Dynamic Programming

#) Edit Distance

Given 2 strs s and t. Min. number of operations that need to be done to convert s to t.

① Insert ② Remove ③ Replace.

— x —

s = " $\underbrace{\text{--- tts}}_{m}$ "   Insert 'b' in s.   Recur for (m, n-1)

t = " $\underbrace{\text{--- ttb}}_{n}$ "   Replace 'b' in s   Recur for (m-1, n-1)

Remove 'b' in s   Recur for (m-1, n).

Code

```
        m                 n
int dp[s.length()+1] [t.length+1]

for(int i=0 ; i<m+1 ; i++){
    for (int j=0; j<n+1; j++){
  →     if(i==0)        // s. empty → Insert all chars of t.
            dp[i][j] = j;

  →     else if (j==0)   // Remove all chars from s as t is empty.
            dp[i][j] = i;

  →     else if (s[i-1] == t[j-1])    //No ops.
            dp[i][j] = dp[i-1][j-1];

  →     else
            dp[i][j] = 1+ min(dp[i-1][j-1], min(dp[i][j-1], dp[i-1][j]))
    }
}

return dp[m][n];
```

Maximize Cut Segments

Cut into 3 segments x, y or z. Total no. of cut seg. must be maximum

I/P :- N = 4
x = 2, y = 1, z = 1.

O/P :- 4

① Make sure after cut of particular length rod length is >= 0. $(n-x) \geq 0$ or $(n-y) \geq 0$

② also make sure the cut is are going to make is available
dp [n - x] != (-1).

Code

```
vector<int> dp (n+1, -1);
dp[0] = 0;

for (int i = 1; i < (n+1); i++) {
    if (i-x >= 0 && dp[i-x] != -1)
        dp[i] = max(dp[i], 1 + dp[i-x]);

    // 2 more if condit^n for y and z
}

return (dp[n] <= 0) ? 0 : dp[n];
```

Maximum size Sq.-Submatrix of 1's

I/P :-
$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

O/P :- 2

① ~~First~~ last row
② last col
} keep mat[i][j]

else
1+ min(down, right, diagonally down);

Code

```
int dp[n][m]; int ans = 0;
if (n==1){
    for(j=0; j<m; j++)   ans = max(ans, mat[n-1][j]);
}
else if (m==1){
    for(i=0; i<n; i++)   ans = max(ans, mat[i][m-1]);
}
else{
    for (int i=n-1; i>=0; i--){
        for (int j=m-1; j>=0; j--){
            if( i==(n-1) && j==(m-1))
                dp[i][j] == mat[i][j];
            else if (i==(n-1))
                dp[i][j] = mat[i][j];
            else if (j==(m-1))
                dp[i][j] = mat[i][j];
            else{
                dp[i][j] = 1+ min(dp[i][j+1], min(dp[i+1][j+1], dp[i+1][j]
            ans = max(ans, dp[i][j]);
            }
        }
    }
} return ans;
```

Minimum Path Sum

Movement to solve from bottom to top.

Choose b/w $\min\begin{pmatrix} dp[i+1][j], \\ dp[i][j+1] \end{pmatrix}$ → down, → right

Code

```
int dp[n][m];
for(int i=n-1; i>=0; i--){
    for(int j=m-1; j>=0; j--){
        if(i==(n-1) && j==(m-1))
            dp[i][j] = arr[i][j];
        else if (i==(n-1))      // Movement right (last Row)
            dp[i][j] = dp[i][j+1] + arr[i][j];
        else if (j==(m-1))      // Movement down (last col)
            dp[i][j] = dp[i+1][j] + arr[i][j]
        else
            dp[i][j] = min(dp[i+1][j], dp[i][j+1])
                        + arr[i][j];
    }
}
return dp[0][0];
}
```

#) Optimal Strategy for a game

Code

```
int dp[1001][1001];
long long solve(int *arr, int i, int j){
    if(i > j)
        return 0;
    if(dp[i][j] != -1)
        return dp[i][j];

    // If I take i; opponent chooses from (i+1 -> j)
    Opponent tries to take val that makes opt for me min
    If opponent takes (i+1) -> I have (i+2) -> j
    If opponent takes (j) -> I have (i+1) -> (j-1)
    int val1 = arr[i] + min( solve(arr, i+2, j),
                             solve(arr, i+1, j-1));

    // If I take j; opponent chooses from (i, j-1)
    If opponent takes i; I choose from (i+1, j-1)
    If opponent takes j-1; I choose from (i, j-2)
    int val2 = arr[j] + min( solve(arr, i+1, j-1),
                             solve(arr, i, j-2));
    return dp[i][j] = max(val1, val2);
}
long long maxAmnt(int arr[], int n){
    dp[n][n]
    memset(dp, -1, sizeof(dp));
    return solve(arr, 0, n-1);
}
```

#) Maximum Profit from sales of wine

Each yr you can see only the last and first wine.
On y^{th} year; i^{th} wine will be $Y * P_i$.

Code

```
int maxProfit (int price[], int s, int e, int yr){
    if (dp[s][e] != -1)
        return dp[s][e];
    if (s == e)
        return yr * price[s];

    int left = (price[s] * yr) + maxProfit(price, s+1, e, yr+1);
    int right = (price[e] * yr) + maxProfit(price, s, e-1, yr+1);

    return dp[s][e] = max(left, right);
}

int wine (int price[], int n){
    dp[n][n];
    memset(dp, -1, sizeof(dp))
    return maxProfit(price, 0, n-1, 1);
}
```

Maximum Sum Rectangle in a 2D matrix

Application of Kadane's algorithm in 2D-form.

(i=0; i<R)
(j=i, j<R)
(k=0; k<C)

Code

```
int maxSumRectangle(int R, int C, v<v<int>> M){
    int max = INT_MIN;
    for(int i=0; i<R; i++){
        vector<int> v(C,0);

        // Explore all possible rectangles.
        for(int j=i; j<R; j++){
            for(int k=0; k<C; k++){
                v[k] += M[j][k];
            }

            max = max(max, kadane(v,C));
        }
    }
    return max;
}
```

Regular Expression matching

① For '.' ↖ ② For pc == sc ↖

③ for '*'

1st Col :- look 2 up. (i-2).

else 2 look 2 up comp. if p[i-2] == sc || p[i-2] == '.'

dp[i][j] = dp[i-2][j].

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

<u>Code</u>

```
bool dp[n+1][m+1];          n → pattern.length();
                            m → string.length();
for(i=0; i<n+1; i++){
  for(int j=0; j<m+1; j++){
    if(i==0 && j==0)          // first cell
       dp[i][j] = true;
    elif (i==0)               // first row
       dp[i][j] = false;
    else if (j==0){           // first column.
       char pc = p[i-1];
       if (pc=="*")
          dp[i][j] = dp[i-2][j];
       else
          dp[i][j] = false;
    }
    else{                     // Other.
       char pc = p[i-1];
       char sc = s[j-1];

       if (pc=='*'){
          dp[i][j] == dp [i-2][j];
          char pcsl = dp[i-2];
          if (pcsl ='.' || pcsl == sc)
             dp[i][j] = dp[i][j] || dp [i][j-1];
       }
```

```
else if (pc=='.')
    dp[i][j] = dp[i-1][j-1];

else if (pc==sc)
    dp[i][j] = dp[i-1][j-1];
else
    dp[i][j] = false;
  }
 }
}
return dp[n][m];
}
```

Wild Card Pattern Matching

Solve from down to top.

① if '?' ↘ (i+1) (j+1) ② pc == sc ↘

③ if pc == '*'

last col → dp[i][j] = dp[i+1][j].

else

→ dp[i][j] = dp[i+1][j] || dp[i][j+1];

Code

n → pattern.length();
m → string.length();

```
bool dp[n+1][m+1].
for(i=n; i>=0; i--){
  for(j=m; j>=0; j--){
    if(i==n && j==m)        // last cell
      dp[i][j] = true;
    else if (i==n)          // last Row
      dp[i][j] = false;
    else if(j==m){          // last Col.
      if(p[i] == '*')   dp[i][j] = dp[i+1][j];
      else   dp[i][j] = false;
    }
    else{
      char pc = p[i]; char sc = s[j];
      if (pc == '*')
        dp[i][j] = dp[i][j+1] || dp[i+1][j];
      else if(pc == '?')
        dp[i][j] = dp[i+1][j+1];
      else if (pc == sc)
        dp[i][j] = dp[i+1][j+1];
      else
        dp[i][j] = false;
    }
} }
return dp[0][0];
```

Scanned with CamScanner

Matrix Chain Multiplication

I/P:- {40, 20, 30, 10, 30} O/P:- 26000.

Code O(N^3)

```
int dp[101][101];

int solve(int * arr, int i, int j){
    if(i>=j)
      return 0;
    if(dp[i][j] != -1)
        return dp[i][j];

    int mn= INT_MAX;
    for(int k= i; k<j; k++){
        int tempAns= solve(arr, i, k) + solve(arr
                   + arr[i-1] * arr[k] * arr[j]
                   req. to join individual component.
        if(tempAns < mn)
            mn= tempAns;
    }
    return dp[i][j] = mn;
}

int MCM(int * arr, int n){
    memset(dp, -1, sizeof(dp));
    int i=1, j=n-1;
    return solve(arr, i, j);
}
```

Dungeon Game

Code

```
int n = dungeon.size();
int m = dugeon[0].size();

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

for(int i=n-1; i>=0; i--){
    for(int j=m-1; j>=0; j--){
        if(i==(n-1) && j==(m-1))
            dp[i][j] = min(0, dugeon[i][j]);
        else if (i==(n-1))
            dp[i][j] = min(0, dp[i][j+1]+ dungeon[i][j]);

        else if (j==(m-1))
            dp[i][j] = min(0, dp[i+1][j] + dungeon[i][j]);

        else
            dp[i][j] = min(0, max(dp[i+1][j], dp[i][j+1]) + dungeon[i][j]);
    }
}

return abs(dp[0][0]) + 1;
```

Unique Paths - II (LeetCode - 63)

Dynamic Prog.

Obstacles marked with '1';

Code

```
int uniquePaths(vector<vector<int>> &obstacleGrid){
    int n= obstacleGrid.size();
    int m = obstacleGrid[0].size();
    int dp[n][m];    memset(dp, 0, sizeof(dp));

    //1st col.→ If you encounter a obstacle
    No way to reach cells below.
    for(int i=0; i<n; i++){
        if (obstacleGrid[i][0] == 1)
            break;
        else
            dp[i][0] = 1;
    }

    // 1st row;
    for(int j=0; j<m; j++){
        if (obstacleGrid[0][j] == 1)
            break;
        else
            dp[0][j] = 1;
    }

    for(int i=1; i<n; i++){
        for (int j=1; j<m; j++){
            if (obstacleGrid[i][j] == 1)
                dp[i][j] = 0;
            else
                dp[i][j] = dp[i-1][j] + dp[i][j-1];
        }
    }

    return dp[n-1][m-1];
}
```