STANDARD DP PROBLEMS (Interview Bit)

① Largest area of rectangle with permutation.

Given a binary grid A of size N×M consisting of 0's and 1's, find the area of the largest rectangle inside the grid such that all the cells inside the choose rectangle have 1 in them.

You are allowed to permute the column matrix i.e you can arrange each of the column in any order in the final grid.

Eg : A = $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ → $\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ → largest possible rectangle.

```
int solution ( int grid[ ][ ]){
    for(int i = 1, i< row; i++){
        for(int j = 0; j<col; j++){
            if (grid[i][j] == 1)
                grid[i][j] += grid[i-1][j];
        }
    }
    int ans = INT_MIN;
    for (int i = 0; i<row; i++){
        sort(all(grid[i]));
        for(int j = 0; j<col; j++){
            int area = (grid[i][j] *(col-j));
            ans = max(ans, area);
        }
    }
    return ans;
}
```

(2) Given three prime numbers A, B and C and an integer D. You need to find the first (smallest) distinct D integers which only have A, B, C or a combination of them as their prime factor.

Eg: A = 2, B = 3, C = 5, D = 5 | ans = [2, 3, 4, 5, 6]

```
vector<int> solve(int A, int B, int C, int D){
    vector<int> ans;
    ans.push_back(1);
    int x=0, y=0, z=0;
    while(ans.size()<=D){
        int mini = min({ans[x]*A, ans[y]*B, ans[z]*C});
        ans.push_back(mini);
        if(mini == ans[x]*A)
            x++;
        if(mini == ans[y]*B)
            y++;
        if(mini == ans[z]*C)
            z++;
    }
    ans.erase(ans.begin());
    return ans;
}
```

Using Set

```
set<int> heap;
heap.insert(A); heap.insert(B); heap.insert(C);
while(ans.size()<D){
    int small = *heap.begin();
    heap.erase(heap.begin());
```

```
            ans.push_back(small);
            heap.insert(small*A);
            heap.insert(small*B);
            heap.insert(small*C);
        }
    }
    return ans;
}
```

③ Longest Arithmetic Progression

```
(1) int solve(vector<int>&A){
    int n = A.size();
    if(n<3) return 2;
    int dp[n+1][n+1]; //filled with -1
    map<int,int> m;
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            dp[i][j]=2;
            int diff = A[j]-A[i];
            int need = 2A[i]-A[j];
            if(m.count(need)==0)
                continue;
            dp[i][j] = max(dp[i][j], 1+dp[m[need]][i];
        }
        m[a[i]] = i;
    }
    int ans = 2;
    for(int i=0; i<n; i++)
        for(int j=i+1; j<n; j++)
            ans = max(ans, dp[i][j]);
    return ans;
}
```

Let dp[i][j] be the length of longest A.P that ends in position 'j' i.e A[j] is the last element & A[i] is the second last element. Since we know two elements, let us calculate which number should be before A[i].

X, A[i], A[j]

$\therefore A[i]-X = A[j]-A[i]$

$\Rightarrow X = 2A[i]-A[j]$

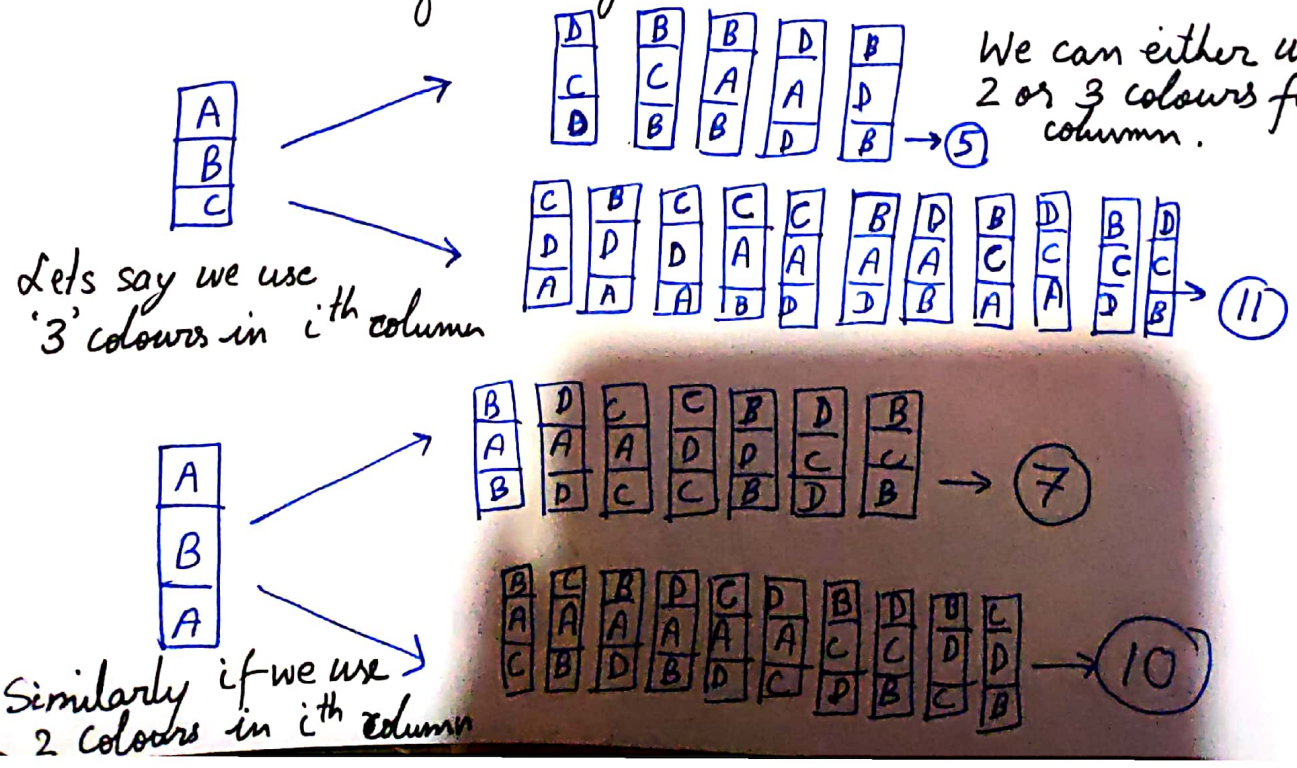
So, we can iterate over all $0 <= k < i$ and if A[k] == X then update dp[i][j] by 1+dp[k][i].

(2) int solution (vector<int> &A){

* Study this method.

```
int n = A.size();
vector <unordered_map<int, int>> dp(n+1);
if (n<=2)
  return n;
int ans = 2;
for(int i=0; i<n; i++){
    for(int j=i+1; j<n; j++){
        int diff = A[j]-A[i];
        if (dp[i].find(diff) ! dp[i].end())
            dp[j][diff] = 1+ dp[i][diff];
        else
            dp[j][diff] = 2;
        ans = max(ans, dp[j][diff]);
    }
}
return ans;
}
```

(4) Ways to colour a 3×N grid with 4 colours, so that no two adjacent grid have the same colour.



$$\therefore \begin{cases} W(i+1) = 10\,Y(i) + 11\,W(i) \\ Y(i+1) = 7\,Y(i) + 5\,W(i) \end{cases}$$

For any given 'n', final answer = $W(n) + Y(n)$.

(5) Buy & Sell Stocks — (1)

Given an array of integers A of size N in which ith element is the price of the stock on day i. You can complete almost / B transactions. Find the maximum profit you can achieve.

Note: You may not engage in multiple transactions at the same time.

```
int solve (vector<int>& A, int B) {
    int n = A.size();
    B = min(n, B);
    int dp[B+1][n+1] = {0};
    for (int i=1; i<=B; i++) {
        int pd = INT_MIN;
        for (int j=1; j<n; j++) {
            Buying  // pd = max(pd, dp[i-1][j-1] - A[j-1]);
            Selling // dp[i][j] = max(dp[i][j-1], pd + A[j]);
        }
    }
    return dp[B][n-1];
}
```

⑥ Longest Valid Parenthesis substring

```
int solution (string A) {
    int n = A.length();
    if (n<=1)
        return 0;
    stack<int> indices;
    indices.push(-1);
    int last = -1; // index of last un-matched closing
    int ans = 0;          bracket.
    for (int i=0; i<n; i++) {
        if (A[i] == '(')
            indices.push(i);
        else {
            int y = indices.top();
            if (y == last) {
                last = i;
                indices.push(i);
            }
            else {
                indices.pop();
                y = indices.top();
            } ans = max(ans, i-y);
        }
    }
    return ans;
}
```

(7) **Best time to buy & sell stocks - III**

Say you have an array A, i^{th} element is the price of a given stock on day i. If you can complete at most 2 transactions, design an algorithm to maximise the profit.

Refer to (5)

```
int solution ( vector<int>&A ){
    int n= A.size();
    int dp[n+1][3][2]={ };
    // dp[n][i][j] - nth day
                     i = ith transaction
                     j = 0 means buy, 1 means sell.

    dp[0][1][0] = INT_MAX;
    dp[0][2][0] = INT_MAX;

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

        dp[i][1][0] = min(dp[i-1][1][0], A[i-1])
        dp[i][1][1] = max(dp[i-1][1][1], A[i-1]-dp[i-1][1][0])

        dp[i][2][0] = min(dp[i-1][2][0], A[i-1]-dp[i-1][1][1]);
        dp[i][2][1] = max(dp[i-1][2][1], A[i-1]-dp[i-1][2][0])
    }

    return dp[n][2][1];
}
```

⑧ Kingdom War

A kingdom is a N×M grid each cell has a value. The strength of any cell on a row is greater than or equal to the strength above it & the strength of any village on column is greater than or equal to strength of every column to its left. Find the largest sum sub-matrix.

```
int solve (vector<vector<int>> &A){
    int ans = INT_MIN;
    int dp[n+1][m+1] = {0};
    for(int i=n-1; i>=0; i--){
        for(int j=m-1; j>=0; j--){
            dp[i][j] = A[i][j] + dp[i+1][j] + dp[i][j+1]
                       - dp[i+1][j+1];
            ans = max(dp[i][j], ans);
        }
    }
    return ans;
}
```

⑨ ~~Max~~ Rectangle in Binary Matrix (Maintain decreasing stack)

Given a 2D binary matrix filled with 0's and 1's find the largest rectangle containing all ones & return its area.

```
int solution (vector<vector<int>> &A){
    int n = A.size();
    int m = A[0].size();
    vector<int> B(m,-1);
```

```
for(int i=0;i<n;i++)
{
    vector<int> l(m,-1);
    vector<int> r(m,m);
    for(int j=0;j<m;j++)
        if(A[i][j]==0)
            B[j]=i;

    stack<int> s;
    for(int j=0;j<m;j++){
        while(!s.empty() & B[s.top()]<=B[j])
            s.pop();
        if(!s.empty())
            l[j]=s.top();
        s.push(j);
    }
    s=stack<int>();
    for(int j=m-1;j>=0;j--){
        while(!s.empty() & B[s.top()]<=B[j])
            s.pop();
        if(!s.empty())
            r[j]=s.top();
        s.push(j);
    }
    for(int j=0;j<m;j++)
        ans=max(ans,(r[j]-l[j]-1)*(i-B[j]));
}
return ans;
}
```

(10) Sub-matrices with zero sum

```
int helper (vector<int>&B){
    int n = B.size();
    vector<int> A(n);
    A[0] = B[0];
    for(int i=1;i<n;i++)
        A[i] = B[i] + A[i-1];
    int ans;
    unordered_map<int, int> freq;
    freq[0]=1;
    for(int i=0; i<n; i++){
        ans += freq[A[i]];
        freq[A[i]]++;
    }
    return ans;
}

int Solution (vector<vector<int>>&A){
    int row = A.size();
    int col = A[0].size();
    for(int base=0; base<row; base++){
        vector<int> temp(col, 0);
        for(int i=base; i<row; i++){
            for(int j=0;j<col;j++)
                temp[j] += A[i][j];
            ans += helper(temp);
        }
    }
    return ans;
}
```

(11) Maximum product sub-array

```
int maxproduct (vector<int> & arr) {
    int maxCount = arr[0];
    int minCount = arr[0];
    int ans = arr[0];
    for(int i=1; i<Arr.size(); i++) {
        if(arr[i]<0)
            swap (maxCount, minCount);
        maxCount = max(arr[i], maxCount * arr[i]);
        minCount = min (arr[i], minCount * arr[i]);
        ans = max(ans, maxCount);
    }
return ans;
```

(12) Flip array : Given an array of positive elements, you have flip the sign of some of its elements such that the resultant sum of elements of array should be as close to zero as possible. Return the minimum number of elements whose signs needs to be flipped.

```
int Solution (vector<int>& A) {
    int sum = 0;
    for (int i=0; i< A.size(); i++) sum + = A[i];
    sum/=2;
    int dp [sum+1] = {INT_MAX};
    dp[0] = 0;
    for (int i = 0; i < A.size(); i++){
        for (int j = sum; j >= A[i]; j --){
            if (dp[j-A[i]] != INT_MAX)
                dp[j] = min(dp[j], 1+ dp[j-A[i]]);
    } }
```

```
for (int i = sum; i >= 0; i--)
    if (dp[i] != INT_MAX)
        return dp[i];
}
```

(13) **Equal average partition** : Check if its possible to partition the array into two equal subsets such that average of both of them are equal.

Let sum of all elements = S & size = n.

Subset 1 S_1, n_1 — Subset 2 S_2, n_2 $\Rightarrow$ $S_1 + S_2 = S$, $n_1 + n_2 = n$

$$\therefore \quad \frac{S_1}{n_1} = \frac{S_2}{n_2}$$

$$\Rightarrow \frac{S_1}{n_1} = \frac{S - S_1}{n - n_1}$$

$$\Rightarrow S_1 n - S_1 n_1 = S n_1 - S_1 n_1$$

$$\Rightarrow \boxed{S_1 = \left(\frac{S}{n}\right) \times n_1}$$