```cpp
//Fibonacci Number
//recursion+dp
int f(int n, vector<int> &dp){
    if(n<=1) return n;
    if(dp[n]!=-1) return dp[n];
    return dp[n]=f(n-1,dp)+f(n-2,dp);
}
//space-Optimization
void f(int n){
    int prev2 = 0;
    int prev1 = 1;
    for(int i=2;i<=n;i++){
        int curi = prev1+prev2;
        prev2 = prev1;
        prev1 = curi;
    }
    cout<<prev1;
}


//Frog Jump
//Recursion
int f(int ind, vector<int>&heights){
    if(ind == 0) return 0;

    int left= f(ind-1, heights) + abs(heights[ind] - heights[ind-1]);

    int right=INT_MAX;
    if(ind>1)
        right= f(ind-2, heights) + abs(heights[ind] -
heights[ind-2]);

    return min(left, right);
}
//DP
int f(int ind, vector<int>&heights, vector<int>&dp){
```

```
    if(ind == 0) return 0;
    if(dp[ind]!=-1) return dp[ind];
    int left= f(ind-1, heights, dp) + abs(heights[ind] -
heights[ind-1]);

    int right=INT_MAX;
    if(ind>1)
        right= f(ind-2, heights, dp) + abs(heights[ind] -
heights[ind-2]);

    return dp[ind] = min(left, right);
}
//memoization - topDown
//tabulation = bottomUp

int f(int n, vector<int>&heights){
    vector<int> dp(n, 0);
    dp[0] = 0;

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

        int ss = INT_MAX;
        if(i>1) ss = dp[i-2]+abs(heights[i] - heights[i-2]);

        dp[i] = min(fs, ss);
    }
    return dp[n-1];
}
//space-Optimization
int f(int n, vector<int>&heights){
    int prev1 =0, prev2 = 0;

    for(int i=1;i<n;i++){
        int fs = prev1+abs(heights[i] - heights[i-1]);

        int ss = INT_MAX;
        if(i>1) ss =  prev2+abs(heights[i] - heights[i-2]);
```

```
        int curi= min(fs, ss);

        prev2 = prev1;
        prev1 = curi;

    }
    return prev1;
}


//frog with k Jumps - followUpQuestion

//********************* WRITE CODE ***********************//



//Maximum Sum of Non Adjacent elements

int f(int ind, vector<int>&nums){
    if(ind == 0) return nums[ind];
    if(ind <0) return 0;

    int pick = nums[ind] + f(ind-2, nums);
    int nonPick = f(ind-1, nums);

    return max(pick, nonPick);
}
//Dp
int f(int ind, vector<int>&nums, vector<int>&dp){
    if(ind == 0) return nums[ind];
    if(ind <0) return 0;
    if(dp[ind]!=-1) return dp[ind];

    int pick = nums[ind] + f(ind-2, nums);
    int nonPick = f(ind-1, nums);

    return dp[ind] = max(pick, nonPick);
```

```
}

int f(int ind, vector<int>&nums, vector<int>&dp){
    vector<int> dp(n, 0);
    dp[0] = 0;

    for(int i=1;i<n;i++){
        int pick = INT_MIN;
        if(i>1) pick = nums[ind] + dp[i-2];

        int nonPick = dp[i-1];

        dp[i] = max(pick, nonPick);
    }
    return dp[n-1];
}

int f(int ind, vector<int>&nums, vector<int>&dp){
    int prev1=0, prev2=0;

    for(int i=1;i<n;i++){
        int pick = nums[ind];
        if(i>1) pick += prev2;

        int nonPick = 0+prev1;

        int curi = max(pick, nonPick);

        prev1=prev2;
        prev1 = curi;
    }
    return prev1;
}

//HOUSE ROBBER1 = Maximum Sum of Non Adjacent elements
//HOUSE ROBBER2 = max(leaveindex0, leaveindexLast) Maximum Sum of Non
Adjacent elements
//HOUSE ROBBER3 = Binary tree- used Map for Dp
```

```cpp
//Ninja's training
//reecursion
int f(int day, int last, vector<vector<int>> &points){
    if(day == 0){
        int maxi =0;
        for(int task= 0; task<3; task++){
            if(task!= last){
                maxi = max(maxi, points[0][task]);
            }
        }
        return maxi;
    }

    int maxi = 0;

    for(int task=0;task<3;task++){
        if(task!=last){
            int point = points[day][task] + f(day-1, task, points);
            maxi = max(maxi, point);
        }
    }
    return maxi;
}

int ninjaTraining(int n, vector<vector<int>>&points){
    return f(n-1,3,points);
}

//dp
int f(int day, int last, vector<vector<int>> &points,
vector<vector<int>> &dp){
    if(day == 0){
        int maxi =0;
        for(int task= 0; task<3; task++){
            if(task!= last){
                maxi = max(maxi, points[0][task]);
```

```
            }
        }
        return maxi;
    }

     if(dp[day][last]!=-1) return dp[day][last];

    int maxi = 0;

    for(int task=0;task<3;task++){
        if(task!=last){
            int point = points[day][task] + f(day-1, task, points,
dp);
            maxi = max(maxi, point);
        }
    }
    return dp[day][last] = maxi;
}

//tabulation




int ninjaTraining(int n, vector<vector<int>>&points){
    vector<vector<int>> dp(n, vector<int>(4,0));

    dp[0][0] = max(points[0][1], points[0][2]);
    dp[0][1] = max(points[0][0], points[0][2]);
    dp[0][2] = max(points[0][0], points[0][1]);
    dp[0][3] = max(points[0][1], max(points[0][1], points[0][2]));

    for(int day=1;day<n;day++){
        for(int last=0;last<4;last++){
            dp[day][last] = 0;

            int maxi = 0;
```

```
            for(int task=0;task<3;task++){
                if(task!=last){
                    int point = points[day][task] +
points[day-1][task];
                    dp[day][last] = max(dp)(day)
                }
            }
        }
    }

}
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^STUCK
//Ninja's training




// Grid Unique Paths
// recursion
int f(int i,int j){
    if(i==0 and j==0) return 1;
    if(i<0 || j<0) return 0;
    int up =  f(i-1, j);
    int left = f(i, j-1);
    return up+left;
}
//dp
int f(int i,int j, vector<vector<int>> dp){
    if(i==0 and j==0) return 1;
    if(i<0 || j<0) return 0;
    if(dp[i][j]!=-1) return dp[i][j];
    int up =  f(i-1, j);
    int left = f(i, j-1);
    return dp[i][j] = up+left;
}
//tabulation
int uniquePaths(int m, int n){
    int dp[m][n];
```

```
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            if(i==0  || j == 0)
                dp[i][j] = 1;
            else{
                int up=0, left =0;
                if(i>0) up=dp[i-1][j];
                if(j>0) left=dp[i][j-1];

                dp[i][j] = up +left;
            }
        }
    }
    return dp[m-1][n-1];
}
//spaceOptimization
int uniquePaths(int m, int n){
    //both prev, cur are horizontal
    vector<int> prev(n,0);
    for(int i=0; i<m; i++){
        vector<int> cur(n,0);
        for(int j=0; j<n; j++){
            if(i==0  || j == 0)
                cur[j] = 1;
            else{
                int up=0, left =0;
                if(i>0) up= prev[j];
                if(j>0) left=  cur[j-1];

                cur[j] = up +left;
            }
        }
        prev = cur;
    }
    return prev[n-1];
}
//trick-google
int uniquePaths(int m, int n){
```

```
    int N = n-m-2;
    int r = m-1;
    double res = 1;

    for(int i=1; i<=r; i++)
        res = res * (N-r+i)/i ;

    return (int)res;
}




//unique Paths - 2
// recursion- UniquePaths1:
    // if(i<0 || j<0 || mat[i][j] == -1) return 0;
//DP
//tabuation- UniquePaths1:
    // if(mat[i][j] == -1) dp[i][j] = 0;
//sPaceOptimization- UniquePaths1:
    // if(mat[i][j] == -1) cur[j] = 0;




//Minimum Path Sum
//recursion
int f(int i, int j, vector<vector<int>>&grid){
    if(i==0 && j==0) return grid[i][j];
    if(i<0 || j<0) return 1e9;

    int up= grid[i][j] + f(i-1, j, grid);
    int left = grid[i][j] + f(i, j-1, grid);

    return min(up, left);
}
int minSumPath(vector<vector<int>>&grid){
```

```
    int n=grid.size(), m=grid[0].size();
    return f(n-1, m-1, grid);
}
//DP
int f(int i, int j, vector<vector<int>>&grid,
vector<vector<int>>&dp){
    if(i==0 && j==0) return grid[i][j];
    if(i<0 || j<0) return 1e9;
    if(dp[i][j]!= -1) return dp[i][j];

    int up= grid[i][j] + f(i-1, j, grid, dp);
    int left = grid[i][j] + f(i, j-1, grid, dp);

    return dp[i][j] = min(up, left);
}
//Tabulation
int minSumPath(vector<vector<int>>&grid){
    int n=grid.size(), m=grid[0].size();
    vector<vector<int>> dp(n, vector<int>(m,0));

    for(int i=0 ; i<n; i++){
        for(int j=0; j<m ; j++){
            if(i==0 and j==0) dp[i][j] == grid[i][j];
            else{
                int up= grid[i][j];
                if(i>0) up+=dp[i-1][j];
                else up+=1e9;

                int left = grid[i][j];
                if(j>0) left+=dp[i][j-1];
                else left+=1e9;

                dp[i][j] = min(left, up);
            }
        }
    }
    return dp[n-1][m-1];
}
```

```cpp
//spaceOptimization
int minSumPath(vector<vector<int>>&grid){
    int n=grid.size(), m=grid[0].size();
    vector<int> prev(n,1e9);

    for(int i=0 ; i<n; i++){
        vector<int> cur(n,1e9);
        for(int j=0; j<m ; j++){
            if(i==0 and j==0) cur[j] == grid[i][j];
            else{
                int up= grid[i][j];
                if(i>0) up+=prev[j];
                else up+=1e9;

                int left = grid[i][j];
                if(j>0) left+=cur[j-1];
                else left+=1e9;

                cur[j] = min(left, up);
            }
        }
        prev = cur ;
    }
    return prev[m-1];
}




// Triangle- Fixed Staring Point and Variable Ending Point
//BruteForce
int f(int i, int j, vector<vector<int>>& triangle, int n){
    if(i==n-1) return triangle[i][j];

    int d = triangle[i][j] + f(i+1, j, n);
    int dg = triangle[i][j] + f(i+1, j+1, n);

    return min(d, dg);
```

```
}
//DP
int f(int i, int j, vector<vector<int>>& triangle, int n,
vector<vector<int>>&dp){
    if(i==n-1) return triangle[i][j];
    if(dp[i][j]!=-1) return dp[i][j];
    int d = triangle[i][j] + f(i+1, j, n, dp);
    int dg = triangle[i][j] + f(i+1, j+1, n, dp);

    return dp[i][j]= min(d, dg);
}
//Tabulation
int minimumPathSumTriangle(vector<vector<int>>& triangle, int n){
    vector<vector<int>> dp(n, vector<int>(n,0));

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

    for(int i=n-2; i>=0;i--){
        for(int j=i; j>=0; j--){
            int d = triangle[i][j] + dp[i+1][j];
            int dg = triangle[i][j] + dp[i+1][j+1];

            dp[i][j] = min(d, dg);
        }
    }
    return dp[0][0];
}
//spaceOptimization
int minimumPathSumTriangle(vector<vector<int>>& triangle, int n){
    vector<int> front(n, 0);

    for(int j=0;j<n;j++) front[j] = triangle[n-1][j];

    for(int i=n-2; i>=0;i--){
        vector<int> cur(n,0);
        for(int j=i; j>=0; j--){
            int d = triangle[i][j] + front[j];
            int dg = triangle[i][j] + front[j+1];
```

```
            cur[j] = min(d, dg);
        }
        front = cur;
    }
    return front[0];
}


//Minimum Maximum Falling Path Sum
//recursion
int f(int i, int j, vector<vector<int >> & matrix){
    if(j < 0 || j>=matrix[0].size()) return -1e8;
    if(i == 0) return matrix[0][j];

    int u = matrix[i][j] + f(i-1, j, matrix);
    int ld = matrix[i][j] + f(i-1, j-1, matrix);
    int rd = matrix[i][j] + f(i-1, j+1, matrix);

    return max({u, ld, rd});
}
int getMaxPathSum(vector<vector<int >> &matrix){
    int n = matrix.size(), m = matrix[0].size();
    int maxi = -1e8;
    for(int j=0; j<m;j++)
        maxi = max(maxi, f(n-1, j, matrix));
    return maxi;
}

//dp
int f(int i, int j, vector<vector<int >> & matrix, vector<vector<int
>> & dp){
    if(j < 0 || j>=matrix[0].size()) return -1e8;
    if(i == 0) return matrix[0][j];
    if(dp[i][j]!=-1) return dp[i][j];

    int u = matrix[i][j] + f(i-1, j, matrix, dp);
    int ld = matrix[i][j] + f(i-1, j-1, matrix, dp);
```

```
    int rd = matrix[i][j] + f(i-1, j+1, matrix, dp);

    return dp[i][j] = max({u, ld, rd});
}

//tabulation
int getMaxPathSum(vector<vector<int >> &matrix){
    int n = matrix.size(), m = matrix[0].size();
    vector<vector<int >> dp(n, vector<int>(m,0));

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

    for(int i=1; i<n; i++){
        for(int j=0; j<m; j++){
            int u  = matrix[i][j] + dp[i-1][j];
            int ld = matrix[i][j] + (j>0)?dp[i-1][j-1] : -1e8 ;
            int rd = matrix[i][j] + (j<m-1)?dp[i-1][j+1]: -1e8;

            dp[i][j] = max({u, ld, rd});
        }
    }

    int maxi = -1e8;
    for(int j=0; j<m;j++)
        maxi = max(maxi, dp[n-1][j]);
    return maxi;
}
//spaceOptimization
int getMaxPathSum(vector<vector<int >> &matrix){
    int n = matrix.size(), m = matrix[0].size();
    vector<int > prev(n, 0);

    for(int j=0; j<m; j++) prev[j] = matrix[0][j];

    for(int i=1; i<n; i++){
        vector<int > cur(n, 0);
        for(int j=0; j<m; j++){
            int u  = matrix[i][j] + prev[j];
```

```
            int ld = matrix[i][j] + (j>0)?prev[j-1] : -1e8 ;
            int rd = matrix[i][j] + (j<m-1)?prev[j+1]: -1e8;

            cur[j] = max({u, ld, rd});
        }
        prev = cur
    }

    int maxi = -1e8;
    for(int j=0; j<m;j++)
        maxi = max(maxi, prev[j]);
    return maxi;
}



//cherry PickUp
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^STUCK





//Subset Sum equal to target
//recursion
bool f(int ind, int target, vector<int>&arr){
    if(target==0) return 1;
    if(ind==0 && arr[0]==target) return 1;

    bool notTake = f(ind-1, target, arr);

    bool take = 0;
    if(arr[ind] <= target)
        take = f(ind-1, target-arr[ind], arr);

    return take | notTake;
}
//DP
// vector<vector<int >> dp(n, vector<int>(target+1,-1));
```

```cpp
bool f(int ind, int target, vector<int>&arr, vector<vector<int >>
dp){
    if(target==0) return 1;
    if(ind==0 && arr[0]==target) return 1;

    if(dp[ind][target]!=-1) return dp[ind][target];

    bool notTake = f(ind-1, target, arr, dp);

    bool take = 0;
    if(arr[ind] <= target)
        take = f(ind-1, target-arr[ind], arr, dp);

    return dp[ind][target] = take | notTake;
}

//tabulation k==target
bool subsetSumToK(int n, int k, vector<int>arr){
    vector<vector<int >> dp(n, vector<int>(k+1,0));
    for(int i=0; i<n; i++) dp[i][0] = 1;
    dp[0][arr[0]]  =1;
    for(int ind = 1; ind<n; ind++){
        for(int target=1; target<=k; target++){
            bool notTake = dp[ind-1][target];
            bool take = false;

            if(arr[ind] <= target) take = dp[ind-1][target -
arr[ind]];
            dp[ind][target] = take | notTake;
        }
    }
    return dp[n-1][k];
}
//spaceOptimization
bool subsetSumToK(int n, int k, vector<int>arr){
    vector<int>prev(k+1, 0);

    prev[0] = 1;
```

```cpp
    if(arr[0]<=k) prev[arr[0]] = 1;

    for(int ind = 1; ind<n; ind++){
        vector<int> cur(k+1, 0);
        for(int target=1; target<=k; target++){
            bool notTake = prev[target];
            bool take = false;

            if(arr[ind] <= target) take = prev[target - arr[ind]];
            cur[target] = take | notTake;
        }
        prev = cur;
    }
    return prev[k];
}


//Partition Equal Subset Sum - similar to subset Sum to K
// k = accumulate(arr.begin(), arr.end()) / 2;


// Partition a set into two subsets with Minimum Absolute Sum
DIfference
//o(2^n)
int minSubsetSumDifference(vector<int>& arr, int n){
    int toSum = 0;
    for(int i=0; i<n; i++) toSum += arr[i];

    int k = toSum;

    vector<vector<int >> dp(n, vector<int>(k+1, 0));
    for(int i=0; i<n; i++) dp[i][0] = 1;
    dp[0][arr[0]]  =1;
    for(int ind = 1; ind<n; ind++){
        for(int target=1; target<=k; target++){
            bool notTake = dp[ind-1][target];
            bool take = false;
```

```
            if(arr[ind] <= target) take = dp[ind-1][target -
arr[ind]];
            dp[ind][target] = take | notTake;
        }
    }
    //dp[n-1][col->0 ... toSum] as can be seen via table
    int mini = 1e9;
    for(int s1 = 0; s1 <= toSum/2; s1++){
        if(dp[n-1][s1])
            mini = min(mini, abs((toSum-s1) - s1));
    }
    return mini;
}



// Count subsets with sum K (arr[i]>0)
//recursion
int f(int ind, int sum, vector<int> &num){
    if(sum == 0) return 1;
    if(ind == 0) return num[0]==sum;

    int notTake = f(ind-1, sum, num);
    int take = 0;
    if(num[ind] <= sum) f(ind-1, sum-num[ind], num);

    return notTake + take;
}
//dp
int f(int ind, int sum, vector<int> &num, vector<vector<int >> &dp){
    if(sum == 0) return 1;
    if(ind == 0) return num[0]==sum;
    if(dp[ind][sum]!=-1) return dp[ind][sum];


    int notTake = f(ind-1, sum, num, dp);
    int take = 0;
    if(num[ind] <= sum) f(ind-1, sum-num[ind], num, dp);
```

```
    return dp[ind][sum] = notTake + take;
}
//tabulation
int findWays(int n, int k, vector<int>arr){
    vector<vector<int >> dp(n, vector<int>(k+1,0));

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

    for(int ind = 1; ind<n; ind++){
        for(int target=0; target<=k; target++){
            int notTake = dp[ind-1][target];
            int take = 0;
            if(arr[ind] <= target) take = dp[ind-1][target -
arr[ind]];

            dp[ind][target] = take + notTake;
        }
    }
    return dp[n-1][k];
}
//spaceOptimization
int findWays(int n, int k, vector<int>arr){
    vector<int> prev(n,0);

    prev[0] = 1;
    prev[arr[0]] = 1;

    for(int ind = 1; ind<n; ind++){
        vector<int> cur(n,0);
        for(int target=0; target<=k; target++){
            int notTake = prev[target];
            int take = 0;
            if(arr[ind] <= target) take = prev[target - arr[ind]];

            cur[target] = take + notTake;
        }
```

```
        prev = cur;
    }
    return prev[k];
}



//Count Partitions with given DIfference
//(totSum-D)/2   -subsSetSum


//0-1 KnapSack
//recursion
int f(int ind, int W, vector<int> &wt, vector<int> &val){
    if(ind == 0){
        if(wt[0]<=W) return val[0];
        return 0;
    }

    int notTake = 0 + f(ind-1, W, wt, val);
    int take = INT_MIN;
    if(wt[ind] <= W){
        take = val[ind] + f(ind -1, W - wt[ind], wt, val);
    }
    return max(take, notTake);
}
//DP
int f(int ind, int W, vector<int> &wt, vector<int> &val,
vector<vector<int>>& dp){
    if(ind == 0){
        if(wt[0]<=W) return val[0];
        return 0;
    }

    if(dp[ind][W]!=-1) return dp[ind][W];

    int notTake = 0 + f(ind-1, W, wt, val);
    int take = INT_MIN;
```

```
    if(wt[ind] <= W){
        take = val[ind] + f(ind -1, W - wt[ind], wt, val);
    }
    return dp[ind][W] = max(take, notTake);
}
//tabulation
int knapsack(vector<int>wt, vector<int>val, int n, int maxWeight){
    vector<vector<int >> dp(n, vector<int>(maxWeight+1,0));

    for(int W= wt[0]; W<= maxWeight; W++)  dp[0][W] = val[0];

    for(int ind = 1; ind<n; ind++){
        for(int W=0; W<=maxWeight; W++){
            int notTake = dp[ind-1][W];
            int take = 0;
            if(wt[ind] <= W) take = dp[ind-1][ W - wt[ind]];

            dp[ind][W] = take + notTake;
        }
    }
    return dp[n-1][maxWeight];
}
//spaceOptimized
int knapsack(vector<int>wt, vector<int>val, int n, int maxWeight){
    vector<int> prev(n,0);

    for(int W= wt[0]; W<= maxWeight; W++)  prev[W] = val[0];

    for(int ind = 1; ind<n; ind++){
        vector<int> cur(n,0);
        for(int W=0; W<=maxWeight; W++){
            int notTake = prev[W];
            int take = 0;
            if(wt[ind] <= W) take = prev[ W - wt[ind]];

            cur[W] = take + notTake;
        }
    }
```

```
    return prev[maxWeight];
}
//spaceOptimized-2
int knapsack(vector<int>wt, vector<int>val, int n, int maxWeight){
    vector<int> prev(n,0);

    for(int W= wt[0]; W<= maxWeight; W++)  prev[W] = val[0];

    for(int ind = 1; ind<n; ind++){
        vector<int> cur(n,0);
        for(int W=maxWeight; W>=0; W--){
            int notTake = prev[W];
            int take = 0;
            if(wt[ind] <= W) take = prev[ W - wt[ind]];

            prev[W] = take + notTake;
        }
    }
    return prev[maxWeight];
}


//Minimum Coins
//recursion
int f(int ind, int T, vector<int>& nums){
    if(ind == 0){
        if( T% nums[0] == 0) return T/nums[0];
        return 1e9;
    }
    int notTake = 0 + f(ind-1, T, nums);
    int take = INT_MAX;
    if(nums[ind] <= T)  take = 1+ f(ind, T-nums[ind], nums);

    return min(take, notTake);
}
//DP
int f(int ind, int T, vector<int>& nums, vector<vector<int>>&dp){
    if(ind == 0){
```

```cpp
        if( T% nums[0] == 0) return T/nums[0];
        return 1e9;
    }
    if(dp[ind][T]!= -1) return dp[ind][T];
    int notTake = 0 + f(ind-1, T, nums);
    int take = INT_MAX;
    if(nums[ind] <= T)  take = 1+ f(ind, T-nums[ind], nums);

    return  dp[ind][T] = min(take, notTake);
}
//Tabulation
int minimumElements(vector<int>&nums, int target){
    int n= nums.size();
    vector<vector<int>> dp(n, vector<int>(target+1, 0));

    for(int T=0; T <= target; T++){
        if(T% nums[0] == 0) dp[0][T] = T / nums[0];
        else dp[0][T] = 1e9;
    }

    for(int ind = 1; ind <n; ind++){
        for(int T=0; T<=target; T++){
            int notTake = dp[ind-1][T];
            int take = INT_MAX;

            if(nums[ind] <= T){
                take = 1 + dp[ind][T-nums[ind]];
            }

            dp[ind][T] =  min(take, notTake);
        }
    }
    return dp[n-1][target];
}
//SpaceOPtimized
int minimumElements(vector<int>&nums, int target){
    int n= nums.size();
    vector<int>prev(n, 0);
```

```
    for(int T=0; T <= target; T++){
        if(T% nums[0] == 0) prev[T] = T / nums[0];
        else prev[T] = 1e9;
    }

    for(int ind = 1; ind <n; ind++){
        vector<int> cur(n, 0);
        for(int T=0; T<=target; T++){
            int notTake = prev[T];
            int take = INT_MAX;

            if(nums[ind] <= T){
                take = 1 + cur[T-nums[ind]];
            }

            cur[T] =  min(take, notTake);
        }
    }
    return prev[target];
}
int minimumElements(vector<int>&nums, int target){
    int n= nums.size();
    vector<int>prev(n, 0);

    for(int T=0; T <= target; T++){
        if(T% nums[0] == 0) prev[T] = T / nums[0];
        else prev[T] = 1e9;
    }

    for(int ind = n-1; ind <n; ind++){
        for(int T=0; T<=target; T++){
            int notTake = prev[T];
            int take = INT_MAX;

            if(nums[ind] <= T){
                take = 1 + prev[T-nums[ind]];
            }
```

```
            prev[T] =  min(take, notTake);
        }
    }
    return prev[target];
}



//TargetSum -- SIMILAR--> +,-
//Count Partitions with given DIfference
//(totSum-D)/2   -subsSetSum


//Coin Change - 2
long f(int ind, int T, int *a){
    if(ind == 0) return T%a[0] == 0;

    long notTake = f(ind-1, T, a);
    long take = 0;
    if(a[ind]<= T) take = f(ind, T-a[ind], a);
    return take+notTake;
}
//dp
long f(int ind, int T, int *a, vector<vector<int>>&dp){
    if(ind == 0) return T%a[0] == 0;
    if(dp[ind][T]!=-1) return dp[ind][T];

    long notTake = f(ind-1, T, a);
    long take = 0;
    if(a[ind]<= T) take = f(ind, T-a[ind], a);
    return dp[ind][T] = take+notTake;
}
//tabulation
long countWays(int *a, int n, int value){
    vector<vector<int>> dp(n, vector<int>(n+1, 0));

    for(int T=0; T<=value; T++) dp[0][T] = T % a[0];
```

```
    for(int ind=1;ind<n;ind++){
        for(int T=0; T<=value; T++){
            long notTake = dp[ind-1][T];
            long take = 0;

            if(a[ind] <= T) take= dp[ind][T-a[ind]];
            dp[ind][T] = take+notTake;
        }
    }
    return dp[n-1][value];
}
//space-Optimized
long countWays(int *a, int n, int value){
    vector<int> prev(value+1,0);

    for(int T=0; T<=value; T++) prev[T] = T % a[0];

    for(int ind=1;ind<n;ind++){
        vector<int> cur(value+1,0);
        for(int T=0; T<=value; T++){
            long notTake = prev[T];
            long take = 0;

            if(a[ind] <= T) take= cur[T-a[ind]];
            cur[T] = take+notTake;
        }
        prev =cur;
    }
    return prev[value];
}


//UnBounded KnapSack
//recursion
int f(int ind, int W, vector<int> &val, vector<int> &wt){
    if(ind == 0) return ((int)(W/wt[0]) * val[0]);
```

```
    int notTake = 0 + f(ind-1, W, val, wt);
    int take =0;
    if(wt[ind] <= W){
        take = val[ind] + f(ind, W - wt[ind], val, wt);
    }
    return max(take, notTake);
}
//DP
int f(int ind, int W,vector<int> &val, vector<int> &wt,
vector<vector<int>>&dp ){
    if(ind == 0) return ((int)(W/wt[0]) * val[0]);
    if(dp[ind][W]!= -1) return dp[ind][W];

    int notTake = 0 + f(ind-1, W, val, wt, dp);
    int take =0;
    if(wt[ind] <= W){
        take = val[ind] + f(ind, W - wt[ind], val, wt, dp);
    }
    return dp[ind][W] = max(take, notTake);
}
//tabulation
int unboundedKnapsack(int n, int w, vector<int>&val, vector<int>&wt){
    vector<vector<int>> dp(n, vector<int>(W+1,0));
    for(int W=0; W<=w; W++){
        dp[0][W] = ((int)(W/wt[0]) * val[0]);
    }
    for(int ind =1; ind<n; ind++){
        for(int W=0; W<=w; W++){
            int notTake = 0 + dp[ind-1][W];
            int take = 0;
            if(wt[ind] <= W){
                take = val[ind] + dp[ind][W -wt[ind]];
            }

            dp[ind][W] = max(take, notTake);
        }
    }
    return dp[n-1][w];
```

```
}
//space-Optimization
int unboundedKnapsack(int n, int w, vector<int>&val, vector<int>&wt){
    vector<int> prev(w+1,0);

    for(int W=0; W<=w; W++){
        prev[W] = ((int)(W/wt[0]) * val[0]);
    }
    for(int ind =1; ind<n; ind++){
        vector<int> cur(w+1,0);
        for(int W=0; W<=w; W++){
            int notTake = 0 + prev[W];
            int take = 0;
            if(wt[ind] <= W){
                take = val[ind] + cur[W -wt[ind]];
            }

            cur[W] = max(take, notTake);
        }
        prev = cur;
    }
    return prev[w];
}
//space-Optimization
int unboundedKnapsack(int n, int w, vector<int>&val, vector<int>&wt){
    vector<int> prev(w+1,0);

    for(int W=0; W<=w; W++){
        prev[W] = ((int)(W/wt[0]) * val[0]);
    }
    for(int ind =1; ind<n; ind++){
        for(int W=0; W<=w; W++){
            int notTake = 0 + prev[W];
            int take = 0;
            if(wt[ind] <= W){
                take = val[ind] + prev[W -wt[ind]];
            }
```

```
            prev[W] = max(take, notTake);
        }
    }
    return prev[w];
}




//Rod Cutting Problem
//recursion
int f(int ind, int N, vector<int> &price){
    if(ind == 0) return N*price[0];

    int notTake = 0 + f(ind-1, N, price);
    int take = INT_MIN;
    int rodLength = ind+1;
    if(rodLength<=N){
        take = price[ind] + f(ind-1, N-rodLength, price);
    }
    return max(take, notTake);
}
//dp
int f(int ind, int N, vector<int> &price, vector<vector<int>>&dp){
    if(ind == 0) return N*price[0];
    if(dp[ind][N]!=-1) return dp[ind][N];

    int notTake = 0 + f(ind-1, N, price, dp);
    int take = INT_MIN;
    int rodLength = ind+1;
    if(rodLength<=N){
        take = price[ind] + f(ind-1, N-rodLength, price, dp);
    }
    return dp[ind][N] = max(take, notTake);
}
//tabulation
int cutRod(vector<int>&prices, int n){
    vector<vector<int>> dp(n, vector<int>(n+1, 0));
```

```
    for(int N=0; N<=n; N++){
        dp[0][N] = N * prices[0];
    }

    for(int ind = 1; ind<n; ind++){
        for(int N=0; N<=n; N++){
            int notTake = 0 + dp[ind-1][N];
            int take = INT_MIN;
            int rodLength = ind+1;
            if(rodLength <= N){
                take = prices[ind] + dp[ind][N - rodLength];
            }
            dp[ind][N] = max(take, notTake) ;
        }
    }
    return dp[n-1][n];
}
//spaceOptimization
int cutRod(vector<int>&prices, int n){
    vector<int> prev(n+1,0);

    for(int N=0; N<=n; N++){
        prev[N] = N * prices[0];
    }

    for(int ind = 1; ind<n; ind++){
        vector<int> cur(n+1,0);
        for(int N=0; N<=n; N++){
            int notTake = 0 + prev[N];
            int take = INT_MIN;
            int rodLength = ind+1;
            if(rodLength <= N){
                take = prices[ind] + cur[N - rodLength];
            }
            cur[N] = max(take, notTake) ;
        }
        prev = cur;
```

```
    }
    return prev[n];
}
//spaceOptimization
int cutRod(vector<int>&prices, int n){
    vector<int> prev(n+1,0);

    for(int N=0; N<=n; N++){
        prev[N] = N * prices[0];
    }

    for(int ind = 1; ind<n; ind++){
        for(int N=0; N<=n; N++){
            int notTake = 0 + prev[N];
            int take = INT_MIN;
            int rodLength = ind+1;
            if(rodLength <= N){
                take = prices[ind] + prev[N - rodLength];
            }
            prev[N] = max(take, notTake) ;
        }
    }
    return prev[n];
}
```

//Minimum cost to fill given weight in a bag

Instead of doing maximization, here we need to do minimization.

```
//LCS- Longest Commom Subsequence
//recursion
int f(int i, int j, string &s , string &t){
    if(i<0 || j<0) return 0;
    if(s[i] == t[j]) return 1+ f(i-1, j-1, s, t);
    return max(f(i-1, j, s, t), f(i, j-1, s, t));
}
```

```cpp
//dp
int f(int i, int j, string &s , string &t, vector<vector<int>>dp){
    if(i<0 || j<0) return 0;
    if(dp[i][j]!=-1) return dp[i][j];
    if(s[i] == t[j]) return 1+ f(i-1, j-1, s, t, dp);
    return dp[i][j] = max(f(i-1, j, s, t, dp), f(i, j-1, s, t, dp));
}
//tabulation
int lcs(string s, string t){
    int n= s.size();
    int m= t.size();

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

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[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[n][m];
}
//space-optimization
int lcs(string s, string t){
    int n= s.size();
    int m= t.size();

    vector<int> prev(n,0);

    for(int i=1; i<=n; i++){
        vector<int> cur(n,0);
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]) cur[j] = 1+prev[j-1];
            else cur[j] = max(prev[j], cur[j-1]);
        }
    }
    return prev[m];
}
```

```cpp
//Print Longest Common SubSequence
string printLCS(string s, string t){
    int n= s.size();
    int m= t.size();

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

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]) dp[i][j] = 1+dp[i-1][j-1];
            else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }

    int len = dp[n][m];

    string ans = "";
    for(int i=0;i<len;i++) ans+='$';

    int index = len-1;
    int i=n, j=m;
    while(i>0 && j>0){
        if(s[i-1] == t[j-1]){
            ans[index] = s[i-1];
            index--;
            i--; j--;
        }
        else if(dp[i-1][j] > dp[i][j-1]){
            i--;
        }
        else{
            j--;
        }
    }
    return ans;
}
```

```cpp
//Longest Common Substring
//dp
int lcs(string s, string t){
    int n= s.size();
    int m= t.size();
    int ans = INT_MIN;
    vector<vector<int>> dp(n+1, vector<int>(m+1,0));

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]) {
                dp[i][j] = 1+dp[i-1][j-1];
                ans = max(ans, dp[i][j]);
            }
            else dp[i][j] = 0;
        }
    }
    return ans;
}
//space-Optimization
int lcs(string s, string t){
    int n= s.size();
    int m= t.size();
    int ans = INT_MIN;
    vector<int> prev(n+1, 0);

    for(int i=1; i<=n; i++){
        vector<int> cur(n+1, 0);
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]) {
                cur[j] = 1+prev[j-1];
                ans = max(ans, cur[j]);
            }
            else cur[j] = 0;
        }
        prev = cur;
```

```
    }
    return ans;
}


//Longest Palindromic subsequence
// LCS of array and reverseArray

//Longest Palindromic substring
//Longest commom substring of array and ReverseArray




//Minimum insertions to make a String Palindrome
// count length of longest palindromic subsequence
// ans = s.size() - count ;


// Minimum insertion Deletions to convert string A to B
// count length of longest common subsequence
// deletions =  A.size() - count ;
// insertions = B.size() - count ;
// ans = deletions+insertions = n+m- 2*(len(lcs))



// Shortest Common SuperSequence
// length = a.size() + b.size() - lcs.size()
string shortestSupersequence(string s, string t){
    int n= s.size();
    int m= t.size();

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

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]) dp[i][j] = 1+dp[i-1][j-1];
```

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

    string ans = "";
    int i = n, j=m;
    while( i > 0  && j >0){
        if(s[i-1] == t[j-1]) {
            ans+= s[i-1];
            i--; j--;
        }
        else if(dp[i-1][j] > dp[i][j-1]){
            ans+= s[i-1];
            i--;
        }
        else{
            ans+= t[j-1];
            j--;
        }
    }
    while(i>0){
        ans+= s[i-1];
        i--;
    }
    while(j>0){
        ans+= s[j-1];
        j--;
    }
    reverse(ans.begin(), ans.end());
    return ans;
}



//Distinct Subsequences
//recursion
int f(int i, int j, string s, string t){
```

```
    if(j<0) return 1;
    if(i<0) return 0;

    if(s[i]==t[j]) return f(i-1,j-1,s,t)+f(i-1,j,s,t);
    return f(i-1,j,s,t);
}
//dp
int f(int i, int j, string s, string t, vector<vector<int>>& dp){
    if(j<0) return 1;
    if(i<0) return 0;

    if(dp[i][j]!=-1) return dp[i][j] ;

    if(s[i]==t[j]) return dp[i][j] = f(i-1,j-1,s,t)+f(i-1,j,s,t);
    return dp[i][j] = f(i-1,j,s,t);
}
//tabulation
int numDistinct(string s, string t) {
    int n = s.size();
    int m = t.size();
    vector<vector<double>> dp(n+1, vector<double>(m+1, 0));

    for(int i=0; i<=n; i++) dp[i][0] = 1;
    for(int j=1; j<=m; j++) dp[0][j] = 0; //j starting from j=1 not 0
    dp[0][0] =1;

    for(int i=1;i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]){
                dp[i][j] = dp[i-1][j-1] + dp[i-1][j];
            }
            else{
                dp[i][j] = dp[i-1][j];
            }
        }
    }
    return (int)dp[n][m];
}
```

```
    //use double to avoid data overflow, then typecast it

//spaceOptimization
int numDistinct(string s, string t) {
    int n = s.size();
    int m = t.size();
    vector<double> prev(m+1, 0), cur(m+1, 0);

    prev[0] = cur[0] = 1;
    for(int j=1; j<=m; j++) prev[j] = 0; //j starting from j=1 not 0

    for(int i=1;i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]){
                cur[j] = prev[j-1] + prev[j];
            }
            else{
                cur[j] = prev[j];
            }
        }
        prev = cur;
    }
    return (int)prev[m];
}
//spaceOptimization-2
int numDistinct(string s, string t) {
    int n = s.size();
    int m = t.size();
    vector<double> prev(m+1, 0);

    prev[0]  = 1;
    for(int j=1; j<=m; j++) prev[j] = 0; //j starting from j=1 not 0

    for(int i=1;i<=n; i++){
        for(int j=1; j<=m; j++){
            if(s[i-1] == t[j-1]){
                prev[j] = prev[j-1] + prev[j];
            }
```

```
            else{
                prev[j] = prev[j];
            }
        }
    }
    return (int)prev[m];
}




//Edit Distance
//recursion
int f(int i, int j, string s1, string s2){
    if(i<0) return j+1;
    if(j<0) return i+1;
    if(s1[i] == s2[j]) return f(i-1, j-1, s1, s2);
    return 1+ min({ f(i-1, j), f(i, j-1), f(i-1, j-1) });
}
//DP
int f(int i, int j, string s1, string s2, vector<vector<int>>& dp){
    if(i<0) return j+1;
    if(j<0) return i+1;
    if(dp[i][j]!= -1) return dp[i][j];
    if(s1[i] == s2[j]) return dp[i][j] = f(i-1, j-1, s1, s2);
    return dp[i][j] = 1+ min({ f(i-1, j), f(i, j-1), f(i-1, j-1) });
}
//tabulation
int editDistance(string s1, string s2){
    int n= s1.size();
    int m= s2.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

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

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

```
        for(int j=1;j<=m; j++){
            if(s1[i-1] == s2[j-1]) return dp[i][j] = dp[i-1][j-1];
            else{
                dp[i][j] = min({ dp[i-1][j], dp[i][j-1],
dp[i-1][j-1]});
            }
        }
    }
    return dp[n][m];
}
//spaceOptimization
int editDistance(string s1, string s2){
    int n= s1.size();
    int m= s2.size();
    vector<int> prev(m+1, 0), cur(m+1, 0);

    for(int j=0; j<=m; j++) prev[j] = j;

    for(int i=1; i<=n; i++){
        cur[0] = i; //important for(int i=0; i<=n; i++) dp[i][0] = i;
        for(int j=1;j<=m; j++){
            if(s1[i-1] == s2[j-1]) return cur[j] = prev[j-1];
            else{
                cur[j] = min({ prev[j], cur[j-1], prev[j-1]});
            }
        }
        prev = cur;
    }
    return prev[m];
}
```

GOLD MINE PROBLEM

```
//recursive
```

```
int collectGold(vector<vector<int>> gold, int r, int c, int n, int m)
{
    if ((r < 0) || (r == n) || (c == m)) {
        return 0;
    }
    int rightUpperDiagonal = collectGold(gold, r - 1, c + 1, n, m);
    int right = collectGold(gold, r, c + 1, n, m);
    int rightLowerDiagonal = collectGold(gold, r + 1, c + 1, n, m);
    return  gold[r] + max(max(rightUpperDiagonal,
rightLowerDiagonal), right);
}
//dp
int collectGold(vector<vector<int>> gold, int r, int c, int n, int m,
vector<vector<int>> &dp) {
    if ((r < 0) || (r == n) || (c == m)) {
        return 0;
    }
    if(dp[r] != -1)
        return dp[r] ;
    int rightUpperDiagonal = collectGold(gold, r - 1, c + 1, n, m,
dp);
    int right = collectGold(gold, r, c + 1, n, m, dp);
    int rightLowerDiagonal = collectGold(gold, r + 1, c + 1, n, m,
dp);
    return dp[r] = gold[r] + max(max(rightUpperDiagonal,
rightLowerDiagonal), right);
}
//tabulation
int getMaxGold(int gold[][MAX], int m, int n)
{
    int goldTable[m][n];
    memset(goldTable, 0, sizeof(goldTable));

    for (int col=n-1; col>=0; col--)
    {
        for (int row=0; row<m; row++)
        {
            int right = (col==n-1)? 0: goldTable[row][col+1];
```

```
            int right_up = (row==0 || col==n-1)? 0:
                           goldTable[row-1][col+1];
            int right_down = (row==m-1 || col==n-1)? 0:
                            goldTable[row+1][col+1];
            goldTable[row][col] = gold[row][col] +
                             max(right, max(right_up, right_down));

        }
    }

    int res = goldTable[0][0];
    for (int i=1; i<m; i++)
        res = max(res, goldTable[i][0]);
    return res;
}


ASSEMBLY-LINE
int carAssembly(int a[][NUM_STATION],
                int t[][NUM_STATION],
                int *e, int *x)
{
    int T1[NUM_STATION], T2[NUM_STATION], i;
    T1[0] = e[0] + a[0][0];
    T2[0] = e[1] + a[1][0];
    for (i = 1; i < NUM_STATION; ++i)
    {
        T1[i] = min(T1[i - 1] + a[0][i],
                    T2[i - 1] + t[1][i] + a[0][i]);
        T2[i] = min(T2[i - 1] + a[1][i],
                    T1[i - 1] + t[0][i] + a[1][i]);
    }
    return min(T1[NUM_STATION - 1] + x[0],
               T2[NUM_STATION - 1] + x[1]);
}
```

//Painting Fence Problem
long countWays(int n, int k)

```
{
   long dp[n + 1];
   memset(dp, 0, sizeof(dp));
   long long mod = 1000000007;

   dp[1] = k;
   dp[2] = k * k;

   for (int i = 3; i <= n; i++) {
      dp[i] = ((k - 1) * (dp[i - 1] + dp[i - 2])) % mod;
   }

   return dp[n];
}
total[i] = same[i] + diff[i]
same[i]  = diff[i-1]
diff[i]  = (diff[i-1] + diff[i-2]) * (k-1)
         = total[i-1] * (k-1)
```

// Maximize the number of segments

```
//recursive + dp
int func(int l, int p, int q, int r)
{
   if(l==0)
     return 0;

    if(dp[l]!=-1)
      return dp[l];

    int a(INT_MIN),b(INT_MIN),c(INT_MIN);
    if(p<=l)
      a=func(l-p,p,q,r);

    if(q<=l)
      b=func(l-q,p,q,r);

    if(r<=l)
      c=func(l-r,p,q,r);
    return dp[l]=1+max({a,b,c});
}
//tabulation-coin change variation
int findMaximum(int l, int p, int q, int r)
{
   int dp[l + 1];
```

```
    memset(dp, -1, sizeof(dp));
    dp[0] = 0;

    for (int i = 0; i <= l; i++) {
        if (dp[i] == -1)
            continue;
        if (i + p <= l)
            dp[i + p] = max(dp[i + p], dp[i] + 1);
        if (i + q <= l)
            dp[i + q] = max(dp[i + q], dp[i] + 1);
        if (i + r <= l)
            dp[i + r] = max(dp[i + r], dp[i] + 1);
    }
    if (dp[l] == -1) {
        dp[l] = 0;
    }
    return dp[l];
}
```

//LCS (Longest Common Subsequence) of three strings

```
//recursion
int dp[100][100][100];
int lcsOf3(int i, int j,int k)
{
    if(i==-1||j==-1||k==-1)
        return 0;
    if(dp[i][j][k]!=-1)
        return dp[i][j][k];

    if(X[i]==Y[j] && Y[j]==Z[k])
        return dp[i][j][k] = 1+lcsOf3(i-1,j-1,k-1);
    else
        return dp[i][j][k] = max(max(lcsOf3(i-1,j,k),
                        lcsOf3(i,j-1,k)),lcsOf3(i,j,k-1));
}
//tabulation
int lcsOf3( string X, string Y, string Z, int m,
                            int n, int o)
{
    int L[m+1][n+1][o+1];
    for (int i=0; i<=m; i++) {
        for (int j=0; j<=n; j++) {
```

```
            for (int k=0; k<=o; k++) {
                if (i == 0 || j == 0||k==0)
                    L[i][j][k] = 0;

                else if (X[i-1] == Y[j-1] && X[i-1]==Z[k-1])
                    L[i][j][k] = L[i-1][j-1][k-1] + 1;

                else
                    L[i][j][k] = max(max(L[i-1][j][k],
                                         L[i][j-1][k]),
                                     L[i][j][k-1]);
            }
        }
    }
    return L[m][n][o];
}
```

//Maximum Sum Increasing Subsequence

```
int maxSumIS(int arr[], int n)
{
    int i, j, max = 0;
    int msis[n];
    for ( i = 0; i < n; i++ )
        msis[i] = arr[i];
    for ( i = 1; i < n; i++ )
        for ( j = 0; j < i; j++ )
            if (arr[i] > arr[j] &&
                msis[i] < msis[j] + arr[i])
                msis[i] = msis[j] + arr[i];
    return *max_element(msis.begin(), msis.end());
}
```

//Count all subsequences having product less than K

```
int productSubSeqCount(vector<int> &arr, int k)
{
    int n = arr.size();
    int dp[k + 1][n + 1];
    memset(dp, 0, sizeof(dp));

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

```
            dp[i][j] = dp[i][j - 1];
            if (arr[j - 1] <= i)
                dp[i][j] += dp[i/arr[j-1]][j-1] + 1;
        }
    }
    return dp[k][n];
}



int longestSubseqWithDiffOne(int arr[], int n)
{
    int dp[n];
    for (int i = 0; i < n; i++)
        dp[i] = 1;
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (abs(arr[i]-arr[j]) <=1)
                dp[i] = max(dp[i], dp[j] + 1);
        }
    }
    int result = 1;
    for (int i = 0; i < n; i++)
        if (result < dp[i])
            result = dp[i];
    return *max_element(dp.begin(), dp.end());
}



//Maximum subsequence sum such that no three are consecutive
int maxSumWO3Consec(int n)
{
    if(sum[n]!=-1) return sum[n];
    if(n==0) return sum[n] = 0;
    if(n==1) return sum[n] = arr[0];
    if(n==2) return sum[n] = arr[1]+arr[0];

    return sum[n] = max(max(maxSumWO3Consec(n-1),
                maxSumWO3Consec(n-2) + arr[n]),
                arr[n] + arr[n-1] + maxSumWO3Consec(n-3));


}
```

```
int maxSumWO3Consec(int arr[], int n)
{
    int sum[n];
    if (n >= 1)
        sum[0] = arr[0];

    if (n >= 2)
        sum[1] = arr[0] + arr[1];

    if (n > 2)
        sum[2] = max(sum[1], max(arr[1] +
                            arr[2], arr[0] + arr[2]));
    for (int i = 3; i < n; i++)
        sum[i] = max(max(sum[i - 1], sum[i - 2] + arr[i]),
                    arr[i] + arr[i - 1] + sum[i - 3]);

    return sum[n - 1];
}
```

// Count All Palindromic Subsequence in a given String

```
int countPS(string str)
{
    int N = str.length();
    int cps[N + 1][N + 1];
    memset(cps, 0, sizeof(cps));

    for (int i = 0; i < N; i++)
        cps[i][i] = 1;

    for (int L = 2; L <= N; L++) {
        for (int i = 0; i <= N-L; i++) {
            int k = L + i - 1;
            if (str[i] == str[k])
                cps[i][k]=
                    cps[i][k-1]+cps[i+1][k]+ 1;
            else
                cps[i][k] =
                    cps[i][k-1]+cps[i+1][k]-cps[i+1][k-1];
        }
    }
    return cps[0][N - 1];
}
```

//Binomial Coefficient

```
int binomialCoeff(int n, int k)
{
    int C[n + 1][k + 1];
    int i, j;

    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= min(i, k); j++) {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }

    return C[n][k];
}
int binomialCoeffUtil(int n, int k, int** dp)
{
    if (dp[n][k] != -1)
        return dp[n][k];

    if (k == 0 || k == n) {
        dp[n][k] = 1;
        return dp[n][k];
    }

    dp[n][k] = binomialCoeffUtil(n - 1, k - 1, dp) +
               binomialCoeffUtil(n - 1, k, dp);
    return dp[n][k];
}
```

Permutation Coefficient

```
int permutationCoeff(int n, int k)
```

```
{
    int P[n + 1][k + 1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= std::min(i, k); j++) {
            if (j == 0)
                P[i][j] = 1;
            else
                P[i][j] = P[i - 1][j] +
                        (j * P[i - 1][j - 1]);
            P[i][j + 1] = 0;
        }
    }
    return P[n][k];
}
```

//catalan no

```
//recursive
unsigned long int catalan(unsigned int n)
{
    if (n <= 1)  return 1;
    unsigned long int res = 0;
    for (int i = 0; i < n; i++)
        res += (catalan(i)  * catalan(n - i - 1));
    return res;
}
//dp
unsigned long int catalanDP(unsigned int n)
{
    unsigned long int catalan[n + 1];
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= n; i++) {
        catalan[i] = 0;
        for (int j = 0; j < i; j++)
            catalan[i] += catalan[j] * catalan[i - j - 1];
    }
    return catalan[n];
}
```

Interleaved Strings

```
bool isInterleaved(
    char* A, char* B, char* C)
    if (!(*A || *B || *C)) return true;
```

```
    if (*C == '\0')  return false;
    return ((*C == *A) && isInterleaved(
                            A + 1, B, C + 1))
           || ((*C == *B) && isInterleaved(
                              A, B + 1, C + 1));
}
bool isInterleaved(
    char* A, char* B, char* C)
    int M = strlen(A), N = strlen(B);
    bool IL[M + 1][N + 1];
    memset(IL, 0, sizeof(IL));
    if ((M + N) != strlen(C))
        return false;
    for (int i = 0; i <= M; ++i) {
        for (int j = 0; j <= N; ++j) {
            if (i == 0 && j == 0)
                IL[i][j] = true;

            else if (i == 0) {
                if (B[j - 1] == C[j - 1])
                    IL[i][j] = IL[i][j - 1];
            }

            else if (j == 0) {
                if (A[i - 1] == C[i - 1])
                    IL[i][j] = IL[i - 1][j];
            }

            else if (
                A[i - 1] == C[i + j - 1]
                && B[j - 1] != C[i + j - 1])
                IL[i][j] = IL[i - 1][j];

            else if (
                A[i - 1] != C[i + j - 1]
                && B[j - 1] == C[i + j - 1])
                IL[i][j] = IL[i][j - 1];
            else if (
                A[i - 1] == C[i + j - 1]
                && B[j - 1] == C[i + j - 1])
                IL[i][j]
                    = (IL[i - 1][j]
                       || IL[i][j - 1]);
        }
```

```
    }
    return IL[M][N];
}
```

WildCard Matching

```
//recursion
bool f(int i, int j, string &pattern, string &text){
    if(i<0 && j <0) return 1;
    if(i<0 && j >=0) return 0;

    if(j<0 && i>=0){
        for(int ii=0; ii<=i; i++){
            if(pattern[ii]!='*') return 0;
        }
        return 1;
    }

    if(pattern[i] == text[j] || pattern[i]=='?'){
        return f(i-1,j-1,pattern, text);
    }
    if(pattern[i] == '*'){
        return f(i-1, j, pattern, text) | f(i, j-1, pattern, text);
    }
    return 0;
}
//dp
bool f(int i, int j, string &pattern, string &text, vector<vector<int>>&dp){
    if(i<0 && j <0) return 1;
    if(i<0 && j >=0) return 0;

    if(j<0 && i>=0){
        for(int ii=0; ii<=i; i++){
            if(pattern[ii]!='*') return 0;
        }
        return 1;
    }

    if(dp[i][j]!=-1) return dp[i][j];

    if(pattern[i] == text[j] || pattern[i]=='?'){
        return dp[i][j] = f(i-1,j-1,pattern, text, dp);
    }
    if(pattern[i] == '*'){
```

```
        return dp[i][j] = f(i-1, j, pattern, text, dp) | f(i, j-1, pattern, text, dp);
    }
    return 0;
}

//tabulation
bool wildCardMatching(string pattern, string text){
    int n= pattern.size(), m = text.size();
    vector<vector<bool>> dp(n+1, vector<bool>(m+1, false));

    dp[0][0] = true;
    for(int j=1; j<=m; j++){
        dp[0][j] = false;
    }

    for(int i=1; i<=n; i++){
        int flag = true;
        for(int ii = 1; ii<=i; ii++){
            if(pattern[ii-1] != '*'){
                flag = false;
                break;
            }
        }
        dp[i][0] = flag;
    }

    for(int i=1;i<=n; i++){
        for(int j=1; j<=m; j++){
            if(pattern[i-1] == text[j-1] || pattern[i-1]=='?'){
                dp[i][j] = dp[i-1][j-1];
            }
            else if(pattern[i-1]=='*'){
                dp[i][j] = dp[i-1][j] | dp[i][j-1];
            }
            else dp[i][j] = 0;
        }
    }
    return dp[n][m];
}

//spaceOPtimization
bool wildCardMatching(string pattern, string text){
    int n= pattern.size(), m = text.size();
    vector<bool>cur(n+1, 0), prev(n+1, 0);
```

```
    prev[0] = true;
    for(int j=1; j<=m; j++){
        prev[j] = false;
    }

    for(int i=1;i<=n; i++){
        int flag = true;
        for(int ii = 1; ii<=i; ii++){
            if(pattern[ii-1] != '*'){
                flag = false;
                break;
            }
        }
        cur[0] = flag;

        for(int j=1; j<=m; j++){
            if(pattern[i-1] == text[j-1] || pattern[i-1]=='?'){
                cur[j] = prev[j-1];
            }
            else if(pattern[i-1]=='*'){
                cur[j] = prev[j] | cur[j-1];
            }
            else cur[j] = 0;
        }
        prev = cur;
    }
    return prev[m];
}



//Best time to buy and Sell Stock -1
int maximumProfit(vector<int>prices){
    int miniPrice = prices[0];
    int maxProfit = 0;
    int n = profit.size();

    for(int i=0;i<n;i++){
        int earn = prices[i]-miniPrice;
        maxProfit = max(maxProfit, earn);
        miniPrice = min(miniPrice, prices[i]);
    }
    return maxProfit;
```

```
}


//Best time to buy and Sell Stock -2 (buy sell infinte times)
//recursion
long f(int ind, int buy, long *values, int n){
    if(ind == n) return 0;
    long profit = 0;
    if(buy){
        profit = max(-value[ind] + f(ind+1,0, values, n),
                0 + f(ind+1, 1, values, n));
    }
    else{
        profit = max(values[ind] + f(ind+1, 1, values, n),
                0 + f(ind+1, 0, values, n));
    }
    return profit;
}
//dp
long f(int ind, int buy, long *values, int n, vector<vector<int>>&dp){
    if(ind == n) return 0;
    long profit = 0;
    if(dp[ind][buy]!=-1) return dp[ind][buy];
    if(buy){
        profit = max(-value[ind] + f(ind+1,0, values, n, dp),
                0 + f(ind+1, 1, values, n, dp));
    }
    else{
        profit = max(values[ind] + f(ind+1, 1, values, n, dp),
                0 + f(ind+1, 0, values, n, dp));
    }
    return dp[ind][buy] = profit;
}
//tabulation
long getMaximumProfit(long *values, int n){
    vector<vector<long>> dp(n+!, vector<long>(2, 0));

    dp[n][0] = dp[n][1] = 0;
    for(int ind = n-1; ind>=0; ind--){
        for(int buy=0; buy<=1;buy++){
            long profit = 0;
            if(buy){
                profit = max(-value[ind] + dp[ind+1][0], 0+dp[ind+1][1]);
            }else{
```

```
            profit = max(value[ind] + dp[ind+1][1], 0+dp[ind+1][0]);
        }
        dp[ind][buy] = profit;
      }
    }
}
//spaceOptimization
long getMaximumProfit(long *values, int n){
    vector<long> ahead(2, 0), cur(2, 0);

    ahead[0] = ahead[1] = 0;

    for(int ind = n-1; ind>=0; ind--){
        for(int buy=0; buy<=1;buy++){
            long profit = 0;
            if(buy){
                profit = max(-value[ind] + ahead[0], 0+ahead[1]);
            }else{
                profit = max(value[ind] + ahead[1], 0+ahead[0]);
            }
            cur[buy] = profit;
        }
        ahead = cur;
    }
    return ahead[1];
}
//Best Way
long getMaximumProfit(long *prices, int n){
    int ret = 0;
    for (int p = 1; p < prices.size(); ++p)
      ret += max(prices[p] - prices[p - 1], 0);
    return ret;
}



//Best time to buy and Sell Stock -3 (buy sell 2 times)
//recursion
long f(int ind, int buy, long *values, int n, int cap){
    if(ind == n || cap == 0) return 0;
    long profit = 0;
    if(buy){
        profit = max(-value[ind] + f(ind+1,0, values, n, cap),
                    0 + f(ind+1, 1, values, n, cap));
```

```
    }
    else{
        profit = max(values[ind] + f(ind+1, 1, values, n, cap-1),
                0 + f(ind+1, 0, values, n, cap));
    }
    return profit;
}
//dp
long f(int ind, int buy,int cap, long *values, int n, vector<vector<vector<long>>>&dp){
    if(ind == n || cap==0) return 0;
    long profit = 0;
    if(dp[ind][buy][cap]!=-1) return dp[ind][buy][cap];
    if(buy){
        profit = max(-value[ind] + f(ind+1,0, values, n, dp),
                0 + f(ind+1, 1, values, n, dp));
    }
    else{
        profit = max(values[ind] + f(ind+1, 1, values, n, dp),
                0 + f(ind+1, 0, values, n, dp));
    }
    return dp[ind][buy][cap] = profit;
}
//tabulation
long getMaximumProfit(long *values, int n, int cap){
    //cap=2
    vector<vector<vector<long>>> dp(n+1, vector<vector<long>>(2, vector<int>(cap,-1)));

    dp[n][0] = dp[n][1] = 0;
    for(int ind = n-1; ind>=0; ind--){
        for(int buy=0; buy<=1;buy++){
            for(int cap=0; cap<=2; cap++){
                if(buy){
                    dp[ind][buy][cap] = max(-value[ind] + dp[ind+1][0], 0+dp[ind+1][1]);
                }else{
                    dp[ind][buy][cap] =  max(value[ind] + dp[ind+1][1], 0+dp[ind+1][0]);
                }
            }
        }
    }
    return dp[0][1][2];
}
//spaceOPtimization
long getMaximumProfit(long *values, int n, int cap){
    //cap=2
```

```
    vector<vector<vector<long>>> dp(n+1, vector<vector<long>>(2, vector<int>(cap,-1)));

    dp[n][0] = dp[n][1] = 0;
    for(int ind = n-1; ind>=0; ind--){
        for(int buy=0; buy<=1;buy++){
            for(int cap=0; cap<=2; cap++){
                if(buy){
                    dp[ind][buy][cap] = max(-value[ind] + dp[ind+1][0], 0+dp[ind+1][1]);
                }else{
                    dp[ind][buy][cap] =  max(value[ind] + dp[ind+1][1], 0+dp[ind+1][0]);
                }
            }
        }
    }
    return dp[0][1][2];
}
//Best Method
int maxProfit(vector<int>& prices) {
    int minPrice1 = INT_MAX, minPrice2 = INT_MAX;
    int profit1 = 0, profit2 = 0;
    int n= prices.size();
    for(int i=0; i < n ; i++){
        minPrice1 = min(prices[i], minPrice1);
        profit1 = max(profit1, prices[i] - minPrice1);
        minPrice2 = min(minPrice2, prices[i]-profit1);
        profit2 = max(profit2, prices[i] - minPrice2);
    }
    return profit2;
}




//Best time to buy and Sell Stock -4 (buy sell k times)
//bestway
int maxProfit(int k, vector<int>& prices) {
    if(k<=0 || prices.size()<=0) return 0;
    int n = prices.size();
    vector<int> minPrice(k,INT_MAX), maxProfit(k, 0);

    for(int i=0; i < n; i++){
        for(int j=0; j<k; j++){
            minPrice[j] = min(minPrice[j], prices[i]- (j>0?maxProfit[j-1]:0));
            maxProfit[j] = max(maxProfit[j], prices[i]-minPrice[j]);
```

```
        }
    }
    return maxProfit[k-1];
}

//Buy and sell stock with cooldown
// use code for buy and sell 2, and while selling
    // if(buy){
    //     profit = max(-value[ind] + f(ind+1,0, values, n, dp),
    //                 0 + f(ind+1, 1, values, n, dp));
    // }
    // else{
    //     //change it by
    //     profit = max(values[ind] + f(ind+ cooldownPeriod, 1, values, n, dp),
    //                 0 + f(ind+1, 0, values, n, dp));
    // }


// Buy and sell Stocks with Transaction Fees
//  use code for buy and sell 2, and while selling
    // if(buy){
    //     profit = max(-value[ind] -transactionFees + f(ind+1,0, values, n, dp),
    //                 0 + f(ind+1, 1, values, n, dp));
    // }
    // else{
    //     profit = max(values[ind] + f(ind+ 1, 1, values, n, dp),
    //                 0 + f(ind+1, 0, values, n, dp));
    // }



//Longest Increasing Subsequence
//recursion
int f(int ind, int prev_ind, int arr[], int n){
    if(ind == n) return 0;

    int len = 0 + f(ind+1, prev_ind, arr, n);
    if(prev_ind == -1 || arr[ind] > arr[prev_ind]){
        len = max(len, 1 + f(ind+1, ind, arr, n));
    }
    return len;
}
//dp
int f(int ind, int prev_ind, int arr[], int n, vector<vector<int>>&dp){
```

```
    if(ind == n) return 0;
    if(dp[ind][prev_ind+1]!=-1) return dp[ind][prev_ind+1];

    int len =0 + f(ind+1, prev_ind, arr, n, dp);
    if(prev_ind == -1 || arr[i]> arr[prev_ind]){
        len = max(len, 1 + f(ind+1, ind, arr, n, dp));
    }
    return dp[ind][prev_ind+1] = len;
}
//tabulation
int longestIncreasingSubsequence(int arr[], int n){
    vector<vector<int>> dp(n, vector<int>(n+1,0));
    for(int ind =n-1; ind>=0; ind--){
        for(int prev_ind= ind-1; prev_ind>=-1; prev_ind--){
            int len = 0 + dp[ind+1][prev_ind +1];
            if(prev_ind == -1 || arr[ind]>arr[prev_ind]){
                len = max(len, 1 +dp[ind+1][ind+1]);
            }
            dp[ind][prev_ind+1] = len;
        }
    }
    return dp[0][-1+1];
}
//spaceOptimization
//BestWay
int longestIncreasingSubsequence(int arr[], int n){
    vector<int>dp(n,1);
    int maxi = 1;
    for(int i=0; i<n; i++){
        for(int j=0; j<i; j++){
            if(arr[i]< arr[j]){
                dp[j] = max(dp[j], 1+ dp[i]);
            }
        }
        maxi = max(maxi, dp[i]);
    }
    return maxi;
}

//Printing Longest Increasing Subsequence
vector<int> longestIncreasingSubsequence(int arr[], int n){
    vector<int>dp(n+1,1), hash(n);
    int maxi = 1, lastIndex = 0;
    for(int i=0; i<n; i++){
```

```
        hash[i] = i;
        for(int prev = 0; prev <i; prev++){
            if(arr[prev] < arr[i] && 1 + dp[prev] > dp[i]){
                dp[i] = 1 + dp[prev];
                hash[i] = prev;
            }
        }
        if(dp[i]>maxi){
            maxi =  dp[i];
            lastIndex = i;
        }
    }

    vector<int> lis;
    lis.push_back(arr[lastIndex]);
    while(hash[lastIndex] != lastIndex){
        lastIndex = hash[lastIndex];
        lis.push_back(arr[lastIndex]);
    }
    reverse(lis.begin(), lis.end())
    return lis;
}


//Longest Increasing subsequence using Binary Search
int longestIncreasingSubsequence(int arr[], int n){
    vector<int> temp;
    temp.push_back(arr[0]);
    for(int i=1; i<n; i++){
        if(arr[i]>temp.back()){
            temp.push_back(arr[i]);
        } else{
            int ind = lower_bound(temp.begin(), temp.end(), arr[i]);
            temp[ind] = arr[ind];
        }
    }
    return temp.size();
}



//Largest Divisible Subset
    // 1. sort the array
    // 2. code--> Longest Increasing Subsequence
```

```
    // 3. Change condition if(arr[i]%arr[prev]==0 && 1 + dp[prev] > dp[i])



//Longest String Chain
    // variation of LIS


//Longest Bitonic Subsequence--> HILL by lcs
int longestBitonicSubsequence(vector<int>&arr, int n){
    vector<int> dp1(n,1);
    int maxi = 1;
    for(int i=0; i<n; i++)
        for(int j=0; j<i; j++)
            if(arr[i]< arr[j])
                dp1[j] = max(dp1[j], 1+ dp1[i]);

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


    vector<int> dp2(n,1);
    for(int i=0; i<n; i++)
        for(int j=0; j<i; j++)
            if(arr[i]< arr[j])
                dp2[j] = max(dp2[j], 1+ dp2[i]);

    int ans = 0;
    for(int i=0; i<n; i++){
        ans = max(ans, dp1[i]+dp2[i]);
    }
    return ans;
}


//count of longest increasing subsequences
int findNumberOfLIS(vector<int> &arr, int n){
    vector<int>dp(n, 1), cnt(n,1);
    int maxi = 1;
    for(int i=0; i<n; i++){
        for(int prev=0; prev<i; prev++){
            if(arr[prev] < arr[i] && 1+dp[prev]>dp[i]){
                dp[i] = 1+dp[prev];
                //inherit
                cnt[i] = cnt[prev];
```

```
            }
            else if(arr[prev] < arr[i] && 1+dp[prev]==dp[i]){
                //increase the count
                cnt[i]+=cnt[prev];
            }
        }
        maxi = max(maxi, dp[i]);
    }

    int nos = 0;
    for(int i=0;i<n;i++){
        if(dp[i]==maxi)
            nos+=cnt[i];
    }
    return nos;
}


//Partition DP
//recursion
int f(int i, int j, vector<int> &arr){
    if(i==j) return 0;
    int mini = 1e9;
    for(int k=i; k<j; k++){
        int steps = arr[i-1]*arr[k]*arr[j] + f(i,k,arr) +f(k+1,j,arr);
        mini = min(mini, steps);
    }
    return mini;
}
//dp
int f(int i, int j, vector<int> &arr, vector<vector<int>>&dp){
    if(i==j) return 0;
    if(dp[i][j!=-1]) return dp[i][j];

    int mini = 1e9;
    for(int k=i; k<j; k++){
        int steps = arr[i-1]*arr[k]*arr[j] + f(i,k,arr) +f(k+1,j,arr);
        mini = min(mini, steps);
    }
    return dp[i][j]= mini;
}
//tabulation
int matrixMultiplication(vector<int>&arr, int N){
    int dp[N][N];
```

```
    for(int i=1; i<N;i++) dp[i][i] = 0;

    for(int i=N-1; i>=1; i--){
        for(int j=i+1; j<N; j++){
            int mini = 1e9;
            for(int k=i; k<j;k++){
                int steps = arr[i-1]*arr[k]*arr[j] +
                    dp[i][k] + dp[k+1][j];
                mini = min(mini, steps);
            }
            dp[i][j] = mini;
        }
    }
    return dp[1][N-1];
}
```

//minimum cost to cut the stick
//burst baloons

Count Balanced Binary Trees of Height h

```
int countBT(int h)
{
    if (h == 0 || h == 1)
        return 1;
    return countBT(h-1) * (2 *countBT(h-2) +  countBT(h-1));
}
```

Largest rectangular sub-matrix whose sum is 0

```
int subsum(vector<int>& col_sum, int c) {
    int mx = 1;
    int curr_sum = 0;
    unordered_map<int, int> m;

    for (int i = 0;i < c;i++) {
        curr_sum += col_sum[i];
        if (curr_sum == 0) {
            mx = max(mx, i + 1);
        }
        else if (m[curr_sum]) {
```

```
            mx = max(mx, i - m[curr_sum] + 1);
        }
        else m[curr_sum] = i + 1;
    }
    return mx;
}

int matrix(vector<vector<int>> &mat){
    int mx = INT_MIN;
    for (int i=0; i<r; i++) {
        vector<int> col(c, 0);
        for (int j=i; j<r; j++) {
            for (int cl=0; cl<c; cl++)
                col_sum[cl] += arr[j][cl];

            mx = max(mx, subsum(col, c) * (j-i+1));
        }
    }
    return mx;
}
```

Largest area rectangular sub-matrix with equal number of 1's and 0's [IMP]

We will replace all 0s in the matrix by -1. Now we will find the submatrices with 0 sum and return the area of largest of them. A matrix with 0 sum has equal number of -1s and 1s, i.e. equal number of 0s and 1s.

Maximum sum rectangle in a 2D matrix

```
int kadane(int* arr, int* start, int* finish, int n){
    int sum = 0, maxSum = INT_MIN, i;
    *finish = -1;
    int local_start = 0;
    for (i = 0; i < n; ++i){
        sum += arr[i];
        if (sum < 0){
            sum = 0;
            local_start = i + 1;
        }
        else if (sum > maxSum){
            maxSum = sum;
            *start = local_start;
            *finish = i;
        }
```

```
    }
    if (*finish != -1)
        return maxSum;


    // Special Case: When all numbers in arr[] are negative
    maxSum = arr[0];
    *start = *finish = 0;

    for (i = 1; i < n; i++){
        if (arr[i] > maxSum){
            maxSum = arr[i];
            *start = *finish = i;
        }
    }
    return maxSum;
}

void findMaxSum(int M[][COL]){
    int maxSum = INT_MIN, finalLeft, finalRight, finalTop, finalBottom;

    int left, right, i;
    int temp[ROW], sum, start, finish;

    for (left = 0; left < COL; ++left) {
        memset(temp, 0, sizeof(temp));
        for (right = left; right < COL; ++right) {
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

            sum = kadane(temp, &start, &finish, ROW);

            if (sum > maxSum) {
                maxSum = sum;
                finalLeft = left;
                finalRight = right;
                finalTop = start;
                finalBottom = finish;
            }
        }
    }

    // Print final values
    cout << "(Top, Left) (" << finalTop << ", " << finalLeft << ")" << endl;
```

```
    cout << "(Bottom, Right) (" << finalBottom << ", " << finalRight << ")" << endl;
    cout << "Max sum is: " << maxSum << endl;
}
```

Maximum size square sub-matrix with all 1s

```
    int maximalSquare(vector<vector<char>>& matrix) {
        int n = matrix.size(), m = matrix[0].size(), res = 0;
        vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (matrix[i-1][j-1] == '1') {
                    dp[i][j] = min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]}) + 1;
                    res = max(res, dp[i][j]);
                }
            }
        }

        return res*res;
    }
```

Largest rectangle in a histogram

```
// o(n)
int largestRectangleArea(vector<int>& heights) {
        int n = heights.size();
        if(n == 0) return 0;
        int maxArea = 0;
        vector<int> left(n);
        vector<int> right(n);

        left[0] = -1;
        right[n - 1] = n;

        for(int i = 1; i < n; i++){
            int prev = i - 1;
            while(prev >= 0 && heights[prev] >= heights[i])
                prev = left[prev];
            left[i] = prev;
        }
```

```
        for(int i = n - 2; i >= 0; i--){
            int prev = i + 1;
            while(prev < n && heights[prev] >= heights[i])
                prev = right[prev];
            right[i] = prev;
        }

        for(int i = 0; i < n; i++){
            int width = right[i] - left[i] - 1;
            maxArea = max(maxArea, heights[i] * width);
        }
        return maxArea;
    }

//o(n)
int largestRectangleArea(vector<int>& heights) {
    int n = heights.size();
    int maxArea = 0;
    stack<int> st;

    for(int i = 0; i <= n; i++){
        int currHeight = i == n ? 0 : heights[i];
        while(!st.empty() && currHeight < heights[st.top()]){
            int top = st.top(); st.pop();
            int width = st.empty() ? i : i - st.top() - 1;
            int area = heights[top] * width;
            maxArea = max(area, maxArea);
        }
        st.push(i);
    }
    return maxArea;
}
```

Trapping rainwater

1. o(n^2)

```
int maxWater(int arr[], int n)
{
    int res = 0;
    for (int i = 1; i < n-1; i++) {
        int left = arr[i];
```

```
            for (int j=0; j<i; j++)
               left = max(left, arr[j]);
            int right = arr[i];
            for (int j=i+1; j<n; j++)
               right = max(right, arr[j]);
           res = res + (min(left, right) - arr[i]);
        }
        return res;
    }
```

2. o(n) o(n)

```
    int findWater(int arr[], int n)
    {
        int left[n], right[n];
        int water = 0;

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

        right[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--)
            right[i] = max(right[i + 1], arr[i]);

     for (int i = 1; i < n-1; i++)
        {
          int var=min(left[i-1],right[i+1]);
          if(var > arr[i])
          {
            water += var - arr[i];
          }
        }
        return water;
    }
```

3. o(n), o(1)

```
    int findWater(int arr[], int n)
    {
        int result = 0;
        int left_max = 0, right_max = 0;
        int lo = 0, hi = n - 1;

        while (lo <= hi) {
            if (arr[lo] < arr[hi]) {
                if (arr[lo] > left_max)
                    left_max = arr[lo];
                else
                    result += left_max - arr[lo];
                lo++;
            }
            else {
                if (arr[hi] > right_max)
                    right_max = arr[hi];
                else
                    result += right_max - arr[hi];
                hi--;
            }
        }

        return result;
    }
```

4. **Using Stacks**

```
    int maxWater(int height[], int n)
```

```
{
   stack <int> st;
   int ans = 0;
    for(int i = 0; i < n; i++)
   {
      while ((!st.empty()) &&
            (height[st.top()] < height[i]))
      {
         int pop_height = height[st.top()];
         st.pop();

         if (st.empty())
            break;
         int distance = i - st.top() - 1;

         int min_height = min(height[st.top()],
                              height[i]) -
                          pop_height;

         ans += distance * min_height;
      }

           st.push(i);
   }
   return ans;
}
```

Maximal Rectangle

```
    int largestArea(vector<int>& histogram){
         int n=histogram.size(), area=0;
         stack<int> s;

         for(int i=0; i<n; i++){
             while(!s.empty() && histogram[s.top()]>=histogram[i]){
                 int top = s.top();
                 s.pop();

                 int start;
                 if(s.empty())
                     start = -1;
                 else
                     start = s.top();


                 int curr_area = histogram[top] * (i - start -1);
                 area = max(area, curr_area);
             }
             s.push(i);
         }

         while(!s.empty()){
             int top = s.top();
```

```
            s.pop();

            int start;
            if(s.empty())
                start = -1;
            else
                start = s.top();

            int curr_area = histogram[top] * (n - start -1);
            area = max(area, curr_area);
        }

        return area;
    }

    int maximalRectangle(vector<vector<char>>& matrix) {
        int m=matrix.size();
        if(m==0) return 0;
        int n=matrix[0].size(), result=0;
        vector<int> histogram(n, 0);

        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
                if(matrix[i][j]=='1')
                    histogram[j]+=1;
                else
                    histogram[j]=0;
            }

            result = max(result, largestArea(histogram));
            cout<<result<<" ";
        }
        return result;
    }
```

The Skyline Problem

```
vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
    vector<vector<int>> skyline;
    map<int, vector<pair<int, int>>> map; // key : pos, value : vector of <height, start|end> pairs
    for (auto& building : buildings) {
        map[building[0]].push_back({building[2], 0}); // add startpoint
        map[building[1]].push_back({building[2], 1}); // add endpoint
    }
```

```
        multiset<int> q;
        for (auto& [pos, heights] : map) {
            for (auto& [height, type] : heights) {
                if (type == 0) q.insert(height);
                else q.erase(q.find(height));
            }
            int newHeight = q.empty() ? 0 : *q.rbegin();
            if (!skyline.empty() && skyline.back()[1] == newHeight) continue;
            else skyline.push_back(vector<int>({pos, newHeight}));
        }
        return skyline;
    }
```

Water Jug BFS

```
void BFS(int a ,int b, int target)
{
  map<pii, int>m;
  bool isSolvable =false;
  vector<tuple<int ,int ,int>>path;
  map<pii, pii>mp;

  queue<pii>q;

  q.push(make_pair(0,0));
  while(!q.empty())
  {

      auto u =q.front();
     // cout<<u.first<<" "<<u.second<<endl;
     q.pop();
     if(m[u]==1)
       continue;

     if ((u.first > a || u.second > b || u.first < 0 || u.second < 0))
       continue;
```

```
  // cout<<u.first<<" "<<u.second<<endl;

  m[{u.first,u.second}]=1;

  if(u.first == target || u.second==target)
  {
    isSolvable = true;

      printpath(mp,u);
        if (u.first == target) {
          if (u.second != 0)
              cout<<u.first<<" "<<0<<endl;
      }
      else {
          if (u.first != 0)
              cout<<0<<" "<<u.second<<endl;
      }
          return;
  }
  // completely fill the jug 2
  if(m[{u.first,b}]!=1)
  {q.push({u.first,b});
  mp[{u.first,b}]=u;}

  // completely fill the jug 1
  if(m[{a,u.second}]!=1)
 { q.push({a,u.second});
  mp[{a,u.second}]=u;}

//transfer jug 1 -> jug 2
  int d = b - u.second;
  if(u.first >= d)
  {
    int c = u.first - d;
    if(m[{c,b}]!=1)
      {q.push({c,b});
      mp[{c,b}]=u;}
  }
  else
  {
    int c = u.first + u.second;
```

```
      if(m[{0,c}]!=1)
        {q.push({0,c});
        mp[{0,c}]=u;}
    }
    //transfer jug 2 -> jug 1
    d = a - u.first;
    if(u.second >= d)
    {
      int c = u.second - d;
      if(m[{a,c}]!=1)
        {q.push({a,c});
        mp[{a,c}]=u;}
    }
    else
    {
      int c = u.first + u.second;
      if(m[{c,0}]!=1)
        {q.push({c,0});
        mp[{c,0}]=u;}
    }

    // empty the jug 2
    if(m[{u.first,0}]!=1)
      { q.push({u.first,0});
        mp[{u.first,0}]=u;}

    // empty the jug 1
    if(m[{0,u.second}]!=1)
      {q.push({0,u.second});
      mp[{0,u.second}]=u;}


  }
  if (!isSolvable)
      cout << "No solution";
}
```

Infix-prefix-postfix conversion

Pre-in-post conversion

//Egg Dropping Problem

```
int solveEggDrop(int n, int k) {
    if(memo[n][k] != -1) return memo[n][k];
    if (k==1 || k==0)  return k;
    if (n == 1) return k;

    int min = INT_MAX, x, res;

    for (x = 1; x <= k; x++) {
      res = max(
        solveEggDrop(n - 1, x - 1),
        solveEggDrop(n, k - x));
      if (res < min)
        min = res;
    }
    memo[n][k] = min+1;
    return min + 1;
  }
```

// Maximum sum of pairs with specific difference

```
int maxSumPairWithDifferenceLessThanK(int arr[], int N, int K){
    sort(arr, arr+N);
    int dp[N];
    dp[0] = 0;
    for (int i = 1; i < N; i++) {
        dp[i] = dp[i-1];
        if (arr[i] - arr[i-1] < K) {
            if (i >= 2)
                dp[i] = max(dp[i], dp[i-2] + arr[i] + arr[i-1]);
            else
                dp[i] = max(dp[i], arr[i] + arr[i-1]);
        }
    }
    return dp[N - 1];
}
```

JUMP GAME 2

```
//recursion
int jump(vector<int>& nums, int pos = 0) {
            if(pos >= size(nums) - 1) return 0;
            int minJumps = 10001;
            for(int j = 1; j <= nums[pos]; j++)
                        minJumps = min(minJumps, 1 + jump(nums, pos + j));
            return minJumps;
}
//dp
int solve(vector<int>& nums, vector<int>& dp, int pos) {
            if(pos >= size(nums) - 1) return 0;
```

```
        if(dp[pos] != 10001) return dp[pos];
        for(int j = 1; j <= nums[pos]; j++)
                dp[pos] = min(dp[pos], 1 + solve(nums, dp, pos + j));
        return dp[pos];
}
//tabulation
int jump(vector<int>& nums) {
        int n = size(nums);
        vector<int> dp(n, 10001);
        dp[n - 1] = 0;
        for(int i = n - 2; i >= 0; i--)
                for(int jumpLen = 1; jumpLen <= nums[i]; jumpLen++)
                        dp[i] = min(dp[i], 1 + dp[min(n - 1, i + jumpLen)]);
        return dp[0];
}
//Greedy BFS
int jump(vector<int>& nums) {
        int n = size(nums), i = 0, maxReachable = 0, lastJumpedPos = 0, jumps = 0;
        while(lastJumpedPos < n - 1) {
                maxReachable = max(maxReachable, i + nums[i]);
                if(i == lastJumpedPos) {
                        lastJumpedPos = maxReachable;
                        jumps++;
                }
                i++;
        }
        return jumps;
}
```

Minimum removals from array to make max –min <= K

```
int countRemovals(int a[], int i, int j, int k){
   if (i >= j)  return 0;
   else if ((a[j] - a[i]) <= k) return 0;
   else if (dp[i][j] != -1)  return dp[i][j];
   else if ((a[j] - a[i]) > k) {
      dp[i][j] = 1 + min(countRemovals(a, i + 1, j, k),
                  countRemovals(a, i, j - 1, k));
   }
   return dp[i][j];
}
```

// Returns number of ways to reach score n

```
int count(int n) {
   int table[n + 1], i;
   for(int j = 0; j < n + 1; j++)
      table[j] = 0;
   table[0] = 1;
   for (i = 3; i <= n; i++)
      table[i] += table[i - 3];

   for (i = 5; i <= n; i++)
```

```
        table[i] += table[i - 5];

    for (i = 10; i <= n; i++)
        table[i] += table[i - 10];

    return table[n];
}
```

Subset Sums

```
bool isSubsetSum(int set[], int n, int sum){
    bool subset[n + 1][sum + 1];
    for (int i = 0; i <= n; i++)   subset[i][0] = true;
    for (int i = 1; i <= sum; i++) subset[0][i] = false;

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= sum; j++)
            if (j < set[i - 1])
                subset[i][j] = subset[i - 1][j];
            else
                subset[i][j] = subset[i - 1][j] || subset[i - 1][j - set[i - 1]];
    return subset[n][sum];
}


bool findWinner(int x, int y, int n){
    int dp[n + 1];
    dp[0] = false;
    dp[1] = true;

    for (int i = 2; i <= n; i++) {
        if (i - 1 >= 0 and !dp[i - 1])
            dp[i] = true;
        else if (i - x >= 0 and !dp[i - x])
            dp[i] = true;
        else if (i - y >= 0 and !dp[i - y])
            dp[i] = true;
        else
            dp[i] = false;
    }
    return dp[n];
}
```

Count Derangements (Permutation such that no element appears in its original position)

```
countDer(n) = (n - 1) * [countDer(n - 1) + countDer(n - 2)]
```

Word Break Problem

```
//recursion
bool canBrk(int start, string& s, unordered_set<string>& wordDict) {
    int n = s.size();
```

```
    if(start == n) return 1;
    string sub;
    for(int i = start; i<n; i++) if(wordDict.count(sub+=s[i]) && canBrk(i+1,s,wordDict)) return 1;
    return 0;
}
//dp
bool canBrk(int start, string& s, unordered_set<string>& wordDict,vector<char>& mem) {
    int n = s.size();
    if(start == n) return 1;
    if(mem[start]!= -1) return mem[start];
    string sub;
    for(int i = start; i<n; i++) if(wordDict.count(sub+=s[i]) && canBrk(i+1,s,wordDict,mem)) return mem[start] =
1;
    return mem[start] = 0;
}
//dp iterative
bool wordBreak(string s, unordered_set<string>& wordDict) {
    int n = s.size();
    vector<bool> dp(n+1);
    dp[n]=1;
    for(int i=n-1;i>=0;i--) {
        string sub;
        for(int j=i;j<n;j++) if (dp[i] = wordDict.count(sub+=s[j]) && dp[j+1]) break;
    }
    return dp[0];
}
//bfs
bool wordBreak(string s, unordered_set<string>& wordDict) {
    queue<int> q({0});
    unordered_set<int> vstd;
    int n = s.size();
    while(!q.empty()) {
        int start = q.front();
        q.pop();
        if(vstd.count(start)) continue;
        vstd.insert(start);
        string sub;
        for(int i=start;i<n;i++)
            if(wordDict.count(sub+=s[i])) {
                q.push(i+1);
                if(i+1 == n) return 1;
            }
    }
    return 0;
}
```

Longest alternating subsequence

```
int zzis(int arr[], int n){
    int las[n][2];
    for(int i = 0; i < n; i++)
        las[i][0] = las[i][1] = 1;
    int res = 1;
    for(int i = 1; i < n; i++) {
```

```
        for(int j = 0; j < i; j++) {
            if (arr[j] < arr[i] &&
                las[i][0] < las[j][1] + 1)
                las[i][0] = las[j][1] + 1;

            if(arr[j] > arr[i] &&
               las[i][1] < las[j][0] + 1)
                las[i][1] = las[j][0] + 1;
        }
        if (res < max(las[i][0], las[i][1]))
            res = max(las[i][0], las[i][1]);
    }
    return res;
}
//way 2: o(n)
int LAS(int arr[], int n){
    int inc = 1, dec = 1;
    for (int i = 1; i < n; i++)
        if(arr[i]>arr[i-1])       inc = dec + 1;
        else if(arr[i]<arr[i-1])  dec = inc + 1;
    return max(inc, dec);
}
```

Weighted Job Scheduling

```
int latestNonConflict(Job arr[], int i){
    for (int j=i-1; j>=0; j--)
        if (arr[j].finish <= arr[i-1].start)
            return j;
    return -1;
}
int findMaxProfitRec(Job arr[], int n){
    if (n == 1) return arr[n-1].profit;

    int inclProf = arr[n-1].profit;
    int i = latestNonConflict(arr, n);
    if (i != -1)
      inclProf += findMaxProfitRec(arr, i+1);

    int exclProf = findMaxProfitRec(arr, n-1);

    return max(inclProf,  exclProf);
}

//dp iterative
int findMaxProfit(Job arr[], int n){
    sort(arr, arr + n, jobComparator);
    int* table = new int[n];
    table[0] = arr[0].profit;

    for (int i = 1; i < n; i++) {
        int inclProf = arr[i].profit;
        int l = latestNonConflict(arr, i);
        if (l != -1)
```

```
            inclProf += table[l];

        table[i] = max(inclProf, table[i - 1]);
    }

    int result = table[n - 1];
    delete[] table;

    return result;
}
```

Palindrome Partitioning Problem

```
    vector<vector<string>> partition(string s) {
        vector<vector<string> > res;
        vector<string> path;
        func(0, s, path, res);
        return res;
    }

    void func(int index, string s, vector<string> &path,
              vector<vector<string> > &res) {
        if(index == s.size()) {
            res.push_back(path);
            return;
        }
        for(int i = index; i < s.size(); ++i) {
            if(isPalindrome(s, index, i)) {
                path.push_back(s.substr(index, i - index + 1));
                func(i+1, s, path, res);
                path.pop_back();
            }
        }
    }

    bool isPalindrome(string s, int start, int end) {
        while(start <= end) {
            if(s[start++] != s[end--])
                return false;
        }
        return true;
    }
```

Letter Combinations of a Phone Number

```
    vector<string> mappings = {"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"}, ans;
    vector<string> letterCombinations(string digits) {
        if(digits == "") return {};
        string combination = "";
        helper(digits, 0, combination);
        return ans;
    }
```

```
    void helper(string &digits, int i, string &combi){
        if(i == size(digits)) {
            ans.push_back(combi);
            return;
        }
        for(auto &c : mappings[digits[i] - '2']){
            combi.push_back(c);            // add a character from mappings for current position,
            helper(digits, i + 1, combi);  // and recurse for next positions
            combi.pop_back();              // backtrack
        }
    }
```

Letter Combinations of a Phone Number-2 up,down,left, right

```
int solution(int n){
    vector<vector<int>> dp(n+1, vector<int>(10, 0));

    vector<vector<int>> data = {
//where i can go for each data
        {0, 8},
        {1, 2, 4},
        {1, 2, 3, 5},
        {2, 3, 6},
        {1, 4, 5, 7},
        {2, 4, 5, 6, 8},
        {3, 5, 6, 9},
        {4, 7, 8}
        {5, 7, 8, 9, 0}
    };

    for(int i=1;i<=n;i++){
        for(int j=0;j<=9;j++){
            if(i==1) dp[i][j] =1;
            else{
                for(int prev:data[j]){
                    dp[i][j] += dp[i-1][prev];
                }
            }
        }
    }

    int sum=0;
    for(int j=0;j<=9;j++) sum+=dp[n][j];
    return sum;
}
```

Phone keypad Bishop,knight only vector<vector<int>> data will get changed accordingly

Jump game

Stone game