if(i>=j)

    return 0;

    int mini=INT\_MAX;

    for(int k=i;k<=j-1;k++){

        int temp\_ans=mcm(arr, i, k)+mcm(arr, k+1, j)+

(arr[i-1]\*arr[k]\*arr[j]);

        if(temp\_ans<mini){

            mini=temp\_ans;

        }

    }

    return mini;

}

int matrixMultiplication(vector<int> &arr, int N){

    return mcm(arr, 1, N-1);

}

**MEMOIZATION**

#include <bits/stdc++.h>

int mcm(vector<int> &arr, int i, int j, vector<vector<int>> &dp){

    if(i==j)

    return 0;

    if(dp[i][j]!=-1)

    return dp[i][j];

    int mini=INT\_MAX;

    for(int k=i;k<=j-1;k++){

        int temp\_ans=mcm(arr, i, k, dp)+mcm(arr, k+1, j, dp)+

(arr[i-1]\*arr[k]\*arr[j]);

        if(temp\_ans<mini)

        mini=temp\_ans;

    }

    return dp[i][j]=mini;

}

int matrixMultiplication(vector<int> &arr, int N){

    vector<vector<int>> dp(N+1, vector<int> (N+1, -1));

    return mcm(arr, 1, N-1, dp);

}

**02.) PALINDROME PARTITIONING II**

class Solution {

public:

    bool isPalindrome(string &s, int i, int j){

        while(i<=j){

            if(s[i]==s[j]){

                i++;

                j--;

            }

            else

            return false;

        }

        return true;

    }

    int mcm(string &s, int i, int j, vector<int> &dp){

        if(i>=j)

        return 0;

        if(isPalindrome(s, i, j))

        return 0;

        if(dp[i]!=-1)

        return dp[i];

        int mini=INT\_MAX;

        for(int k=i;k<=j-1;k++){

            if(isPalindrome(s, i, k)){

                int temp=mcm(s, k+1, j, dp)+1;

                mini=min(mini, temp);

            }

        }

        return dp[i]=mini;

    }

    int minCut(string s) {

        int n=s.length();

        vector<int> dp(n+1, -1);

        return mcm(s, 0, n-1, dp);

    }

};

**03.) BOOLEAN PARENTHESIZATION**

**EVALUATE EXPRESSIONS TO TRUE**

const int mod = 1000000007;

int f(int i, int j, int isTrue, string& s, vector<vector<vector<long long>>>& dp) {

    if(i>j) return 0;

    if(i==j) {

        if(isTrue) return s[i]=='T';

        else return s[i] == 'F';

    }

    if(dp[i][j][isTrue] != -1)

    return dp[i][j][isTrue];

    long long ways = 0;

    for(int k=i; k<j; k++) {

        long long leftTrue = f(i, k-1, 1, s, dp);

        long long leftFalse = f(i, k-1, 0, s, dp);

        long long rightTrue = f(k+1, j, 1, s, dp);

        long long rightFalse = f(k+1, j, 0, s, dp);

        if(s[k] == '&') {

            if(isTrue)

            ways = (ways + (leftTrue\*rightTrue)%mod)%mod;

            else

            ways = (ways + (leftFalse\*rightTrue)%mod + (leftFalse\*rightFalse)%mod + (leftTrue\*rightFalse)%mod)%mod;

        }

        else if(s[k] == '|') {

            if(isTrue)

            ways = (ways + (leftTrue\*rightTrue)%mod + (leftTrue\*rightFalse)%mod + (leftFalse\*rightTrue)%mod)%mod;

            else

            ways = (ways + (leftFalse\*rightFalse)%mod)%mod;

        }

        else {

            if(isTrue)

            ways = (ways + (leftTrue\*rightFalse)%mod + (leftFalse\*rightTrue)%mod)%mod;

            else

            ways = (ways + (leftTrue\*rightTrue)%mod + (leftFalse\*rightFalse)%mod)%mod;

        }

    }

    return dp[i][j][isTrue] = ways;

}

int evaluateExp(string & exp) {

    int n = exp.size();

    vector<vector<vector<long long>>> dp(n, vector<vector<long long>>(n, vector<long long>(2, -1)));

    return f(0,n-1, 1, exp, dp);

}

**THANK YOU !**