DYNAMIC PROGRAMMING

Handwritten Notes of Striver(TUF) Playlist

by: Aashish Kumar Nayak

NIT Srinagar

AASHISH KUMAR NAYAK

aashishkumar.nayak

DP (Dynamic programming)

1. DP1. Introduction to Dynamic programming | Memoization | Tabulation | Space optimization Techniques
2. DP2. climbing Stairs | Learn How to write 1D Recurrance Relations
3. DP3. Frog Jump | Dynamic programming | Learn to write 1D DP
4. DP4. Frog Jump with K distance | Lecture 3 follow up question
5. DP5. Maximum sum of Non-adjacent elements | House Robber | 1-D | DP on subsequences
6. DP6. House Robber 2 | 1D DP | DP on Subsequences
7. DP7. Ninja's Training | Must for 2D concepts | 2D DP
8. DP8. Grid unique path | Learn everything about DP on Grids | All Techniques
9. DP9. unique paths 2 | DP on Grid with Maze obstacles
10. DP10. Minimum path sum in Grid | DP on Grids | Asked in Microsoft
11. DP11. Triangle | fixed starting point and variable ending point | DP on Grids
12. DP12. Minimum/Maximum falling path sum | Variable starting and ending point | DP on Grids
13. DP13. Cherry pickup II | 3D DP made Easy | DP on Grids
14. DP14. Subset sum Equals to Target | Identify DP on subsequences and ways to solve them
15. partion Equal subset sum | DP on subsequences
16. Partion A set into Two subsets with minimum Absolute sum Difference | DP on subsequences
17. count Subsets with sum K | DP on subsequences
18. Count partions with Given difference | DP on Subsequences
19. 0/1 Knapsack | Recursion to strivers array space optimization approach | DP on subsequences
20. Minimum coins | DP on Subsequences | Infinite Supplies pattern
21. Target Sum | DP on subsequences
22. DP22. Coin Change 2 | Infinite supplies problem | DP on subsequences
23. DP23. Unbounded Knapsack | 1-D Array space optimised Approach
24. DP24. Rod cutting problem | 1-D Array space optimised approach
25. DP25. Longest Common subsequences | Top-down | Bottom-up | Space optimised | DP on String
26. DP26. print longest Common subsequences | DP on string

27. DP27. Longest Common Substring | DP on string
28. DP28. Longest palindromic subsequences |
29. DP29. Minimum insertions to make string palindrome
30. DP30. Minimum insertions | Deletions to cover String A to string B
31. DP 31. Shortest common subsequences | DP on strings
32. DP32. Distinct subsequences | 1D Array Optimization Technique
33. DP 33. Edit distance | Recursive to 1D Array optimised solution
34. DP34. Wildcard matching | Recursive to 1D Array optimisation
35. DP35. Best time to Buy and sell stocks | DP on stocks
36. DP36. Buy and sell stocks - II | Recursion to space optimisation
37. DP37. Buy and sell stocks - III | Recursion to space optimisation
38. DP 38. Buy and sell stocks - IV | Recursion to space optimisation
39. DP39. Buy and sell stocks with cooldown | Recursion to space optimization
40. DP40. Buy and Sell stocks with Transaction fee | Recursion to space optimis-ation
41. DP41. Longest increasing subsequence | Memoization
42. DP42. Printing Longest increasing subsequences | Tabulation | Algorithm
43. DP43. Longest increasing subsequence | Binary search | Intution
44. DP44. Largest divisible subset | Longest increasing subsequences
45. DP45. Longest String chain | Longest increasing subsequence | LIS
46. DP46. Longest Bfot Bitonic subsequence | LIS
47. DP47. Number of Longest increasing subsequences
48. DP48. Matrix chain Multiplication | MCM | partition DP starts
49. DP49. Matrix Chain Multiplication | MCM | Bottom-up | Tabulation
50. DP50. Minimum cost to cut the stick |
51. DP51. Burst Ballons | Partion DP |
52. DP52. Evaluate Boolean Expression to True | partion DP
53. DP53. palindrome partitioning - II | Front partition
54. DP54. partition array for Maximum sum | Front Partition
55. DP55. Maximum Rectangle area with all 1's | DP on Rectangles
56. DP56. Count square submatrices with All ones | DP on Rectangles

DP (Dynamic programming)

PAGE NO.:
DATE: / /

Those who cannot remember the past are condemned to repeat it.

i) Tabulation — Bottom up
ii) Memoization — Top-down

Memoization → Tabulation
↳ TL ↳ TC
SC SC → final soln

Pre requisite
Recursion playlist
1st 10 vdos, 6th & 7th vdo

(1) Fibonacci No.

0 1 1 2 3 5 8 13 21
0 1 2 3 4 5 6 7

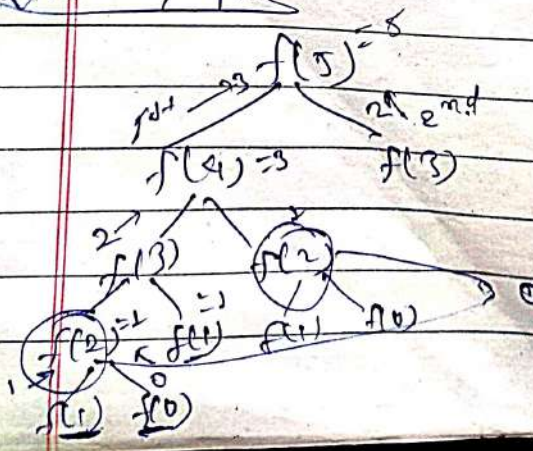
$f(n) = f(n-1) + f(n-2)$

$f(4) =$

F(n) Recursion

```
{
  if(n<=1)
    return n;
  return f(n-1) + f(n-2);
}
```

Recursion tree



f(4) ⟶
f(3) ⟶

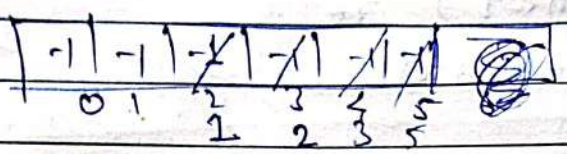
overlapping sub problem

Memoization

@ Aashish kumar Nayak

PAGE NO.:
DATE: / /

Memorization) → tend to store the value of sub problems in some map/ table.



initialy = -1

dp[n+1]

```
f(n)
{
  if (n<=1)
    return n;
  if(dp[n] != -1)
    return dp[n];
  return f(n-1) + f(n-2);   (1)
}
```

① convert Recursion into Memorization.

②

dp[n]

Recursion → DP

3 steps

declearse a dp array
size of sub problem i.e. n

step 1 :- storing the answer which is being computed for every subproblem.

step 2 :- checking if the subproblem has been previously solved. if it is previously solved then the value will not be -1.

step 3 :-

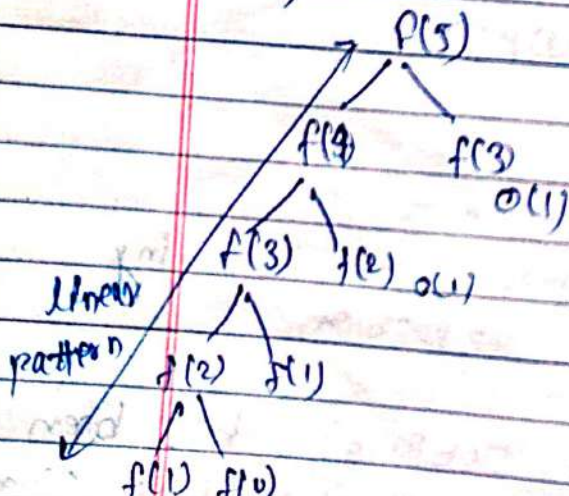
@Aashish kumar Nayak

```
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);
}
int main()
{
    int n;
    cin>>n;
    vector<int> dp(n+1,-1);
    cout<< f(n, dp);
    return 0;
}
```

n+1



TC = O(N)

SC = O(N) + O(N)

@Aashish kumar Nagar

PAGE NO.:
DATE: / /

Recursion → Tabulation (Bottom - Up)
(Top down)

Base case to the Required

Answer require to base case

dp[n+1]

dp[0] = 0 dp[1] = 1

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

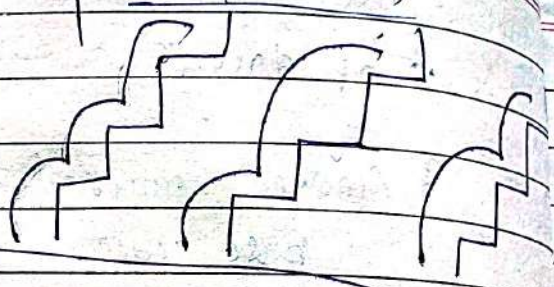
Best solution (space optimization)

```
int main()
{
  int n;
  cin>>n;
  int prev2 = 0;
  int prev1 = 1;
  for(int i=2; i<=n; i++)
    {
      int curr = prev + prev2
      prev2 = prev;
      prev = curr;
    }
  cout<< prev;
```

Intuitive solution

@Ashish Kumar Nayak

Climbing Stairs problem | 1D problem



$n = 3$

understand a DP problem :-

1. concept of try all possible way
   - count
   - best way
2. count the total no. of ways.
3. Optimal solution (min, max)

Then try to apply recursion

JWP.

all possible ways

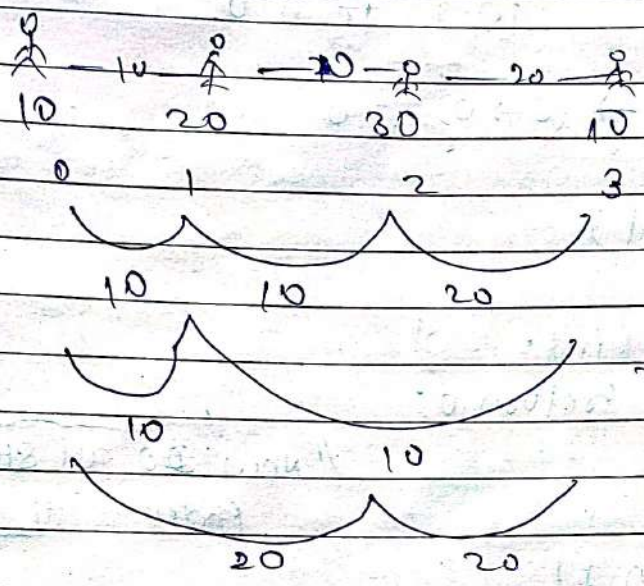
Shortcut

1. Try to represent the problem in terms of index.
2. Do all possible stuffs on that index according to the problem statement.
3. If question say count all ways then sum of all stuffs.
4. min (of all stuffs) → find min

@Aashish Kumar Nayak

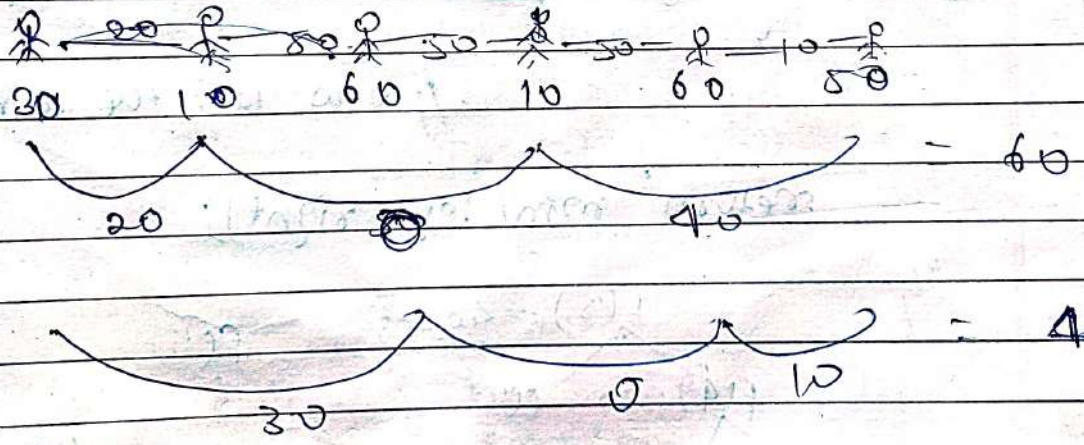
PAGE NO.:
DATE: / /

Frog JUMP problem :-



All possible ways, best ways, Means Recursion / DP

Why a greedy algo does not works ?



Greedy fails here, so we have to try Recursion.

Recursion → try all possible ways.

↓

(min energy)

Step 1 : index

Step 2 : Do all steps on that index acc. to question.

Steps : take @ the min (all steps).

@ Aashish Kumar Nayak

PAGE NO.:
DATE: / /

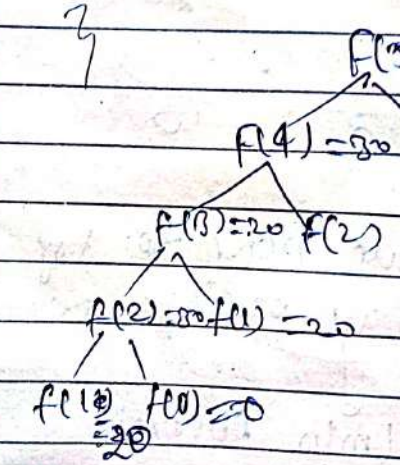
$f(n-1)$ → min energy required to reach $(n-1)$ from 0.

$f(0) \rightarrow (0 \rightarrow 0) = 0$

Recurrence code

```
f(ind)
{
  if(ind == 0)
    return 0;
                    // Now DO all stuffs on that
                       index +1 or -1 jump
  left = f(ind-1) + abs(a[ind] - a[ind-1]);

  if(ind > 1)
  {right = f(ind-2) + abs(a[ind] - a[ind-2]);
  }
                    // Now take the minimal
  return min(left, right);
}
```



f(5)
{
left = f(4) + (10) = 40
(30)
right = f(3) + (40) = 60
(20)
min(40, 60)
= 40 Ans
and so on

@ Aashish kumar Nayak

0	1	2	3	4	5
80	10	60	10	60	50

PAGE NO:
DATE: / /

f(1)
{
left = f(0) + abs(20)
right = INT_MAX —— ∵ ind ≯ 1 ⇒ false
return (20, INT_MAX)
}
∴ f(1) = 20

f(2):
{
left = f(1) + (50) : = 70 (f(1) → 20)
right = f(0) + (30) : = 30 (f(0) → 0)
return (70, 30)
}
∴ f(2) = 30

f(3)
{ left = f(2) + (50) : = 80 (f(2) → 30)
right = f(1) + (0) = 20 (f(1) → 20)
return (80, 20) ⇒ f(3) = 20
}

f(4)
{ left = f(3) + (50) = 70 (f(3) → 20)
right = f(2) + (0) = 30 (f(2) → 30)
return (70, 30)
}
f(4) = 30

f(5)
{ left = f(4) + (10) = 40 (f(4) → 30)
right = f(3) + (40) = 60 (f(3) → 20)
return (40, 60); f(5) = 40 Ans
}

PAGE NO.:
DATE: / /

How Convert Recurence into DP

Recurence → DP

Memorization → look at the parameter are Changing.

ind is changing

∵ Max size of ind is 5

∴ 5+1 = 6

dp[6]

Step 1 :- declare an array of size n+1

```
dp[n+1] := ;
```
— ①

Step 2 :- Before returning add it up. Store it and return.

```
return dp[ind] = min(left, right);
```
— ②

Step 3 :-

Check if it is previously computed or not.

```
if(dp[ind] != -1)
    return dp[ind];
```
— ③

TC → O(N)

SC → O(N) + O(N)

Recursion array

@Ashish Kumar Nayak

...ning LDP on subsequence

PAGE NO.:
DATE: / /

Lets code

```
int ff(int ind, vector<int> &height, vector<int> &dp)
{
    if(ind==0)
        return 0;
    if(dp[ind]!=-1) return dp[ind];
    int left = f(ind-1, height, dp) + abs(height[ind] - height[ind-1]);
    int right = INT_MAX;
    if(ind>1)
    {
        right = f(ind-2, height, dp) + abs(height[ind] - height[ind-2]);
    }
    return dp[ind] = min(left, right);
}
int frogjump(int n, vector<int> &height)
{
    vector<int> dp(n+1, -1);
    return f(n-1, height, dp)
}
```

Only some test case will pass & T.L.E.

Now optimized Memoization.

Step 1 :- declare dp
dp[n+1]

Step 2 :- store it and return (overlapping sub problem)
return dp[ind] = min(left, right);

Step 3 :- Check it if it is previously computed or not
if(dp[ind] != -1)
return dp[ind];

TC =
TC → O(N)
SC → O(N) + O(N)
recursion array dp

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Memoization (Top-down)

Tabulation (Bottom-up)

Tabulation method

int dp[n] → for zero based index

```
int frogJump(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(height[i] - height[i-1]);
        int ss = INT_MAX;
        if(i>1)
        {
            ss = dp[i-2] + abs(height[i] - height[i-2]);
        }
        dp[i] = min(fs, ss);
    }
    return dp[n-1];
}
```

TC → O(N)

SC → O(N) + O(N) (Recursion)

array dp — reduced

Space optimization

PAGE NO.:
DATE: / /

```
int frogJump(int n, vector<int> &height)
{
    int prev = 0;
    int prev2 = 0;

    for(int i = 1; i < n; i++)
    {
        int fs = prev + abs(height[i] - height[i-1]);
        int ss = INT_MAX;
        if(i > 1)
            ss = prev2 + abs(height[i] - height[i-2]);

        int curi = min(fs, ss);
        prev2 = prev;
        prev = curi;
    }
    return prev;
}
```

$T.C. = O(N)$

$S.C. = \cancel{O(N)}$ (array dp) $+ \cancel{O(N)}$ (Recursion)

$\therefore S.C. = O(1)$

both are reduced

Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Recurrence & Code

```
f(ind)
{
  if(ind == 0)
      return 0;

int minSteps = INT_MAX;
    for(int j=1; j<=K ; j++)
{
    if( ind-j >=0)
    {
    Jump = f(ind-j) + abs( a[ind] - a[ind-j]
    minSteps = min( min_steps, jump);
    }
```

Now Memoize the code

```
f(ind)
```

Tabulation

```
int dp[n];
dp[0] = 0;
for(int i=1; i<n; i++)
{
    for( j
int min_steps = INT_MAX;
for( j=1; j<=K; j++)
{ if(i-j>=0)
}
Jump = dp[i-1] + abs( a[i] - a[i-j]
Min steps = Min( Min_steps, jump);
}
```

dp[i] = min_steps;
}

print (dp[n-1]);

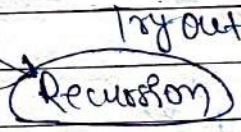
T.C → O(N×K)
S.C → O(N)

Can't optimize further as S.C - O(K) + O(N)

MAXIMUM SUM OF NON-ADJACENT ELEMENTS

Lets try out all subsequences with the given constraints

Try out all means Recursion.



↓

pick the one with maximum sum.

Print all subsequences

↳ pick/Non-pick

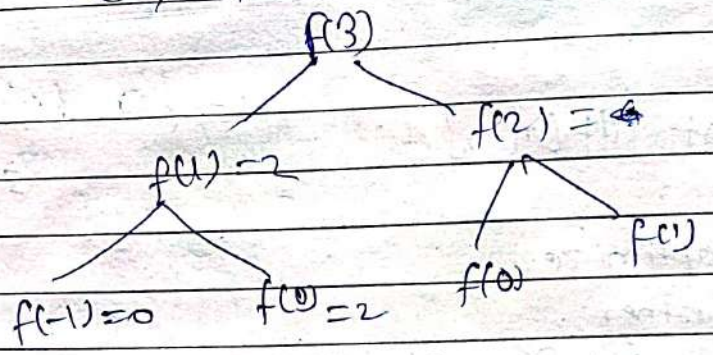
1. Index
2. Do stuffs
3. return the best you can get.

Pick subsequence with no adjacent element.

@Aashish Kumar Nayak

PAGE NO.:
DATE : / /

0 1 2 3
2, 1, 4, 9



f(1)
{
f(1) ⇒ sm[0--1]

Pick = a[1] + f(-1) = 1 + 0 = 1
Not Pick = 0 + f(0) = 0 + 2 = 2

Max(Pick, Not Pick) ⇒
∴ f(1) = 2

f(3)
{
Pick = a[3] + f(1) = 9 + 2 = 11
Not Pick = 0 + f(2) = 0 + 6 = 6
Max(Pick, Not Pick) ∴ f(3) = 11

f(2)
{ Pick = a[2] + f(0) = 4 + 2 = 6
Not Pick = 0 + f(1) = 0 + 2 = 2
Max(Pick, Not Pick) ∴ f(2) = 6

TC - O(N)
SC - O(N) + O(N)

@Ashish kumar Nayak

Leet code

Recurrence code

```
int f(int ind, vector<int> &nums, &dp)
{
  if(ind == 0)
      return nums[ind];
  if(ind < 0)
      return 0;
  int  if(dp[ind] != -1) return dp[ind];
  int pick = nums[ind] + f(ind-2, nums, dp);

  int not_pick = 0 + f(ind-1, nums, dp);
  return max(pick, not_pick);
}  return dp[ind] = max(pick, not_pick);
int Max_Sum_Non_Adjacent(vector<int> &nums)
{
    int n = nums.size();
    Vector<int> dp[n,-1];  -①
    return(n-1, nums, dp);
}
```

Now Memoized to save from TLE

@Ashish Kumar Nayak

PAGE NO.:
DATE: / /

Tabulation

```
int dp[0] = a[0];
int neg = 0;
for(int i = 1; i<n; i++)
{
  take = a[ind];
  Non take = 0 + ;
  if(i>1)
    take = take + dp[i-2];
  Non-take = 0 + dp[i-1];
  dp[i] = max( take, non take);
}
```

TC = O(N)
SC = O(N)

Space optimization :-

```
int prev = a[0]
int prev2 = 0;
for(i=1; i<n; i++)
{
  take = a[i];
  if(i>1)
    take = take + prev2;
  Non take = 0 + prev;
```

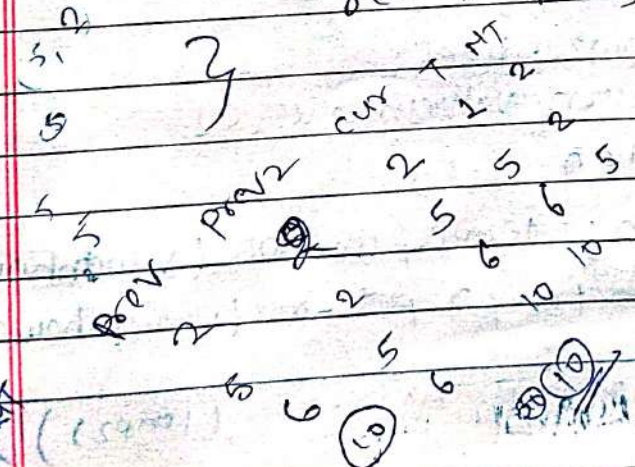
PAGE NO.:
DATE: / /

curi = max(take, non_take);
prev2 = prev;
prev = curi;
}
print (prev);

int Max_sum_Non_Adja(vector<int> &nums)
{
int n = nums.size();
int prev = nums[0];
int prev2 = 0;

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

int not_take = 0 + prev;
int curi = max(take, not_take);
prev2 = prev
prev = curi
}
return prev;
}



@Aashish kumar Nayak

@Aashish kumar Nayak

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

House Robber

```
int max(vector<int> &nums)
{
    int n = nums.size();
    int prev = nums[0];
    int prev2 = 0;
    int
    for(int i = 1; i < n; i++)
    {   int take = nums[i];
        if(i > 1)
            take = take + prev2;

        int not_take = 0 + prev;

        int curi = max(take, not_take);
        prev2 = prev;
        prev = curi;
    }

    return prev;
}

int houseRobber(vector<int> &valueInhouse)
{
    vector<int> temp1, temp2;
    int n = valueInhouse.size();
    if(n == 1) return valueInhouse[0];
    for(int i = 0; i < n; i++)
    {   if(i != 0) temp1.push_back(valueInhouse[i]);
        if(i != n-1) temp2.push_back(valueInhouse[i]);
    }
    return max(maximumNonAdj(temp1), maximumNonAdj(temp2));
}
```

NINJA TRAINING Problem

Whenever

Greedy fails

↓

Try all possible ways

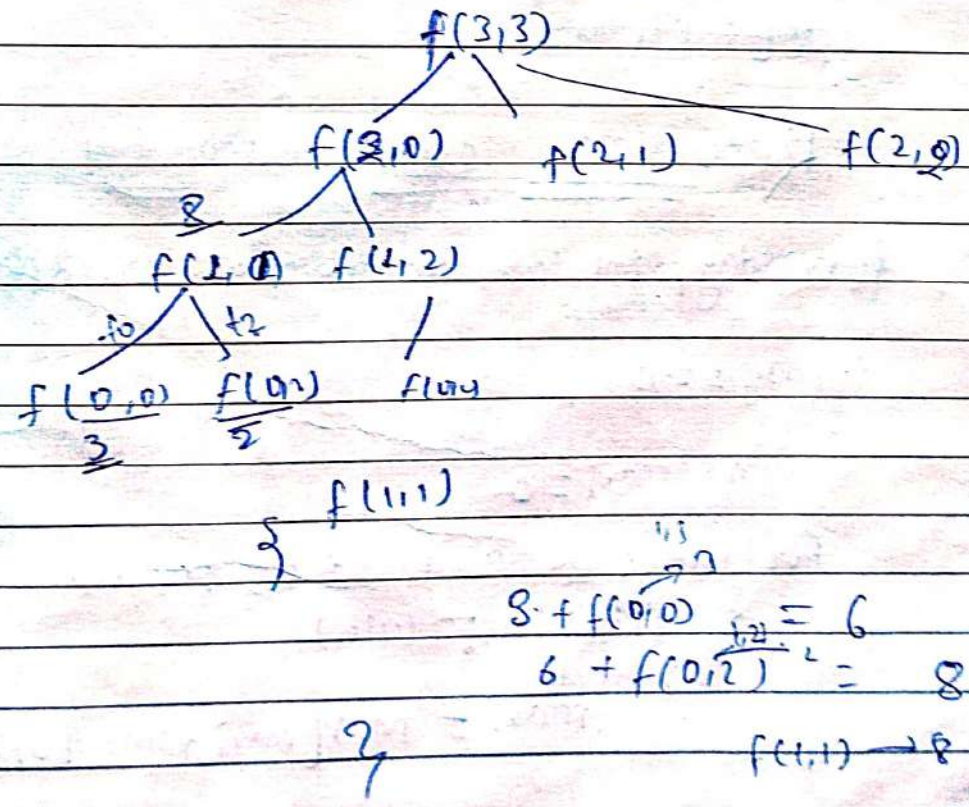
↓

Recursion

↓

1. index
2. DO Stuffs on that index
3. Find Max or Min

t_0	t_1	t_2	
2	1	3	d_0
(3)	4	6	d_1
10	1	6	d_2
8	3	7	d_3



@Aashish Kumar Nayak

@Aashish kumar Nayak

PAGE NO.:
DATE: / /

f(1,2)

f(0,0) f(0,1)

3

{ 3+3 = 6
4+3 = 7 }

f(2,0)

{ 1 + f(1,1) → 8
{ 6 + f(4,2) → 6 = 9, 12 → 12

f(2,1)

{

Overlapping sub problem

↓ Memoization

(changing parameter ind, last)

day → (N)

last → (0,1,2,3) → (4)

N X 4 dp size array

dp[ind][last]

Recursion code + memoized

```
int f(int day, int last, vector<vector<int>> &points, 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;
}

int ninjatraining(int n, vector<vector<int>> &points)
{
    vector<vector<int>> dp(n, vector<int>(4, -1));
    return f(n-1, 3, points, dp);
}
```

TC - O(N*4)*3
SC - O(N) + O(N*4)

Tabulation :-

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

@Ashish Kumar Nayak

@Ashish Kumar Nayak

PAGE NO.:
DATE: / /

Tab

Tabulation

```
int minimumTotal(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][0], points[0][1], points[0][2]);

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

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

TC - O(N*4*3)

SC - O(N*4)

Space optimization

```
int ninjatraining( int n, vector<vector<int>> &points)
{
    vector<int> prev(i,0);

    prev[0] = max[points[0][1], Points[0][2]);
    prev[1] = max( points[0][0], Points[0][2]);
    prev[2] = max ( point[0][0], point[0][1]);
    Prev[3] = max ( point[0][0], point[0][1], point T[0][2]);

for(int day=1 ; day<n ; day++)
    {
      vector<int> temp(i,0);
      for(int last = 0; last < 4 ; last++)
          {
              temp[last] = 0;

      For (int task = 0 ; task < 3; task++)
          {
            if(task != last)
              {
                temp[last] = Max( temp[last],
                points[day][last] + prev[task]);
              }
          }
    }
    prev = temp;
}
    return prev[3];
}
```

TC = O(N×4×3)
SC = O(4) ↓ constant

@Aashish kumar Nayak

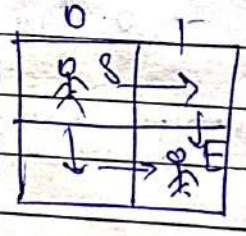
DP on Grids / 2D Matrix

- → Count paths
- → Count paths with obstacles
- ⇒ Min path sum
- ⇒ Max path sum
- → Triangle
- → 2 start points

6 problem

Total unique path

Unique path



(0,0) → (1,1)

2 ways

(0,0) → (m-1, n-1)

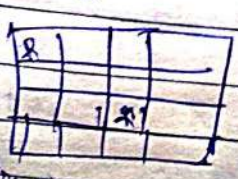
Try all possible ways

Means apply Recursion

How to write Recursion

1. Express in terms of index (i,j)
2. Explore all steps
3. Sum up all ways (min/max)

f(i,j) → no of unique ways (0,0) → (i,j)




```
f(i,j)
{
  Base case
  return 1 if reach destination
  return 0 if can't reach destination
}
```

@Aashish Kumar Nayak

@ Aashish Kumar Nayak

PAGE NO.:
DATE: / /

```
ip f(i,j)
{
    if( i==0 && j==0)
        return 1;
    if( i<0 || j<0)
        return 0;
    if(dp[i][j] != -1) return dp[i][j];
    up = f(i-1, j);
    left = f(i, j-1);
    return up+left;
}
```

Express

Base case

Explore

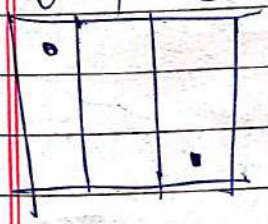
Sum up

dp[i][j] =

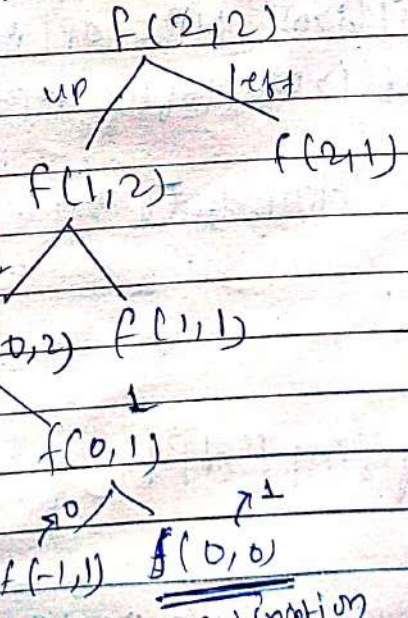
Recursion → DP

Memoization

overlapping sub problem.



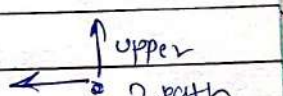
Let m = 3, n = 3



TC → O(N×m)

SC = O(n-1)+(m-1) ≈ O(N×m)

DP



TC → $(2^{m \times n})$

SC → O(path length)

(0,0) (0,1) (0,2) (0,3)
(1,3)
(2,3)
(3,3)

(m-1)+(n-1)

```
f(0,1)
{
    up = f(-1,1)   → 0
    left = (0,0)   → 1
    return 0+1     → 1
```

Memoization

PAGE NO.:
DATE : / /

Tabulation :-

```
dp[m][m];
if( dp[0][0] = 1

for( i=0 → m-1)
{   for(j=0 → n-1)
    {
    }
}
```

Memoization → Tabulation

1. Declare Base case
2. Express various states in for loop
3. copy the recurance & write.

```
if( i==0 && j==0)
    dp[0][0] = 1
else
{
}
if(i>0)  up = dp[i-1][j];
if(j>0)  left = dp[i][j-1];

    dp[i][j] = up + left;
}
}

print( dp[i][j]);
```

TC = O(N×M)
SC = O(N×M) + recursion

@Ashish kumar Nayak

PAGE NO.:
DATE: / /

→ If there is a previous row & previous column, we can space optimised it.

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

best way to optimised

```
dp[n] ⟶ 0
for(i=0 ⟶ m-1)
{ for(j=0 ⟶ n-1)
    {
      if(i==0 && j==0)
          dp[i][j] = 1;
      else {
          dp[i][j] = dp[i-1][j] + dp[i][j-1]
      }
    }
}
```

Code

```
int f(int i, int j, vector<vector<int>> &dp)
{
    if(i==0 && 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, dp);
    int left = f(i, j-1, dp);
    return dp[i][j] = up + left;
}
```

@Aashish Kumar Nayak

```
int uniquepath (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;
        int left =0;
        if(i>0) up
        up = dp[i-1][j];
        left = dp[i][j-1];
        dp[i][j] = up + left;
      }
    }
  }
  return dp[n-1][j-1];
}
```

TLE

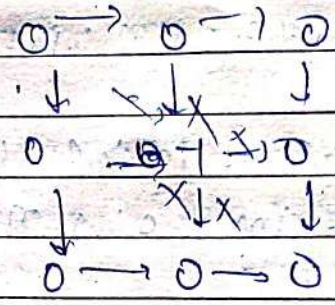
PAGE NO.:
DATE: / /

```
int uniquePaths(int m, int n)
{
    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;
                int left = 0;
                if(i>0)
                    up = prev[j];
                if(j>0)
                    left = cur[j-1];
                dp[i][j] = up + left;
            }
        }
        prev=cur;
    }
    return prev[n-1];
}
```

@Aashish Kumar Nayak

PAGE NO.:
DATE : / /

Unique paths 2 DP on Grid with Maze Obstacle




```
int f(i,j)
{
if(i>=0 && j>=0 && a[i][j]==-1)
    return 0;

if(i==0 && j==0)
    return 1;

if(i<0 || j<0)
    return 0;

if(dp[i][j]!=-1) return dp[i][j];
up= f(i-1,j);
left = f(i, j-1);

return up+left;
}  return dp[i][j] = up+left;
```

PAGE NO.:
DATE: / /

Tabulation :-

```
for(i=0 → m-1)
{
    for(j=0 → n-1)
    {
        if( i==0 && j==0)
            dp[i][j] = 1;
        else
if(i>0)   up = dp[i-1][j];
if(j>0)   left = dp[i][j-1]

        return up+left;

    }
```

Code recursion + Memoization

```
int mod = (int)(1e9+7);
int f(int i, int j, vector<vector<int>> &mat, vector<vector<int>> &dp)
{
    if(i>=0 && j>=0 && mat[i][j]==-1) return 0;
    if(i==0 && 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, mat, dp);
    int left = f(i, j-1, mat, dp);

    return (up + left)%mod;
return dp[i][j] = (up+left)%mod;
}
int mazeObstacles(int n, int m, vector<vector<int>> &mat)
{
    vector<vector<int>> dp(n, vector<int>(m,-1));
    return f(n-1, m-1, mat, dp);
}
```

@Aashish Kumar Nayak

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Tabulation :

```
int mod = (int)(1e9 +7);
{ int dp[n][m];
for(int i=0; i<n; i++)
{ For (int j=0; j<m; j++)
  { if(mat[i][i] == -1) dp[i][j] = 0;
  else if(i==0 && j==0) dp[i][j] = 1;
  else { int up=0, left=0;
       if(i>0) up = up[i-1][j];
       ff(j>0) left = dp[i][j-1];
       dp[i][j] = (up+left) % mod;
  }
}
return dp[n-1][m-1];
}
```

Space optimization

```
int mod =(int)(1e9 +7);

int mazeObstacle(int n, int m, vector<vector<int>> &mat)
{
  Vector<int> prev (m,0);
  for(int i=0; i<n; i++)
  { vector<int> cur(m,0);
    for(int j=0; j<m; j++)
    { if(mat[i][j] == -1)
        curr[j] = 0;
      else if(i==0 && j==0) curr[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) % mod;
}
}
prev = cur;
}
return prev[n-1];
}
```

MINIMUM PATH SUM IN GRID

	0	1	2
0	5	9	6
1	11	5	2

2 X 3

(0,0) → (0,2)

5 + 11 + 5 + 2 = 23 ↓↓

5 + 9 + 5 + 2 = 21 → Min

Why not follow greedy

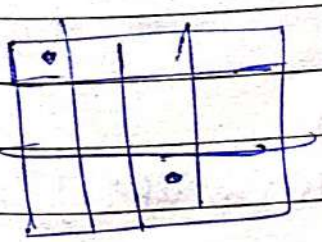
10	8	2
10	5	100
1	1	2

greedy = 10 + 8 + 2 + 100 ✗ +2

Min = 10 + 10 + 1 + 1 + 2 ✓

Try out all paths

↓ (Recursion) ↳ Tell me the path which has min cost



1.) index (i,j)

2.) Explore all Paths

3.) Take the min/~~max~~ path

dp[i,j]

(0,0) → (i,j)

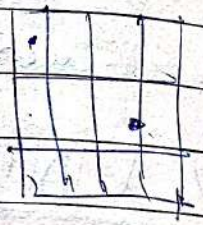
f(n-1, m-1) → (0,0) → (n-1, m-1)

@Aashish Kumar Nayak

f(i,j)
{
if(i==0 && j==0)
return a[0][0];

if(i<0 || j<0)
return INT_MAX;

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



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

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

(3) Recursion → Memorization
overlapping
sub problem

(4) dp[n][m] → -1

Code

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(left, up);
}
int minSumPath(vector<vector<int>> &grid)
{
    int n = grid.size();
    int n = grid[0].size();
vector<vector<int>> dp(n, vector<int>(n,-1));
    return f(n-1, m-1, grid, dp);
}
```

Tabulation :=

```
int minSumPath(vector<vector<int>> &grid)
}
    int n=grid.size();
    int 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 && 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;
```

@Aashish Kumar Nayak

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

    dp[i][j] = min(left, up);
  }
 }
}
```

Space optimisation

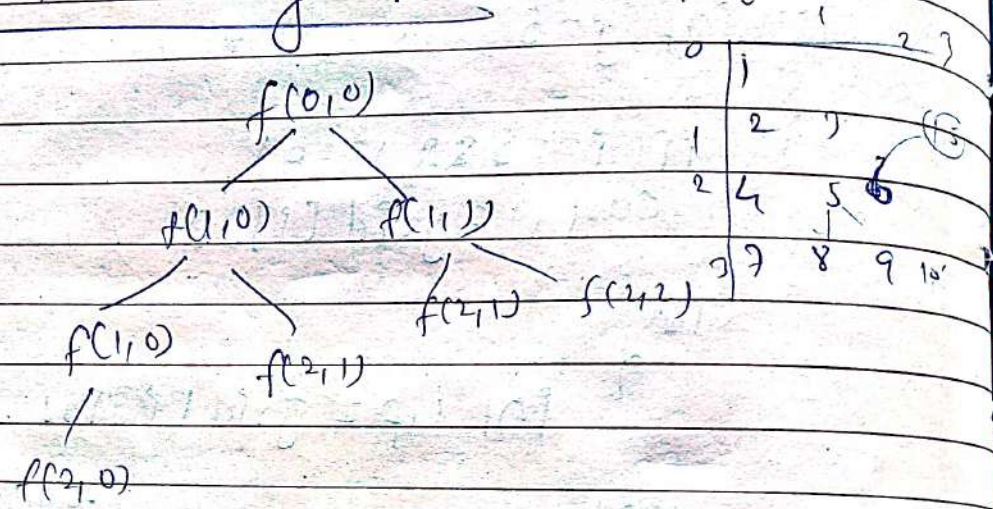
```
vector<int> cur(m,0);
for(int j=0; j<m; j++)
{
  if(i==0 && j==0)
    cur[j] = grid[i][j];
  else
  {
    int up = grid[i][j];
    if(i>0) up += prev[j];
    else up += 1e9;
    int left = grid[i][j];
    if(j>0) left += cur[j-1];
    else left += 1e9;
    cur[j] = min(left, up);
  }
}
prev = cur;
}
return prev[n-1];
}
```

@ Aashish Kumar Nayak

D-11

PAGE NO.:
DATE: / /

Triangle Fixed starting point and Variable Ending point




```
int f(int i, int j, vector<vector<int>> &triangle, int n,
      vector<vector<int>> dp)
{
    if(i == n-1)
        return triangle[n-1][j];

    if( dp[i][j] != -1)
        {
            return dp[i][j];
    int d = triangle[i][j] + f(i+1, j, triangle, n, dp);
    int dg = triangle[i][j] + f(i+1, j+1, triangle, n, dp);

    return dp[i][j] = min(d, dg);
}
int minimumTriangle(vector<vector<int>> &triangle, int n){
    vector<vector<int>> dp(n, vector<int>(n, -1));
    return f(0, 0, triangle, n);
}
```

TLE

@Aashish kumar Nayak

PAGE NO.:
DATE: / /

To avoid TLE ⇒ Tabulation

```
int minimumTotal(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];
}
```

Correct 100%

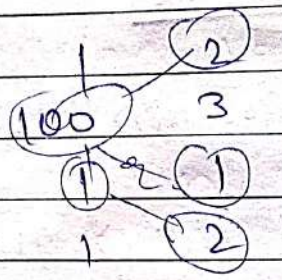
Space optimization :

```
int minimumTotal(vector<vector<int>>& triangle, int n) {
    vector<int> front(n,0), cur(n,0);
    for(int j = 0; j < n; j++)
    {
        front[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] + front[j];
            int dg = triangle[i][j] + front[j+1];
            cur[j] = min(d, dg);
        }
        front = cur;
    }
    return front[0];
}
```

both loop end

D-12

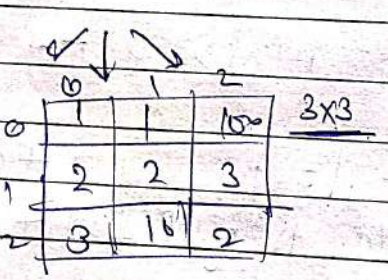
MINIMUM/MAXIMUM FALLING PATH SUM :-

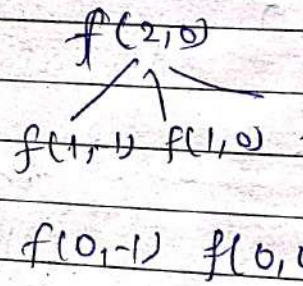


Greedy → fails (because of uniformity)

Greedy will pass only when elements are uniform.

We can not apply greedy where uniformity is not there.





f(1,0)

{

ld = 2 + f(0,-1) → -1eg

u = 2 + f(0,0)

rd = 2 + f(0,1)

Aashish Kumar Nayak

Recursion code

```
int f(int i, int j, vector<vector<int>> & matrix, vector<vector
                <int>> &dp)
{   if( j<0 || j>= matrix[0].size() )
        return -1eg;
    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, max(ld, rd));
}

int getmax( vector<vector<int>> &matrix )
{
    int n = matrix.size();
    int m = matrix[0].size();

    vector<vector<int>> dp(n, vector<int>(m, -1));
    int maxi = -1eg;
    for(int j=0; j<n; j++)
    {
        maxi = max( maxi, f(n-1, j, matrix, dp));
    }
    return maxi;
}
```

@Aashish Kumar Nayak

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Tabulation :-

```
int getmax(vector<vector<int>> &matrix)
{
    int n = matrix.size();
    int m = matrix[0].size();

    vector<vector<int>> dp(n, vector<int>(m, 0));   // prev(m,0), curr(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];          // prev[j]
            int ld = matrix[i][j];
            if(j-1>=0)
                ld += dp[i-1][j-1];                     // prev[j-1]
            else
                ld += -1e8;
            int rd = matrix[i][j];
            if(j+1<m)
                rd += dp[i-1][j+1];                     // prev[j+1]
            else rd += -1e8;
            dp[i][j] = max(u, max(ld, rd));             // curr[j]
        }
        // prev = curr;
    }
```

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

dp[n-1][j] → prev[j]

Space optimization code :-

```
int curr[m], prev[m];
```

Just replace dp[i][j] → prev[j]

& make → prev = curr;

and → return ~~prev~~.prev

in this case

~~return maxi = max(0, maxi);~~

return = max(maxi, prev);

@Aashish Kumar Nayak

DP-13

PAGE NO.:
DATE: / /

Cherry Pickup

Rules

1. Express everything in terms (i_1, j_1) & (i_2, j_2)
2. Explore all the paths ↙↓↘
3. Max sum.

fixed starting point

variable ending point

$(0,0)$ $(0, m-1)$ (At any index in the last row)

(All paths by Alice) + (All paths by Bob)

Recursion Recursion

Alice Together

$f(0,0, 0, m-1)$

Alice Bob

Base case

i) Out of Boundary

ii) last row case (Destination)

Boundary ca always write first.

$f(i_1, j_1, i_2, j_2)$

```
if( j1 < 0 || j1 >= m || j2 < 0 || j2 >= m)
    return -1e8;
```

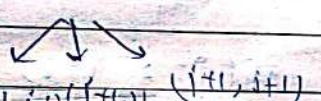
↙↓↘ Means they reach at same time.

PAGE NO.:
DATE: / /

```
if( i == n-1 )
{
  if (j1 == j2)
      return a[i][j1];
  else
      return a[i][j1] + a[i][j2];
}
```

Base condition

① Explore all the paths



In one step move Alice & Bob both

```
maxi = 0;
for(dj1 → -1 → +1)
{
  for dj2 → -1 → +1
  {
    maxi = max( maxi, f(i+1, j1+dj1, j2+dj2) );
  }
}
if(j1 == j2)
    return a[i][j1] + maxi;
else
    return a[i][j1] + a[i][j2] + maxi;
```

fail

out

TC → $3^n \times 3^n$ = exponential ms

SC → O(N) Auxiliary stack space

@Aashish Kumar Nayak

@Ashish Kumar Nayak

PAGE NO.:
DATE: / /

Recursion code

```
int f( int i, int j1, int j2, int r, int c,
 vector<vector<int>> &grid, vector<vector<vector<int>>> &dp)

{
  if (j1<0 || j2<0 || j1>=c || j2>=c)
  {
    return -1e8;
  }
  if(i==r-1)
  {
    if(j1==j2)
      return grid[i][j1];
    else return grid[i][j1] + grid[i][j2];
  }
  if(dp[i][j1][j2]!=-1)
  { return dp[i][j1][j2];
  }
int maxi=-1e8;
for(int dj1=-1; dj1<=+1; dj1++)
  {
  for( int dj2=-1; dj2<=+1; dj2++)
    {
      int value=0;
  if(j1==j2) value=grid[i][j1];
  else
    value = grid[i][j1] + grid[i][j2];
  value += f(i+1, j1+dj1, j2+dj2, r, c, grid, dp);
    maxi = max(maxi, value);
  }
  }
return dp[i][j1][j2] = maxi;
}
```

@ Aashish kumar Nayak

PAGE NO.:
DATE: / /

```
int maximumchocolate( int r, int c, vector<vector<int>,
        & grid)
{
  vector<vector<vector<int>>> dp[r, vector<vector<int>>
        (c, vector<int>(c,-1)));

  return f(0,0, c-1, r, c, grid, dp);
}
```

Tabulation

```
int MAXchocolate( int n, int m, vector<vector<int>> &grid)
{
 vector<vector<vector<int>>> dp(n, vector<vector<int>>(m, vector<int>
 (m,0)));
   for(int j1 = 0; j1<m; j1++)
    {
      for(int j2=0; j2<m; j2++)
       {
         if(j1==j2)
        dp[n-1][j1][j1] = grid[n-1][j1];
       else
       dp[n-1][j1][j2] = grid[n-1][j1] + grid[n-1][j2];
       }
    }
for( int i= n-2 ; i>=0; i--)
   { for(int j1=0 ; j1<m; j1++)
     { for( int j2=0;j2<m ; j2++)
       { int maxi = -1e8;
         for(int dj1=-1; dj1<=+1; dj1++)
          {
            for(int dj2=-1; dj2<=+1; dj2++)
```

PAGE NO.:
DATE: / /

```
{ int value = 0;
  if(j1==j2) value = grid[i][j1];
  else
    value = grid[i][j1] + grid[i][j2];

  if(j1+dj1>=0 && j1+dj1<m
  && j2+dj2>=0 && j2+dj2<m)
      value += -1e8;
  maxi = max(maxi, value);
  }
  }
  dp[i][j1][j2] = maxi;
  }
  }
  }
  return dp[0][0][m-1];
}
```

∴ int dp[r][c][c];
1st 1d → two variable
2D → 1D DP
3D → 2D DP

Space Optimization :-

```
int maximumChocolate(int n, int m, vector<vector
    <int>> &grid)
{
  vector<vector<int>> front(m, vector<int>(m,0));
  vector<vector<int>> cur(m, vector<int>(m,0));

  for(int j1=0; j1<m; j1++)
  { for (int j2=0; j2<m; j2++)
    { if(j1==j2) front[j1][j2] = grid[n-1][j1];
```

@ Aashish Kumar Nayak

PAGE NO.:
DATE: / /

```
else front[j1][j2] = grid[n-1][j1] + grid[n-1][j2];
    }
}
for(int i = n-2 ; i>=0 ; i--)
{ for(int j1=0; j1<m; j1++)
    { for(int j2=0; j2<m; j2++)
        {
            int maxi = -1e8;
        for(int dj1 = -1 ; dj1 <= +1 ; dj1++)
        {
            for(int dj2 = -1; dj2 <= +1 ; dj2++)
            {
                int value = 0;
            if(j1 == j2) value = grid[i][j1];
            else value = grid[i][j1] + grid[i][j2];
            if(j1+dj1 >= 0 && j1+dj1 < m &&
               j2+dj2 >= 0 && j2+dj2 < m)
                value += front[j1+dj1][j2+dj2];
            else
                value += -1e8;
            maxi = max(maxi, value);
            }
        }
        cur[j1][j2] = maxi;
    }
} front = cur;
}
return front[0][m-1];
}
```

DP-14

SUBSET SUM Equa to K

DP on subsequences / subsets

(contigous) Non contigous

[1, 3, 2] → [1, 2], [3, 2]

→ Generate all subsequence & select if any of them gives a sum of K

Tabulation

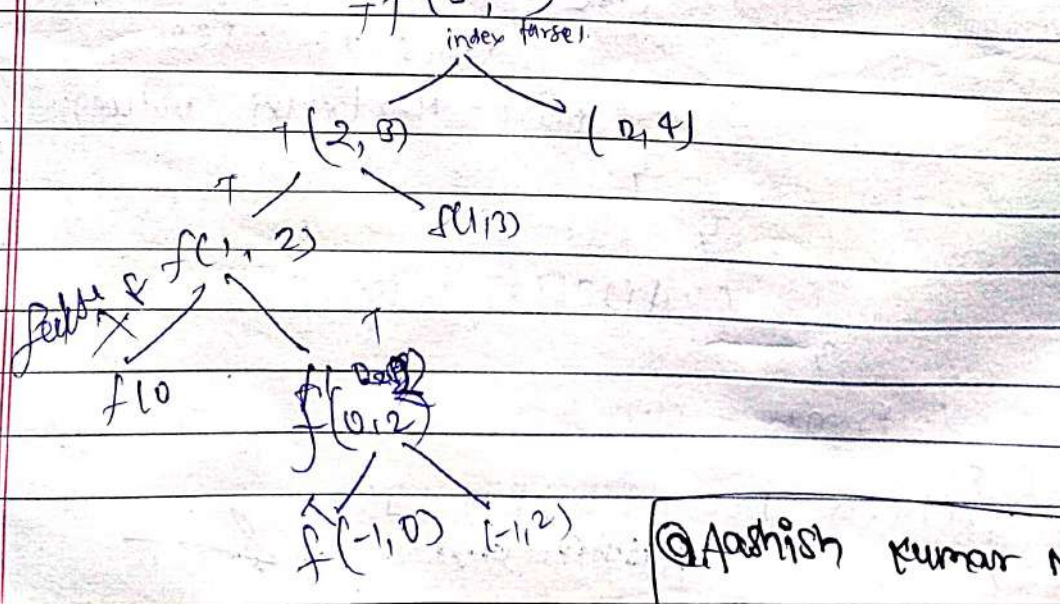
① Base case

② Form the Nested loops for

ind → [1 → n-1]

target → [1 → target]

arr → [2, 3, 1, 1] target = 5



@Aashish Kumar Nayak

Recursion

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

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

  bool take = false;

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

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

bool subsetSumK(int n, int k, vector<int> &arr)
{
  vector<vector<int>> dp(n, vector<int>(k+1, -1));

  return f(n-1, k, arr, dp);
}
```

$$TC = O(2^n)$$
$$SC = O(N)$$

PAGE NO.:
DATE : / /

Tabulation:

```
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] = true;

    dp[0][arr[0]] = true;

    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];
}
```

TC - O(N x target)
SC - O(N x target)

Whenever you see dp[ind-1] means we can optimise space

@Ashish Kumar Nayak

```
bool subsetSumToK( int n, int k, vector<int> &arr)
{
    vector<bool> prev(k+1,0), cur(k+1,0);
    prev[0] = cur[0] = true;
    prev[arr[0]] = true;
    for(int ind=1; ind<n; ind++)
    {
        for(int target=1; target<=k; target++)
        {
            bool not_take = prev[target];
            bool take = false;

            if(arr[ind] <= target)
                take = prev[target-arr[ind]];

            cur[target] = take | not_take;
        }
        prev = cur;
    }
    return prev[k];
}
```

O(1)

Aashish Kumar Nayak

DP-15

Partion Equal Subset sum

PAGE NO.:
DATE: / /

{2, 3, 3, 3, 4, 5}

{2, 3, 5} Subset 1

{3, 3, 4} Subset 2

Subset with n sum

Exactly 2 Subset

$S_1 = S_2$

Entire array sum = S

S

S_1 S_2

$S_1 = S_2 = S/2$

S = total sum

if S = odd → (X)

not possible

If S = even

then look for target sum $\frac{S}{2}$

If one subset you got ($\frac{S}{2}$) then you already got 2nd subset.

NOW this question converts into

arr → [ ]

subset sum = ($S/2$)

where

S = total sum of given array.

@Ashish kumar Nayak

@ Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Space optimization code :-

```
bool subsetSumTOK( int n, int K, vector<int> &arr)
{
   vector<bool> prev(K+1, 0), cur(K+1, 0);
    prev[0]= curr[0] =true;
     prev[arr[0] ] = true;

    for( int ind =1; ind<n; ind++)
      {  for( int target =1; target<=K; target++)
         {
            bool notTake =prev[target];
           bool take =false;

           if(arr[ind] <= target)
             take = prev[target - arr[ind]];
              curr[target] =take | notTake;

         } Prev=curr;
      }
      return prev[K];
    }

bool Canpartition( vector<int> &arr, int n)
{ int totsum =0;
  for(int i=0; i<n; i++)
       { totsum += arr[i];
       }
  if(totsum%2!=0) return false;
  int target = totsum/2;
  return subsetTOK(n, target, arr);
}
```

@Aashish Kumar Nayak

DP-16

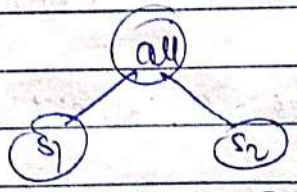
PAGE NO.:
DATE: / /

Partition A set into Two Subset with minimum Absolute sum

[1, 2, 3, 4]

[1, 2, 3, 4]

{1,2} {3,4} = |7 - 3| = 4
{1,3} {2,4} = |6 - 4| = 2
{1,4} {2,3} = |5 - 5| = 0



Sum1 Sum2

abs|Sum1 - Sum2| is minimum

Subset Sum → Tabulation

if we check for a target = K
We can derive if every possible target between (1 & K) is ✓/✗

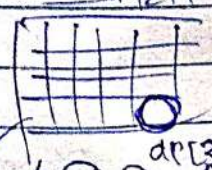
[3, 2, 7]

S1 min 0 → S1 max 12

all possible S1

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
✓ ✗ ✓ ✓ ✗ ✓ ✗ ✓ ✗ ✓ ✓ ✗ ✓

dp[3][12+1] Ans



S2 = 12

diff: 12 0 | 11 10 | 10 8 | 9 6 | 8 4 | 7 2 | 6 0 | 5 2 | 4 4 | 3 6 | 2 8 | 1 10 | 0 12

int dp[n][totsum+1]

Subset sum code

```
int minsubsetsumdifference( vector<int> &arr, int n)
{
    int totsum = 0;
    for(int i=0; i<n; i++)
        totsum += arr[i];

    int k = totsum;
vector<vector<bool>> dp(n, vector<bool>(k+1, 0));
    for(int i=0; i<n; i++) dp[i][0] = true;
    if(arr[0] <= k) dp[0][arr[0]] = true;
    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;
        }
    }

    int mini = 1e9
    for(int s1 = 0; s1 <= totsum/2; s1++)
    {
        if (dp[n-1][s1] == true)
        { mini = min(mini, abs((totsum
                    -s1) - s1));
        }
    }
    return mini;
}
```

@Aashish Kumar Nayak

DP-17

Number of subsets

{1, 2, 2, 3}

{1, 2} target = 3

{1, 2}

{3}

no. of subset = 3

if case of count

in Base case
return 1 if true
return 0 if false

Always

Recurrance

1.) Express in terms of (index, target)
2.) Explore all the possibilities
3.) Sum up all the possibilities & return.

0	1	2	3
1	2	2	3

(ind) (ind+1)

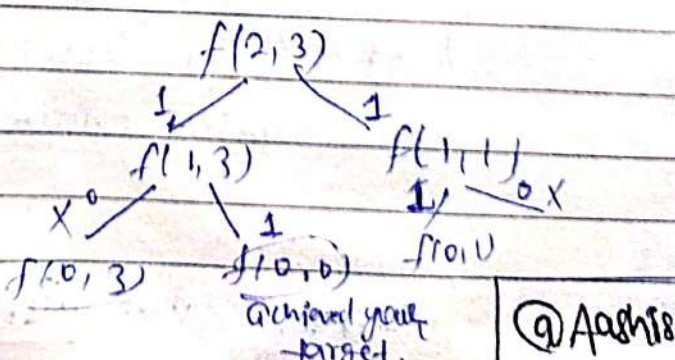
F(n-1, target)

not_pick = f(ind-1, target);

pick = 0;

if (a[ind] <= s) {
pick = f(ind-1, target-a[ind]);

return pick + notpick;
}



PAGE NO.:
DATE: / /

no. of subset = 2 {1,2}, {3}

Tabulation

(1) Base case
(2) looks at the changing parameter & write Nested loop
(3) copy the recursion

```
int dp[N][S+1];
for(i=0 → n-1) dp[i][0] = 1
if(a[0] <= S)  dp[0][a[0]] = 1

for(i=1 → n-1)
    for(s=0 → sum)
    {
        copy the recursion;
    }
```

TC - O(N x Sum)
SP - O(N x Sum)

PAGE NO.:
DATE : / /

Recursion code :-

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

    int notTake = f(ind-1, sum, num, dp);
    int take = 0;

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

    return dp[ind][num] = notTake + take;
}

int findWays(vector<int> &num, int tar)
    {
        int n = num.size();

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

    return f(n-1, tar, num, dp);
}
```

Tabulation :

```
int findways (vector<int> &num, int tar)
{
    int n = num.size();
    vector<vector<int>> dp(n, vector<int> (tar+1, 0));
    for(int i=0; i<n; i++)
        { dp[i][0] = 1;
    if (num[0] <= tar)
        dp[0][num[0]] = 1;
    for(int ind = 1; ind < n; ind++)
        {
            for(int sum = 0; sum <= tar; sum++)
                {
                int notTake = dp[ind-1][sum];
                int take = 0;
                if(num[ind] <= sum)
                {
                    take = dp[ind-1][sum - num[ind]];
                }
                dp[ind][sum] = notTake + take;
            }
        }
    return dp[n-1][tar];
}
```

@Aashish Kumar Nayak

PAGE NO.:
DATE : / /

space optimization code :-

```
int findways( vector<int> &num, int tar)
{
    int n = num.size();
    vector<int> prev(tar+1,0), cur(tar+1);
    prev[0] = cur[0] = 1;
    if(num[0] <= tar)
        prev[num[0]] = 1;

for(int ind = 1; ind < n; ind++)
    {
        for(int sum=0; sum<=tar; sum++)
        {
            int notTake = prev[sum];
            int take = 0;
            if(num[ind] <= sum)
                take = prev[sum - num[ind]];

            cur[sum] = notTake + take;
        }
        prev = cur;
    }
    return prev[tar];
}
```

18

PAGE NO.:
DATE:

Count partitions with given sum difference

arr → {0, 0, 1} sum = 1

ans = 4

{0, 1}
{0, 1}
{0, 0, 1}
{1}

no. of zeros → (2)

pow (2, n) x ans

power of

$2^2 \times 1$

$4 \times 1 = 4$ subset

$S_1 >= S_2$

& $S_1 - S_2 = D$

$S_1 = totsum - S_2$

$D = totsum - S_2 - S_2$

$D = totsum - 2S_2$

$$S_2 = \frac{totsum - D}{2}$$

Count of subset = $\left(\frac{Totsum - D}{2}\right)$

DP-17 → modified targets

Totsum − D >= 0

totsum − D has to be even

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Recursion + Memorization code

```
int mod = (int)(1e9+7);
int f(int ind, int sum, vector<int> &num, vector<vector
    <int>> &dp)
{
 if(ind==0)
  { if(sum==0 && num[0] )
     return 2;
    if(sum==0 || sum==num[0] )
     return 1;
    return 0;
  }
   if(dp[ind][sum] != -1)
       return dp[ind][sum];

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

   return dp[ind][sum] = (notTake + take)%mod;
 }

  return dp[n-1][tar];
}
int countPartitions(int n, int d, vector<int>& arr)
{
  int totSum = 0;
  for(auto &it : arr) totSum += it;
 if(totSum-d<0 || (totSum-d)%2) return false;
  return findWays(arr, (totSum-d)/2);
}
```

```
int findways(vector<int> &sum, int tar)
{
    int n = num.size();
    vector<vector<int>> dp(n, vector<int>(tar+1, -1));
    return f(n-1, tar, num, dp);
}

int countpartitions(int n, int d, vector<int> &arr)
{
    int totsum = 0;
    for(auto &it:arr)
        totsum += it;
    if((totsum - d < 0) || (totsum - d)%2)
        return false;
    return findways(arr,(totsum - d)/2);
}
```

Tabulation :-

```
int findWay(vector<int> &num, int tar)
{
    int n = num.size();
    vector<vector<int>> dp[n, vector<int>(tar+1, 0)];
    if(num[0] == 0)
        dp[0][0] = 2;
    else dp
        dp[0][0] = 1;
    if
```

Space optimization code

```
int findways(vector<int> &num, int
{
    int n= num.size();
    vector<int> prev(tar+1,0),cur(tar+1,0);
    if(num[0]==0)
        prev[0]=2;
    else prev[0]=1;

    if(num[0]!=0 && num[0]<=tar)
        prev[num[0]]=1;
    for(int ind=1; ind<n; ind++)
    {
        for(int sum=0; sum<=tar; sum++)
        {
            int notTake = prev[sum];
            int take=0;
            if(num[ind]<=sum)
            take = prev[sum-num[ind]];
            cur[sum] = (notTake + take)%mod;
        }
        prev = curr;
    }
    return prev[tar];
}
int
```

DP-19

PAGE NO.:
DATE 30/05/22

0-1 Knapsack

$n = 3$

wt. →	3	4	5
val →	30	50	60

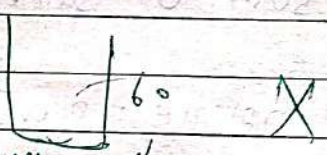
5+3 = 8 weight
val = 60+30
= 90 ✓

4+3 = 7
val = 50+30 = 80 ✗

① Choose 1st costly products.

Greedy Not works here

wt ⇒	3	2	5
val ⇒	30	40	60



wt. 2+3 = 5
val. 40+30 = 70

Law of uniformity was not there that why Greedy fails.

Try out all combination

↓

Take the best value combination

Recursion

(i) Express in terms of (index, wt)

(ii) Explore all possibilities

(iii) looking for maxim.

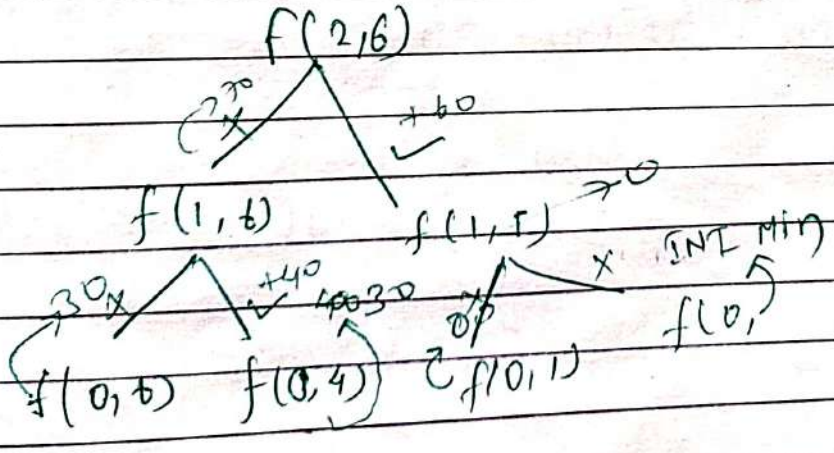
3	4	5
30	50	60

```
f(ind, w)
{ if(ind == 0)
    { if(wt[0] <= w) return val[0]; }
    else {return 0;}
NOT take = 0 + f(ind-1, w)
take = INT_MIN;
if( wt[ind] <= w)
    take = val[ind] + f(n-1, w-wt[ind]);
return max(not take, take);
}
```

wt → | 6 |
val → | 100 |
0

wt →	3	2	5
val	30	40	60

wt = 6



f(0,w)

Memoization.

ind, w

dp[ind+1][w+1];

TC = O(n x w)

SP = O(N x W) + O(N)

Tabulation

@Aashish Kumar Nayak

@Aashish kuma

PAGE NO.:
DATE: / /

Tabulation :-

1. Base case.
2. Write the changing parameters
   ind, w
3. Loops
   no. of loops = no. of changing parameter.

```
int dp[ind][w];
```

Base case :-

```
int dp[ind][w] = 0;

for (i = wt[0] → w)  dp[0][i] = val[0];
```

Recursion + Memoization

PAGE NO.:
DATE: / /

```
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 Not_take = 0 + f(ind-1, W, wt, val, dp);

  int take = INT_MIN;
  IF( wt[ind] <= W)
    {
    take = val[ind] + f(ind-1, W-wt[ind], wt,
                        val, dp);
    }
  return dp[ind][W] = max( take, Not_take);

int knapsack( vector<int> weight, vector<int> value,
  int n, int W)
  {
    vector< vector<int>> dp(n, vector<int>(W+1, -1));
    return f(n-1, W, weight, value, dp);
  }
```

@Ashish kumar Nayak

PAGE NO.:
DATE : / /

Tabulation:

```
int knapsack(vector<int> wt, vector<int> val,
int n, int weight)
{
  vector<vector<int>> dp(n, vector<int>(weight+1, 0));

  for(int w=wt[0]; w<=weight; w++)
  {
    dp[0][w] = val[0];
  }

  for(int ind=1; ind<n; ind++)
  {
    for(int w=0; w<=weight; w++)
    {
      int not_take = 0 + dp[ind-1][w];
      int take = INT_MIN;
      if(wt[ind]<=w)
      {
        take = val[ind] +
          dp[ind-1][w-wt[ind]];
      }
      dp[ind][w] = max(take, not_take);
    }
  }
  return dp[n-1][weight];
}
```

@Aashish Kumar Nayak

Space optimization

PAGE NO.:
DATE: / /

```
int knapsack( vector<int> wt, vector<int> val, int n, int
weight)
{
    vector<int> prev( weigh + 1, 0), cur(weight + 1, 0);
    vector<vector<int>> dp(n, vector<int> (Max weight +1, 0);

    for(int w = wt[0]; w <=
for(int ind = 1; ind < n; ind++)
  {
   for(int w = 0; w <= Max weight ; w++)
   {
     int not_take = 0 + prev[w];

     int take = INT_MIN;
      if( wt[ind] <= w)
      {
       take = val[ind] + prev[w - wt[ind]];
      }

      cur[w] = max(take, not_take);
   }
    prev = curr;
  }
   return prev[weight];
}
```

Single array space optimization

```
int knapsack(vector<int> wt, vector<int> val,
  int n, int maxweight)
{
    vector<int> prev(weight+1, 0);

  for(int w=wt[0]; w<=weight; w++)
      {
        prev[w] = val[0];
      }
  for(int ind=1; ind<n; ind++)
    {
      for(int w = maxweight; w>=0; w--)
      {
        int not_take = 0 + prev[w];
      int take = INT_MIN;

  if(wt[ind] <= w)
    {
      take = val[ind] + prev[w-wt[ind]];
    }
  prev[w] = max(take, not take);
    }
}
  return prev[maxweight];
}
```

Minimum coins | DP on subsequence

arr[] → {1, 2, 3} target = 7

coins may be {3, 3, 1} ans = 3 minimum coins

X (2, 2, 2, 1) ans = 4

Greedy

{1, 2, 3} target = 7

7/3 = (2) {3, 3, 1}

remain = 1

1/2 = 0 3 coins

1/1 = (1)

greedy work for this case but might failed in other case

lets take an example where greedy fails

{9, 6, 5, 1} target = 11

11/9 = (1)

2 remain

2/6 = 0 ans should be 2+1 = 3 but

2/5 = 0 if we choose 6 & 5 ans = 2

2/1 = (2) so greedy fails here.

Approach :- Trying out all combos to form target

↓

take the combo that has min coins

Rules :-

1. Express the recurrence in terms of index.
2. Express or do all stuffs.
3. min. of all.

Recursion

@Aashish Kumar Nayak

Whenever there is statement like infinite supply, multiple case's it will stand at same index it will not go back.

T.C. -> >> O(2^n)

S.C. -> O(N) - recursion

```
f(ind, T)
{
if(ind == 0)
  if( T % arr[0] == 0) return T/arr[0];
  else return 1e9;
```

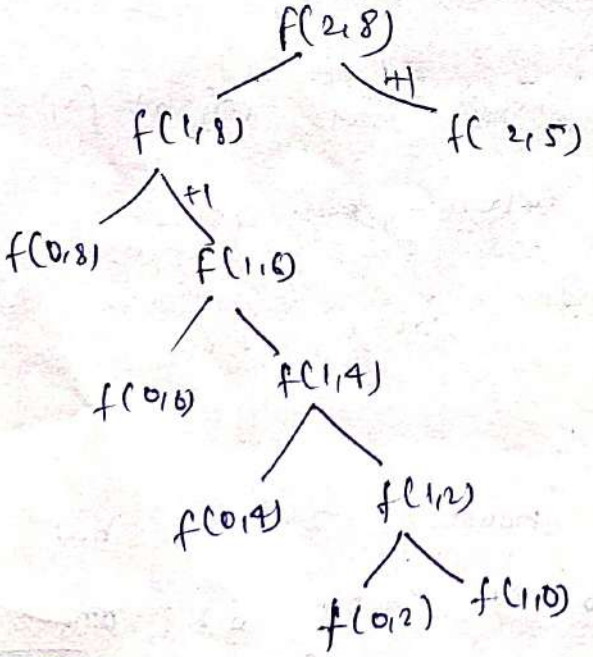
X [6] target = 7
arr[0]

✓ [7] target = 7

[4] target = 12

Base case

```
int not_take = 0 + f(ind-1, T);
int take = INT_MAX;
if(coins[ind] <= T)
    take = 1 + f(ind, T - coins[ind]);
return min(take, NOTtake);
}
```



T.C. -> >> O(2^n)
S.C. -> O(N)

Term Exponential.

Now we will Memoize it,

changing parameters -> ind, target

dp[ind][target] = Now

T.C. = O(N*T)
S.C = O(N*T) + O(T) Auxiliary space

@Aashish Kumar Nayak

Tabulation

*** Any memoization can be converted into tabulation.

1. Base case
2. ind target and write it in opposite fashions.
3. Copy the tabulation.

for any target :

```
for(T=0 → target)
    if(T%a[0] == 0)
        dp[0][T] = T/a[0];
    else
        dp[0][T] = INT_MAX;
```

Code :

```
int f(ind, int T, vector<int> &nums, vector<vector<int>> &dp)
{ if(ind == 0)
  { if(T%nums[0] == 0)
        return T/nums[0];
    return 1e9;
  } if(dp[ind][T] != -1) return dp[ind][T];
int not_take = 0 + f(ind-1, T, nums, dp);
int take = INT_MAX;
if(nums[ind] <= T)
  { take = 1 + f(ind, T-nums[ind], nums, dp);
  }
return dp[ind][T] = min(take, not_take);
}
int minimumElements(vector<int> &num, int target)
{ int n = num.size(); vector<vector<int>> dp(n, vector<int>(target+1, -1));
int ans = f(n-1, target, nums, dp);
if(ans >= 1e9) return -1;
return ans;
}
```

@Aashish Kumar Nayak

Tabulation code :-

```
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 = 0 + 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);
       }
    }
    int ans = dp[n-1][target];
     if(ans >= 1e9)
          return -1;
     return ans;
}
```

Space Optimization : (using prev1 & prev2)

```
int minimumElements(vector<int> &nums, int target)
{
    int n = nums.size();
    vector<int> prev(target+1, 0), cur(target+1, 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++)
    {
        for(int T=0; T<=target; T++)
        {
            int notTake = 0 + prev[T];
            int take = INT_MAX;
            if(nums[ind] <= T)
            {
                take = 1 + cur[T - nums[ind]];
            }
            cur[T] = min(take, notTake);
        }
        prev = cur;
    }
    int ans = prev[target];
    if(ans >= 1e9)
        return -1;
    return ans;
}
```

@Aashish Kumar Nayak

Target sum problem | DP on subsequence

arr[] = {1, 2, 3, 1} Assign +,-

target = 3

+ + - -
1 2 3 1 = -1
1 3 0 -1

count all the ways to get target.

- + + -
1 2 3 1 = 3
-1 1 4 3

2 ways

+ - + +
1 2 3 4 = 3

$S1 - S2 = D$

sum partition

- + + -
[1 2 3 1]

3 + 2 -1 -1
S_1 S_2
5 - 2 = 3
S_1 S_2

same problem like in DP-18

Code:

```
int targetSum(int n, int target, vector<int>& arr)
{ return countPartition(n, target, arr);
}
int countPartition(int n, int d, vector<int>& arr)
{ int totSum = 0;
  for(auto &it : arr)
      totSum += it;
  if(totSum - d < 0) || (totSum - d) % 2
      return false;
  return findWaysArr, (totSum-d)/2;
}
```

```
int findways(vector<int> &num, int tar)
{
    int n = num.size();
    vector<int> prev(tar+1,0), cur(tar+1,0);
    if(num[0] == 0)
        prev[0] = 2;
    else
        prev[0] = 1;
    if(num[0] != 0 && num[0] <= tar)
        prev[num[0]] = 1;
    for(int sum = 0; sum <= tar; sum++)
    {
        int notTake = prev[sum];
        int take = 0;
        if(num[ind] <= sum)
            take = prev[sum - num[ind]];
        cur[sum] = (notTake + take);
    }
    prev = cur;
}
return prev[tar];
}
```

DP-22 ways to make coin change

N=3 | 1 | 2 | 3 | target=4

any element can be used any no. of times.

{1, 1, 1, 1} = 4
{1, 1, 2} = 4
{2, 2} = 4
{1, 3} = 4

} 4 differs

figout total no. of ways → try out all the ways

f( )
{
Base case (if destination is meet. Recursion ↓ else.)
1 or 0

f4 ___
f2 ___
return (— + —);
}

Recurance :

1. Express → (ind, T)
2. Explore all possibility → not Take / take } By all the DP subsequen.. problem.
3. Sum all possibilities and return.

⊛ | 1 | 2 | 3 |
0 1 2

f(2, 4)
ind T

This index 2, in how many ways can you form 4

@Aashish Kumar Nayak

```
f(ind, T)
{
  if(ind == 0)
  {
    if(T % arr[0] == 0)
      return 1;
    else return 0;
  }
  notTake = f(ind-1, T);
  take = 0
  if(arr[ind] <= T)
    take = f(ind, T-arr[ind]);
  return notTake + take;
}
```

✓ [4] arr[0] tar = 4

× [3] tar = 4

✓ [2] tar = 4

tar % arr[0] then can be formed.

we are standing at same index.

So, T.C. >> $O(2^n)$ exponential

S.C. >> O(N) standing on the same index.

Memoization will solve this issue

T.C. = O(N X T)

S.C = O(N X T) + O(target) auxiliary space ASS

we Tabulation to optimise ASS

steps :- Tabulation — Bottom-up approach

1. Base case
2. ind (1 — n-1)
   T → (0, T)
3. Copy the recurrance.

```
for(T → 0 → target)
dp[0][T] = (T % arr[0] == 0)
```

Code :-

```
long countWaysToMakeChange(int *denomination, int n, int value)
{
  vector<vector<int>> dp(n, vector<long>(value+1, -1));
  return f(n-1, value, denomination, dp);
}
```

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

Tabulation :-

```
long countWaysToMakeChange(int *denomination, int n, int value)
{   vector<vector<long>> dp(n, vector<long>(value+1, 0));
    for(int T=0; T<=value; T++)
    { dp[0][T] = (T % a[0] == 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];
}
```

@Aashish Kumar Nayak

Space optimization

Whenever you see like ind-1, ind. then you can always optimise by using couple of stuffs.

Code :-

```
long countWaysToMake(int *a, int n, int value)
{
    vector<vector<long>> dp(n, vector<long>(value+1, 0));
    vector<long> prev(value+1, 0), cur(value+1, 0);
    for(int T=0; T<=value; T++)
    {
        prev[T] = T%a[0]==0;
    }
    for(int ind=1; ind<n; ind++)
    {
        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];
}
```

DP-23 unbounded knapsack

prerequisite
DP-19 — 0/1 knapsack
in this there was only one occurrences.

wt → {2, 4, 6}
val → {5, 11, 13}

But here infinite supply of every item. we can pick any one item many times.

Ex:-
=> wt → 6 (13) wt → 4 (11)

wt → 6
10
wt → 2 x 2
+ 5 x 2 = 23

wt → 4 x 2 + wt 2
11 x 2 + 5 = 27 maximum possible value.

(wt → 2) x 5 = 10 wt
5 x 5 = 25 val

Base case we write for 0

val 10
wt 5
arr[]

W = 8 ✓ 2
if W = 2 x 0

int (W / wt[0]) x val[0]

```
f(ind, W)
{ if(ind == 0)
    return [ W/wt[0] x val[0] ];

notTake = 0 + f(ind-1, W);
take = INT_MIN;
if(wt[ind] <= W)
    take = val[ind] + f(ind, W - wt[ind]);

return max(take, notTake);
```

Memoize — T.C. — exponential

Tabulation
1. Base case
2. Changing parameter ind
3. Copy the recurrance.

=> O(W)

@Ashish Kumar Nayak

w can be anything 0, 1, 2, ——— wt. bag

for(w → 0 to wt bag)

{ dp[0][w] = $\left(\frac{w}{wt[0]}\right)$ × val[0]

}

Code ÷ * memoized

```
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);
}

int unboundedKnapsack(int n, int w, vector<int> &val,
                      vector<int> &wt)
{ vector<vector<int>> dp(n, vector<int>(w+1, -1));
    return f(n-1, w, val, wt, dp);
}
```

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-1, int — so space optimized is possible.

```
int unboundedKnapsack( int n, int w, vector<int> & val,
                       vector<int> & wt)
{
    vector<int> prev(w+1, 0), cur(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] + cur[w - wt[ind]];
              }                    //prev[w - wt[ind]];
//prev[w]= max(take,notTake);
              cur[w] = max(take, notTake);
        }
R       // prev = cur; //Remove this line
    }
    return prev[w];
}
```

Remove.

prev: 3 4 5 6 7 1 2

not take = 3 ; 4th cur

take =

than rewrite.

1D array optimization

DP-24 Rod cutting problem | DP on subsequences

N = 5 → Rod length

price array [] = | 2 | 5 | 7 | 8 | 10 |
1 2 3 4 5

length 1 → for length 1 price = 2

for length 2 price = 5

we need to cut the rod into pieces

price 2 2 2 2 2
length 1 1 1 1 1 = 10

2 5 5
1 2 2 = 12 ↑↑ maximum

5 7
2 3 = 12 maximum

we need to print maximum cost.

1 2 3 4 5

Collect rod length to make N

↓

Maximize the price — Similar to knapsack

| 2 | 5 | 7 | 8 | 10 |
0 1 2 3 4

means 4 length — means 5 length

Try to pick lengths and sums them up to make the given N.

↳ In all possible ways

↳ Recurrence

1. Express in terms of index
2. Explore all possibilities ← not take / take
3. Maximize the price or possibilities.

2	5	7	8	10
0	1	2	3	4

f(4, N)

Till index 4, what is the minimum price you obtain

```
f(ind, N)
{
  if(ind == 0)
    return N*price[0];
  int notTake = 0 + f(ind-1, N);
  int take = INT_MIN;
  rod_length = ind+1;
  if(rod_length <= N)
    take = price[ind] + f(ind, N - rod_length);
  return max(take, notTake);
}
```

n-1 ⟶ 0

[6] — Price[0]

rod_length = 1

{1, 1, 1 ... 12 times}

N = 12 → looking to make it.

price = N*price[0];

Recursion ⟶ Memoization

always greater than 2^n i.e. expon

T.C. - O(Exponential)

S.C. → O(Target)

T.C. ind, N → [N]x[N+1]

T.C. O(N×N)

S.C. → O(N×N) + O(Target)

AS

Mem Tabulation :

1. Base car
2. Changing parameter — ind N
3. Copy recurrance

Base case.

```
for (N → 0 to n)
    dp[0][N] = N*price[0];
```

~~for~~ ind → 1 ⟶ n-1

N ⟶ 0 ⟶ N

(Copy the recurrance)

@Aashish Kumar Nayak

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

Tabulation :-

```
int cutRod ( vector<int> &price, int n)
{ vector<vector<int>> dp(n, vector<int>(n+1, 0));
   for ( int N=0; N<=n; N++)
     { dp[0][N] = N* price[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 = price[ind] + dp[ind][N-rodLength];
}
dp[ind][N] = max(take, notTake);
}
}
return dp[n-1][n];
}
```

Space optimization :-

```
int cutRod(vector<int> &price, int n)
{ vector<vector<int>> dp(n, vector<int>(n+1,0));
vector<int> prev(n+1,0), cur(n+1,0);
for(int N=0; N<=n; N++)
{ prev[N] = N* price[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 = price[ind] + cur[N-rodLength];
}
cur[N] = max(take, notTake);
} // prev[N-rodLength];
} //prev[N]
return dp[n-1][n];
}
```

we can do 1D array optimization like previous one.

@Ashish Kumar Nayak

DP on Strings

[DP on strings] → Comparison of strings
→ replace / edit

longest common [subsequences]

S_1 = "adebc" S_2 = "dcadb"

DP-25 **Longest Common Subsequence :-**

abc → [a, b, c, ab, bc, ac, abc, " "]

power set / ~~Rec~~ or Recursion ← we can use these ~~two~~ method to print ~~all~~ the subset.

S_1 = "a d e b c" — 2^5 subsequences
1 2 3 4 5

a, d, e, b, c, ad, ae, ac, de, db, dc, adc, ——— adb

S_2 = "dcadb" — 2^5 subsequences

d, c, a, d, b, dc, da, dd, db, - - - — adb

we have to find all the subsequences and take out the longest one.

Brute force

Generate all subsequences

take one & compare ↓ so on

T.C. :- O(exponential)

New approach

⇒ Generate all subsequences & compare on way.

↓

Recursion method, we will not generate instead we will use palameter to generate them.

We will write some recurrence which give the answer through the way.

Rules to write the recurrence

1. Express everything in terms of index
2. Explore possibilities on that index.
3. Take the best among them.

Let's take an example

a c d | c e d

if match: 1 + f(ind1-1, ind2-1);

a c | c e

if not match: f(ind1-1, ind2) , f(ind1, ind2-1)

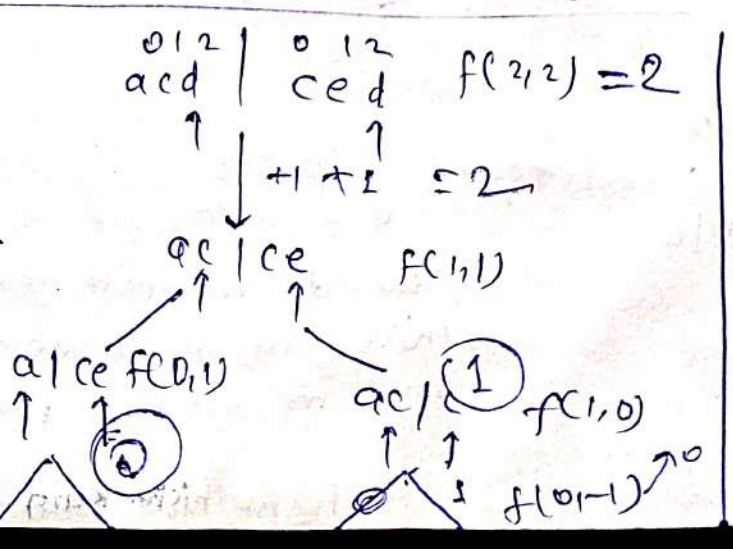
0 + max[f(ind1-1, ind2), f(ind1, ind2-1)];

f(ind1, ind2) -ve means end of the strings.

```
{
// base case
if(ind1 < 0 || ind2 < 0)
    return 0;
```

match: if(s1[ind1] == s2[ind2])
return 1 + f(ind1-1, ind2-1);

not match: return 0 + max[f(ind1-1, ind2), f(ind1, ind2-1)];



Recursion $2^n \times 2^m \approx$ Exponential

↓optimised if there are overlapping sorce problem then apply memoization.

Code :-

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

int lcs(string s, string t)
{
    int n = s.size();
    int m = t.size();
    vector<vector<int>> dp(n, vector<int>(m, -1));
    return f(n-1, m-1, s, t, dp);
}
```

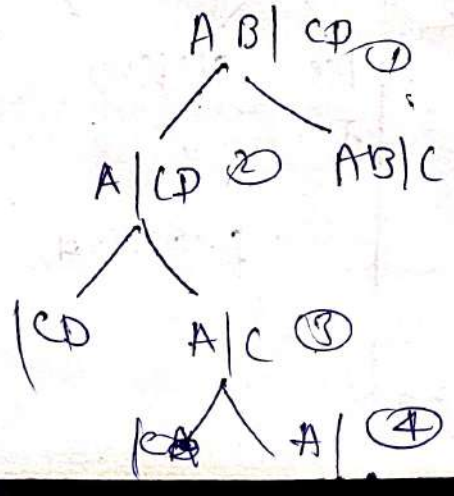
n+1 m+1

n m

T.C. - $O(N \times M)$

S.C - $O(N \times M)$ + $O(N+M)$

A.S.S.



2+2 = 4 alternate deletion

if we do alternate deletion then there can be n deletion from s_1 and m deletion from s_2.

Tabulation :-

1. Copy the Base Case
2. changing the parameter
3. Copy the recurrance.

shifting of indexes.

-1 0 1 2 3 ... n-1

```
if (i==0 || j==0) return 0
    return dp[0][j]

for(j=0 → m-1)  dp[0][j]=0
for(i=0 → n-1)  dp[i][0]=0
```

Code :-

```
int lcs(string s, string t)
{
  int n=s.size();
  int m= t.size();
  vector<vector<int>> dp(n+1, vector<int>(m+1, -1));
for(int j = 0; j<=m; j++) dp[0][j] = 0;
for(int i= 0; i<=n ; i++)  dp[i][0] = 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(m+1, 0), cur(m+1, 0);
    for(int j=0; j<=m; j++)
    {
        prev[j] = 0;
    }
    for(int i=1; i<=n; i++)
    {
        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]);
        }
        prev = cur;
        cur = prev;
    }
    return prev[m];
}
```

D6-26 print longest common subsequences :-

S_1 = "abcde" S_2 = "bdgek"

len(LCS) = 3 // in previous problem

Sout = bde

dp

		0	1 b	2 d	3 g	4 e	5 k
0		0	0	0	0	0	0
1	a	0	0	0	0	0	0
2	b	0	1	1	1	1	1
3	c	0	1	1	1	1	1
4	d	0	1	2	2	2	2
5	e	0	1	2	2	3	3

max(3,2)

come here.

match

if not match then go to max of diagonal [i-1][j] & [i][j-1]

dp[5][5] = 3

abcde Bdgek

dp[4][2] → 2

abcd bd

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

only when s1[i-1] == s2[j-1]

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

Code

```
i = n, j = m
while(i > 0 && j > 0)
{ if(s1[i-1] == s2[j-1])
  {
  }
  else dp[i][j] = max(dp[i-1][j]
}
```

@Aashish Kumar Nayak

~~else~~ i = m, j = m; len = dp[n][m], string s = ""; index = len-1;

while(i > 0 && j > 0)

for(i = 0 → len) s += "$";

$	$	$

{ if(s1[i-1] == s2[j-1])

{ s[index] = s1[i-1];

index--;

i--; j--;

}

else if (dp[i-1][j] > dp[i][j-1])

{

}

else

}

}

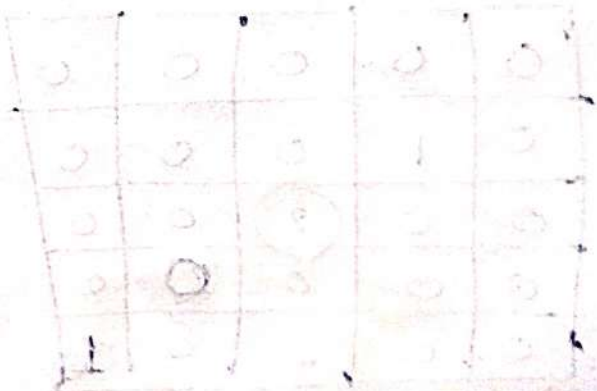
Code :-

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

@Aashish Kumar Nayak

```
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--;
                }
            cout<<ans;
        }
int main()
{ string s1 = "abcde";
    string s2 = "bdgek";
    lCS(s1,s2);
}
```

Tc. O(n+m)



DP-27 Longest Common Substring :-

S_1 = "abcjklp" S_2 = "acjlqp"

Should be consecutive

Les
- matching
  $dp[i][j] = 1 + dp[i-1][j-1]$
- not matching
  $dp[i][j] = max(dp[i-1][j], dp[i][j-1])$

This will not work on substring

lets take an example

acd | axd

This is consecutive so we don't dependent on the previous guys.

Breaking the consecutive rule

ac | ax

a | ax (+1) ac | a

/ax a/d c/a ac/

no need to do a break directly put 0

$dp[i][j] = 0$

abcd abzd

		a	b	z	d
	0	1	2	3	4
0	0	0	0	0	0
a 1	0	1	0	0	0
b 2	0	0	2	0	0
c 3	0	0	0	0	0
d 4	0	0	0	0	1

The maximum value in the entire matrix.

Code

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

Code :-

```
int lcs(string &s, string &t)
{
    int n = s.size();
    int m = t.size();
    vector<int> prev(m+1, 0), cur(m+1, 0);
    int ans = 0;
    for(int i = 1; i <= n; i++)
    {
        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;
}
```

DP-28 **longest palindromic subsequences :**

S = " bbbab"

→ ab
→ bb
→ bbb
→ bbbb — longest one
→ bab

Ans = 4 (length)

Brute force

Generate all the subsequences → Check for palindrom & pick up the longest.

S_1 = " b babcbcab"

S_2 = " bacbcba bb" — reverse order of S_1

Just find lcs (S_1, rev(S_1))

Code :

```
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 j = 0; j<=m; j++) dp[0][j] = 0;
for(int i=0; i<=n; i++) dp[i][0] = 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];
}
int longestPalindromeSubsequence (string s)
{ string t = s;
  reverse(t.begin(), t.end());
  return lcs(s,t);
}
```

DP-29 Minimum insertions to make a string palindrome:

prerequisite → LCS
→ LPS

S = "abcaa"

↓

abcaa aa cba
reverse

max operation = len(S)

S + rev(S) — becomes palindrome.

aa bc baa ⟶ palindrom and min operation = 2

Approach:-

keep the palindrome portion intact.

S = abcaa - a lot of palindrom so I will try to keep longest palindrome intact.

a b c a a

aaa

we will rotate 'b' not rotate then we will place copy of that in right part.

a b c a b a

coding ninja

n — longest palindromic subsequence.

size of string.

Code :-

```
int minInsertion(string &str)
{ return str.size() - longestPalindromeSubseque(str);
}
int longestPalindromeSubsequence(string s)
{string t = s;
 reverse(t.begin(), t.end());
 return lcs(s, t);
}

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

DP-80 Minimum insertion/deletion to convert string A to string B

str1 → "abcd" str2 = "anc"

Delete everything from str1 and insert all character from str2 to str1.

it takes str1.size() + str2.size operations.

intution what can I not touch

a b c d → anc
x x

lets delete b & d

a c — 2 deletion

now insert n

anc — 1 insertion

3 operations

longest common subsequences.

deletion = n - len(lcs)
insertions = m - len(lcs)

n+m - 2*len(lcs) → Ans

Code :

```
int canYouMake(string &str, string &ptr)
{ return str.size() + ptr.size() - 2*lcs(str, ptr);
}
int lcs(string s, string t)
{ int n = s.size();
  int m = t.size();
```

@Ashish Kumar Nayak

```
vector<vector<int>> dp(m+1, vector<int>(m+1,0));
    for(int j=0; j<=m; j++)
        dp[0][j] = 0;
    for(int i=0; i<=n; i++)
        dp[i][0] = 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];
}
```

DP-31 Shortest Common Supersequence :-

S_1 = "brute" S_2 = "groot"

&

"brutegroot" ← len = 10 this can be called common supersequence.

both string should be present in the ans string. and order has to be same.

~~"brut"~~ "bgruoote" — len = 8

S_1 = "bleed" S_2 = "blue"

"bleued"

fix. one length

print #

Common guys should be taken Once

common guy in two string means lcs(—,—)

↓

it tells me longest common guy

"bgruoote"

(S1) (S2)

blue (&) goo

(n+m) − len(lcs)

print string

$s_1[i-1] == s[j-1]$

$dp[i][j] = 1 + dp$

we will create dp table.

LCS table

		0	1 g	2 r	3 o	4 o	5 t
	0	0	0	0	0	0	0
b	1	0	0	0	0	0	0
r	2	0	0	1	1	1	1
u	3	0	0	1	1	1	1
t	4	0	0	1	1	1	2
e	5	0	0	1	1	1	2

if not match max(i,j-1)

common guys get added by 1

etouorbg

reverse it

gbrouote → len(8)

```
i=n, j=m
while( i>0 && j>0)
  { if( s1[i-1] == s2[j-1])
    { ans += s1[i-1];
      i--, j--;
    }
  else if (dp[i-1] > dp[i][j-1])
  {
    ans += s1[i-1];
      i--;
  }
  else
    { ans += s2[j-1];
      j--;
    }
```

@ Apshish Kumar Nayak

i>0 | j>0 | @Aashish Kumar Nayak

String 1 has some length

s2 has some length

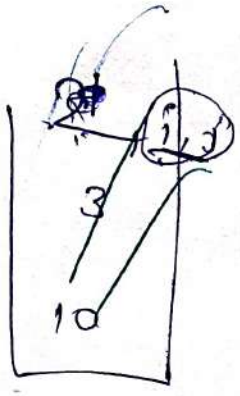
while (i>0) ans += s1[i-1]; i--
while (j>0) ans += s2[j-1], j--

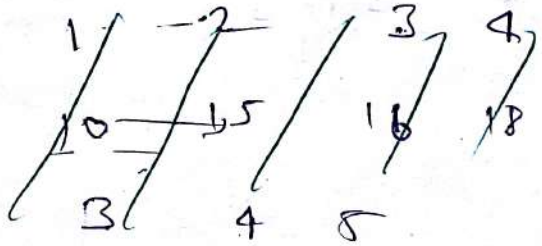
Code :-

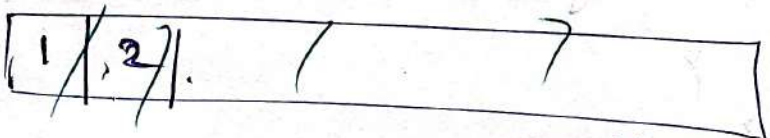
```
string shortestSupersequence( string s, string t);
{   int n = s.size();
    int m = t.size();
vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
for( int j = 0; j<=m ; j++)
    dp[0][j] = 0;
for( int i=0; i<=n; i++
    dp[i][0] = 0;
for(int i = 1; i<=n; i++)
  { for(int j=1; j<=m; j++)
    { if( 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])
    { 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 += t[j-1]
    j--;
}}
reverse(ans.begin(), ans.end());
return ans;
}
```







~~Dp on~~

Distinct subsequences

S = "babgbag" s_2 = "bag"

occurrence of s_2 in s_1 in babgbag → 5 sub distinct ways

Different methods of comparing

Trying all the ways → Recursion

Count the no. of ways

```
f()
{
  Base case ↑' ya

  f1()
  f2()

  return f1()+f2();
}
```

// Base case will return 1 or 0 ∴ count ways.

Q How to write recurrence

(i) Express everything in terms of i & j

(ii) Explore all possibilities

(iii) Count all ways return the summation of all possibilities

(iv) Base case

f(i,j)

S_1 = babgbag (i = n-1)

S_2 = bag (j = m-1)

f(m-1, m-1) ← start call

no. of distinct subsequences of $S_2[0 \ldots j]$ in $S_1[0 \ldots i]$

if(j<0) return 1;

if(i<0) return 0

if($S_1[i] == S_2[j]$) ← if they are matching the string reduced to ba from bag

return (f(i-1, j-1) + f(i-1, j));

else

return f(i-1, j) // Reduced and look for someone else.

Space Complexity :-

Tc → exponential (2^n, 2^m)

Sc → O(N+m)

Optimise → overlapping subproblems

↓ Memoization

T.C. O(N×m)

S.C. O(N×m) + O(N+m)

@Aashish Kumar Nayak

Code :-

```
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 f(i-1, j-1, s, t, dp) + f(i-1, j, s, t, dp);
  return dp[i][j] = f(i-1, j, s, t, dp);
}

int numDistinct(string s, string t)
{ int n = s.size(); int m = t.size();
  vector<vector<int>> dp(n, vector<int>(m, -1));
  return f(n-1, m-1, s, t, dp);
}
```

Tabulation

(i) declare the dp array of same size.

(ii) changing parameter in opposite fashion.

1-based indexing instead of 0-based indexing in order to avoid -1. negative

Base case $\boxed{j==0}$ → 1

no string

s_2

$j == 0$

$\hookrightarrow 0$

```
for(i = 0 → n)
    dp[i][0] = 1
```

Code :-

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

converted back

Space Optimization :-

```
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 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];
}; }
```

single array optimization

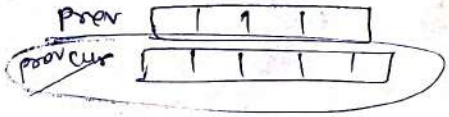
```
int numDistinct(string s, string t)
{ int n = s.size();
  int m = t.size();
  vector<double> dp(m+1,0);
  prev[0] = 1;
  for(int i=1; i<=n; i++)
  for(int j=m; j>=1; j--)
     { if(s[i-1] == t[j-1])
       { dp[j] = dp[j-1]
                 + dp[j];
       }
     }
  return (int) dp[m];
}
```

i = 1 ⟶ n

j = 1 ⟶ m

if() → curr[j] = prev[j-1] + prev[j]

else → cur[j] = prev[j]



DP-33. # Edit Distance

S_1 = "horse" S_2 = "ros"

1. Insert
2. Remove
3. Replace

The minimum no. of steps by which we can convert S_1 into S_2.

---

(1) Brute Method: delete all characters from S_1 and then insert all characters from S_2 to S_1.

it will take $len(S_1) + len(S_2)$ steps

(2) Method

replace ↑

S_1 = "horse" S_2 = "ros"

replace h with r — rorse

remove r — rose

remove e — ros

3 1

It took 3 steps.

Example

S_1 = "intention" S_2 = "execution"

remove 't' — inention

replace i → e — enention

replace n → x — exention

insert u — execution

5 operations

String matching using recursion

horse / ros

match — ok

not match — insert / delete

not match — replace

two case

replace or insert the same char

→ if not matching

→ insert of the same character

→ delete & try finding somewhere else

→ replace & match.

Try all ways → Recursion

@Aashish Kumar Nayak

How to write Recurrence?

1. Express in terms of (i, j).
2. Explore all paths of matching.
3. Return min(all paths).
4. Base case.

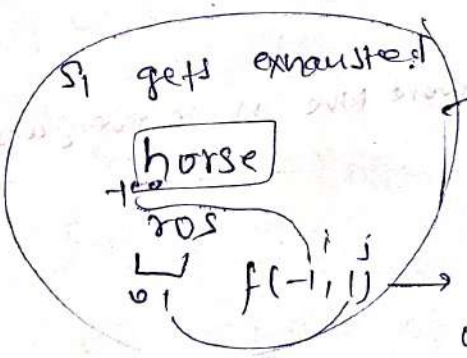
f(i, j)

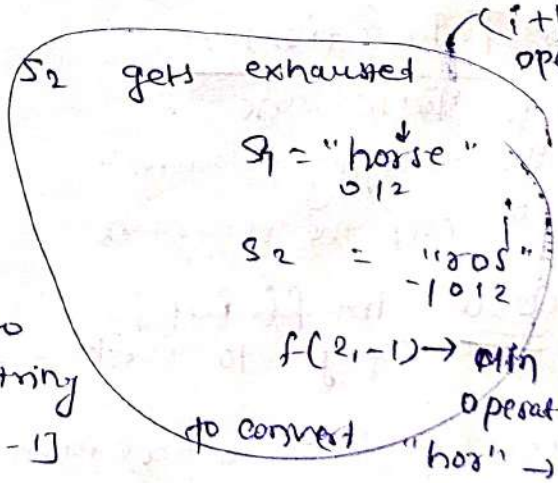
horses (i) / ros (j)

f(n-1, m-1) → min operation or to convert.

S1[0---i] to S2[0---j]

```
if( s1[i] == s2[j] )
    return 0 + f(i-1,j-1); // if they are matching shrink both string and also don't require any steps.
// if not matching i can insert
1+ f(i, j-1); //insert
1+ F(i-1, j); // delete
1+ f(i-1,j-1); // replace
```






```
if(i<0) return j+1;
if(j<0) return i+1; // insert
```

Recursion

TC → Exponential

SC → O(N+M)

overlapping subproblem → memoize

After memoization

T.C. → O(N*M)

S.C → O(N×M) + O(N+m)

```
int editDistance(string &str1, string &str2, vector<vector<int>> &dp)
{ if( i<0 ) return j+1; // if( i==0) return j+1;
  if(j<0) return i+1; // if(j==0) return i;
    if(dp[i][j] != -1) return dp[i][j];
  if (s1[i] == s2[j]) return f(i-1, j-1, s1, s2,
  //s1[i-1] == s2[j-1]                dp[i][j] =>
  return 1 + min( f(i-1,j,s1,s2,dp), min(f(i,j-1,s1,s2,dp), f(i-1,j-1,s1,s2
  dp[i][j] =>                                                    ,dp)));
}

int editdistance(string str1, string str2)
{ int n = str1.size();
  int m = str2.size();
  vector<vector<int>> dp(n, vector<int>(m, -1));
  return f(n-1, m-1, str1, str2, dp);
}
```

Tabulation :-

1. Base case —— in order to avoid the +1 to everywhere
2. changing parameters
3. copy the recurrence

Code :- ~~int f(int i~~
try to write Base case.

f(0,j) i==0 return j — Here j can be anything.

Means for(j = 0 → m) dp[0][j] = j; // Base case I

j==0 return i

for(i=0→n) dp[i][0] = i;

2. i = 1 — n

j = 1 — m

3. copy the recurance.

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

Note: Space optimization

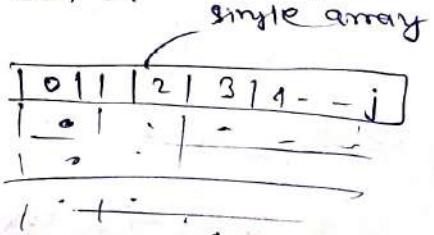
We can definitly optimise the space because there i-1, something like that is involved.

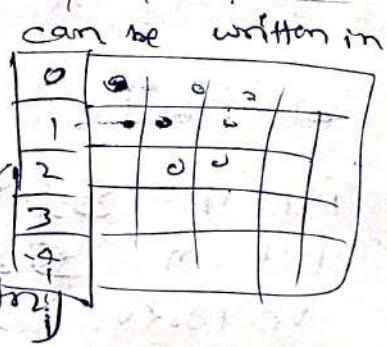
Base co

Base cases:

dp[0][j] = j

dp[i][0] = i






```
int editDistance(string str1, string str2)
{
    int n = str1.size();
    int m = str2.size();
    ~~For (vector~~
    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;
      for(int j=1; j<=m; j++)
      { if(str1[i-1] == str2[j-1])
          ~~cur~~ cur[j] = prev[j-1];
        else
          cur[j] = 1+min( prev[j],
                          min(cur[j-1],
                              prev[j-1]));
      }
      prev = cur;
    }
    return prev[m];
}
```

@Aashish kumar Nayak

Dp-34 Wildcard Matching | DP on string (last problem)

s_1 = "?ay"
s_2 = "ray" } all three chars are matching → return true

? → matches with single char.
* → matches with sequence of length 0 or more.

s_1 = "a?b*?d"
s_2 = "acbxdefxd"

return true

s_1 = "**abcd"
s_2 = "abcd"

return true

s_1 = "a?b?d"
s_2 = "acbxd"

return false

1st thing which comes in pattern string matching → we can do using recurrance

Problem faced →

ab*c?d
abdefcd } for this star we don't know how much of length of other string it will cover

so we have to try all possible ways → means recursion

len → 0, 1, 2

so we will try string matching by recursion.

Rules to write Recurance :-

1. Express (i, j)
2. Explore comparisons
3. out of all comparisons, if anyone can give return true f(n-1, m-1)

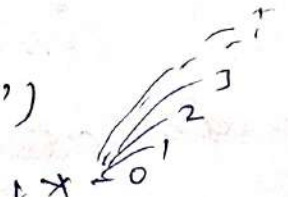
s1[0...4]
s2[0...6]

f(m-1, n-1) →
f(4,6) → (T/F)

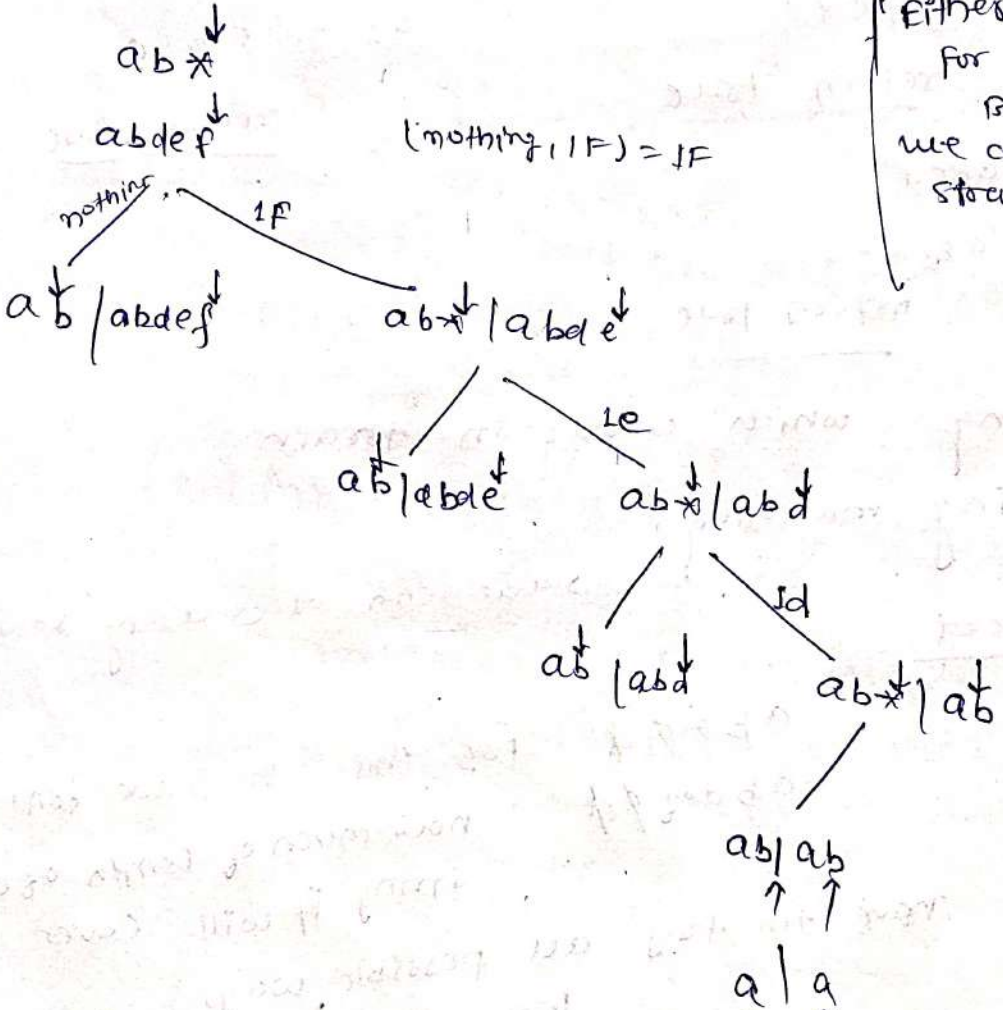
0	1	2	3	4
a	b	*	c	d

a	b	d	e	f	cd
0	1	2	3	4	5	6

```
if( s1[i] == s2[j] || s1[j] == '?')
    return f(i-1, j-1)
```



Either we can write for loop
But we can do this in straight forward recursion




```
if( s1[i] == '*')
    return f(i-1, j) or f(i, j-1)

return false;
```

Now analyze the Base case

Base case → return true/false

Either S_1 gets exhausted

S_1 → has no } Both are exhausted
S_2 → has no

if (i<0 && j<0) return true

return true

only S_1 is get exhausted

if (i<0 && j>=0).

return false

if (j<0 && i>=0) — only S_2 is get exhausted.

if S_1 is left it has to be all "*"

for (ii = 0 → i)

if (s[ii] != '*')

return false

return true.

} f(i,j)

if(i<0 && j<0) return true

if(i<0 && j>=0) return false

if(j<0 && i>=0) for (any (____ ..)

Recursion

T.C. — O(exponential)

S.C. — O(N+M)

Auxillary Stack Space

↓

After memoization

T.C. : O(N×m)

S.C. : O(N×m) + O(M+m)

↑

Ass

```
int f( int i, int j, string &pattern, string &text, vector
vector<vector<int>> &dp)

{
i==0 && j==0  if(i<0 && j<0) return true;
i==0 && j>0   if( i<0 && j>=0) return false;
j==0 && i>0   if( j<0 && i>=0)
              {
                for(int ii = 0; ii<=i; ii++)
                  { if( pattern[ii-1] != '*')
                      return false;
                  }
                return true;
              }
            if ( dp[i][j] != -1) return dp[i][j];
        if ( pattern[i-1] == text[j-1] || pattern[i-1] == '?')
          { return dp[i][j]=f(i-1, j-1, pattern, text, dp);
          }
        if( pattern[i-1] == '*')
          { return dp[i][j] = f(i-1, j, pattern, text, dp) ||
                              f( i, j-1, pattern, text, dp);
          }
      return dp[i][j] = false;
}
bool wildMatching(and (string pattern, string text)
{ int n = pattern.size();
  int m = text.size();
vector<vector<int>> dp(n, vector<int>(m, -1));
return f(n-1, m-1, pattern, text, dp);
}
```

Tabulation :-

@Aashish Kumar Nayak

1. Base case
2. changing the parameter
3. copy the recurance

1st convert it into 1-based indexing. by just increasing +1 every where

```
bool wildcardMatching(string pattern, string text)
{   int n = pattern.size();
    int 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++)
        { bool 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] = false;
    }
}
return dp[n][m];
```

Space optimization :-

```
bool wildcardMatching(string pattern, string text)
{
    int n = pattern.size();
    int m = text.size();
    vector<bool> prev(m+1, false), cur(m+1, false);
    prev[0] = true;
    for(int j = 1; j <= m; j++)
    {
        prev[j] = false;
    }
    for(int i = 1; i <= n; i++)
    {
        bool 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
            {
                cur[j] = false;
            }
        }
        prev = cur;
    }
    return prev[m];
}
```

```
        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] = false;
            }
            prev = cur;
        }
        return prev[m];
    }
```

DP-35 **Best time to buy and sell stocks** DP on Stocks

6 problems

arr[] = { 7, 1, 5, 3, 6, 4 } , n = 6

(1st day price, 2nd day price, 3rd day price)

only done once { Buy → 1, Sell → 6

Buying has to be done before selling.

if we are selling on ith day then we buy on the minimum price from 1st → (i-1)

{ 7, 1, 5, 3, 6, 4 }

keep track of minimal on left

mini = a[0], profit = 0

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

{ cost = a[i] - mini;

profit = max(profit, cost);

mini = min(mini, a[i]);

for 4 min on left is 1 so profit = 3

for 6 min on left is 1 so profit = 5

for 3 ——— 1 ——— 2

for 5 ——— 1 ——— 4

for 1 ——— 7 ——— -6

max = 5 Ans

Code:

```
int maximumprofit(vector<int>& prices)
{ int mini = prices[0];
  int maxprofit = 0;
  int n = prices.size();
  for(int i=1; i<n; i++)
  { int cost = prices[i] - mini;
    maxprofit = max(maxprofit, cost);
    mini = min(mini, prices[i]);
  }
  return maxprofit;
}
```

T.C. - O(N)

S.C. - O(1)

@Aashish Kumar Nayak

DP-36 Buy and sell stocks - II | Recursion to space optimization

Q Best time to Buy & sell stocks - II

In this question we can buy and sell many number of times.

prices[] = { 7, 1, 5, 3, 6, 4 }
B S B S
4 + 3 = 7

Buy and sell has to be done before next buy. i.e. Before buying a stock previous stock should be sold and maximise the profit

7 1 5 3 6 4
↑ ↑
Buy not Buy, Sell not sell

B S B S B S ---- A lot of ways
↓
Try all ways
↓
Recursion
↓
Best answer.

Rule for recurance :-

① Express everything in terms of index (ind, buy)
② Explore possibilities on that day
③ Take the max all profits made.
④ Base case.

f(ind, buy)

0	1	2	3	4	5
7	1	5	3	6	4

f(0,1) → start on 0th day with buy, what max profit?

find (ind, buy)

@Ashish Kumar Nayak

~~if(buy)~~ if(buy)

profit = max { price[ind] + f(ind+1, 0) } take
else { 0 + f(ind+1, 1) } not take

sell → profit = max { price[ind] + f(ind+1, 1); , 0 + f(ind+1, 0); }

return profit;

}

Base case

when we exhausted or complete the array

```
if (ind == n)
    return 0;
```

~~T.C. → O(Exponential 2^n)~~

T.C. - O(2^n)

S.C. - O(N)

There is overlapping subproblem

so apply memoization.

dp[ind][buy]

T.C. - O(

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

Tabulation :-

T.C - $O(N \times 2)$

S.C - $O(N \times 2) + \alpha N$

1. Base case
2. i → (n-1 → 0)

   Buy 0 → 1
3. copy the recurance.

~~dp[n+1]~~

```
long getmaximumprofit(long *values, int n)
{
 vector<vector<long>> dp(n+1, 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(-values[ind] +dp[ind+1][0],
                        0 +dp[ind+1][1]);
        }
       else {
          profit = max(values[ind] + dp[ind+1][1],
                        0 +dp[ind+1][0]);
        }
        dp[ind][buy] = profit;
      }
  }
  return dp[0][1];
}
```

Space optimization :

```
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(-values[ind] + ahead[0],
                     0 + ahead[1]);
      }
      else
      {
        profit = max(values[ind] + ahead[1],
                     0 + ahead[0]);
      }
      cur[buy] = profit;
    }
    ~~ahead = cur;~~ ahead = cur;
  }
  return ahead[1];
}
```

DP-37 Buy and sell stocks - III

Dream stocks

Q Best time to buy & sell stock - IV - at max 2 transaction

prices = [3, 3, 5, 0, 0, 3, 1, 4]

at max 2 transaction.

```
f(ind, buy, cap)        // at max transaction
{
  if(ind == n)
    return 0;
  if(cap == 0)
    return 0;

  if(buy)
  {
return max [-prices[ind] + f(ind+1, 0, cap),
            0 + f(ind+1, 1, cap)];
  }
  else
  {
sell it → return max[ price[ind] + f(ind+1, 1, cap-1),
                      0 + f(ind+1, 0, cap)];
```

Recursion

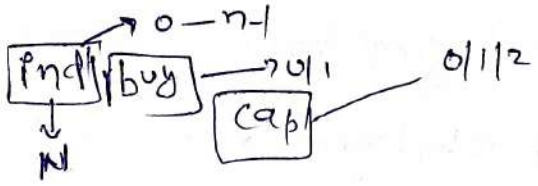
T.C. → exponential
S.C. → O(N) → ASS (Auxiliary stack space)

overlapping subproblems

↓

Memoization

@Ashish Kumar Nayak



three changing parameter | 3D Dp

dp[N][2][3] N x 2 x 3

T.C. = N x 2 x 3

S.C. = O(N x 2 x 3) + O(N)

↓ Asc for recursion

Code:

```
int f(int ind, int buy, int cap, vector<int> &prices
int n, vector<vector<vector<int>>> &dp)
{
  if(ind == n || cap == 0)
      return 0;
  if(buy)
  {
    return dp[ind][buy][cap] = max[-prices[ind] + f(ind+1, 0, cap, prices, n, dp)
                                   0 + f(ind+1, 1, cap, prices, n, dp)];
  }
  return dp[ind][buy][cap] = max[prices[ind] + f(ind+1, 1, cap-1, prices, n, dp)
                                 0 + f(ind+1, 0, cap, prices, n, dp)];
}
int maxProfit(vector<int> &prices, int n)
{ vector<vector<vector<int>>> dp(n,
    vector<vector<int>>(2, vector<int>(3, -1)));
  return f(0, 1, 2, prices, n, dp);
}
```

Tabulation

① Base case
② changing parameter ind Buy cap
③ copy the recurance

④

```
if( cap == 0 || ind == n)
        return 0;
```

if(cap == 0) ind & Buy can be anything
ind: 0 → n-1, Buy: 0 → 1

if (ind == n) cap & Buy can be anything
cap: 0 → 2, Buy: 0 → 1

```
for( ind=0 → n-1)
    for (buy → 0 → 1)
        dp[ind][buy][0] = 0

for (buy = 0 → 1)
    for (cap = 0 → 2)
        dp[0][buy][cap] = 0
```

⑤

i → run in reverse order
n-1 → 0
buy → 0 → 1
cap → 0 → 2

⑥ answer will be initial call i.e dp[0][1][2]

```
int MaxProfit(vector<int> & prices, int n)
{
vector<vector<vector<int>>> dp(n, vector<vector<int>>(2, vector<int>(3,0)));
                                  // already dp is full with 0.
```

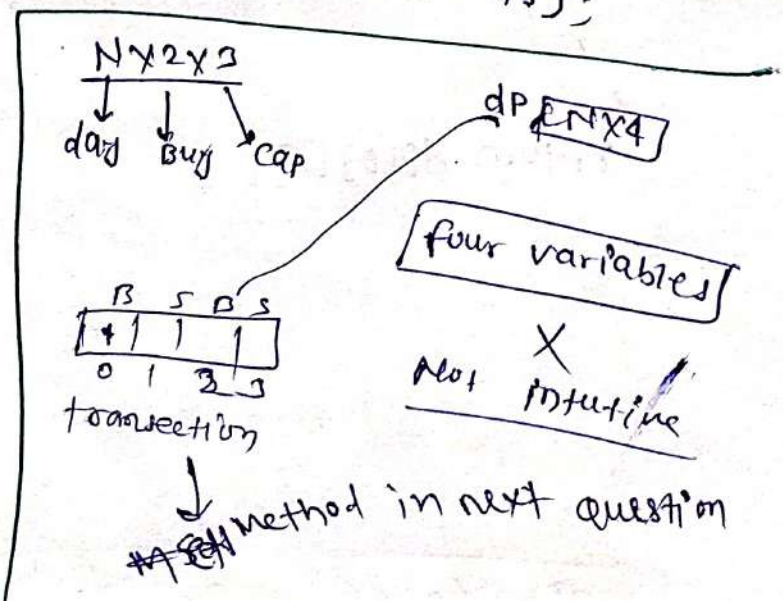
So we don't write any base cases. If there would be something different like 1, 0 then we may write.

```
for(int ind=n-1; ind>=0; ind--)
    {
    for(int buy=0; buy<=1; buy++)
        {
        for(int cap=1; cap<=2; cap++)   // ignore 0.
            {
            if(buy==1)
            {
dp[ind][buy][cap] = max(-prices[ind] + dp[ind+1][0][cap],
                        0 + dp[ind+1][1][cap]);
            }
            else
            {
dp[ind][buy][cap] = max( price[ind] + dp[ind+1][1][cap-1],
                         0 + dp[ind+1][0][cap]);
            }
        }
    }
return dp[0][1][2];
}
```

Space optimization :-

@Aashish Kumar Nayak

```
int maxprofit(vector<int>& prices, int n)
{
  vector<vector<int>> after(2,vector<int>(3,0));
  vector<vector<int>> cur(2,vector<int>(3,0));
  for(ind=n-1; ind>=0; ind--)
  {
    for(int buy=0; buy<=1; buy++)
    {
      for(int cap=1; cap<=2; cap++)
      {
        if(buy==1)
        {
          cur[buy][cap]=max(-prices[ind]+after[0][cap], 0+after[1][cap]);
        }
        else
        {
          cur[buy][cap]=max(prices[ind]+after[1][cap-1],
                            0+after[0][cap]);
        }
      }
    }
    after=cur;
  }
  return after[1][2];
}
```



DP-38 Buy and Sell Stocks - IV | DP on Stocks

At max K transactions

K can be anything

Code is Similar only change is 1< Replace 2 with 1<

Code (Space Optimization)

```
int maxprofit(vector<int> &prices, int n, int k)
{ vector<vector<int>> after(2,vector<int> (k+1,0));
  vector<vector<int>> cur(2, vector<int>(k+1,0));
  for( int ind =n-1; ind >=0; ind--)
     { for(int buy = 0; buy <=1; buy++)
        { for( int cap = 1; cap <=k; cap++)
            { if( buy ==1)
               { cur[buy][cap] =
              max(-prices[ind] + after[0][cap], 0 + after[1][cap]);
               }
          else
        }
    cur[buy][cap] = max( prices[ind] + after[1][cap],
                         0 + after[0][cap]);
        }
     }
    after = cur;
    }
    return after[1][k];
    }
```

@Aashish kumar Nayak

Code

```
int f(int ind, int tranNo, vector<int>& prices, int n, int k,
    vector<vector<int>> &dp)
{ if( ind == n || tranNo == 2*k)
      return 0;
  if(dp[ind][tranNo] != -1)
      return dp[ind][tranNO];
  if( tranNo % 2 == 0)
  {
    return dp[ind][tranNO] = max(-prices[ind] +
                f(ind+1, tranNo+1, prices, n, K, dp),
            0 + f(ind+1, tranNo, prices, n, K, dp));
  }
  return dp[ind][tranNO] = max( prices +
  f(ind+1, tranNo+1, prices, n, K, dp), 0+f(ind+1, tranNo,
  prices, n, K, dp));
}

int maximumprofit(vector<int> &prices, int n, int k)
{ vector<vector<int>> dp(n, vector<int>(2*k, -1));
  return f(0, 0, prices, n, K, dp);
}
```

ie Tabulation

```
int maxiprofit(vector<int>& prices, int n, int k)
{
 vector<vector<int>> dp(n+1, vector<int>(2*k+1,0));
 for(int ind = n-1; ind>=0; ind--)
   { for(int tranNO = 2*k-1; tranNO >=0; tranNO--)
      { if(tranNO %. 2 == 0)
          }
    dp[ind][tranNO] = max(-prices[ind] + dp[ind+1][tranNO+1],
                          dp[ind+1][tranNO]);
   }
   else
   {
    dp[ind][tranNO] = max(prices[ind] + dp[ind+1][tranNO-1],
                          dp[ind+1][tranNO]);
   }
}
return dp[0][0];
} Space
```

Space optimization :-

```
int maxprofit(vector<int> &prices, int n, int k)
{
    vector<int> after(2*k+1, 0);
    vector<int> cur(2*k+1, 0);
    for(int ind = n-1; ind >=0; ind--)
    {
        for(int tranNO = 2*k-1; tranNO >=0; tranNO--)
        {
            if(tranNO%2 ==0)
            {
                cur[tranNO] = max(-prices[ind] + after[tranNO+1],
                                  0 + after[tranNO]);
            }
            else
            {
                cur[tranNO] = max(prices[ind] + after[tranNO+1],
                                  0 + after[tranNO]);
            }
        }
        after = cur;
    }
    return after[0];
}
```

T.C. ≈ O(2K)
≈ O(K)

S.C.

DP-39 Buy and sell stocks with cooldown

| 5th problem
DP on stocks

Cooldown → can't Buy right after sell.

price [] → {4, 9, 0, 4, 10}.

5 + 10 = 15

Cooldown {4, 9, 0, 4, 10}
B S X B
B ✓

```
f(ind, Buy)
{
  if(ind == n) return 0

  if(Buy)
return max { -price[ind] + f(ind+1, 0),
             0 + f(ind+1, 1) },
  else
return max ( price[ind] + f(ind+1, 1),
             0 + f(ind+1, 0) );
```

Buy and stock - II
→ unlimited time
attach cooldown condition on this problem

ind+2 → that day we can buy.

only this minor change on DP on stocks II problem we solved in DP on stock ✓

@Aashish Kumar Nayak

```
int f(int ind, int buy, vector<int> &prices, vector<vector<int>> &dp)
{
    if(ind >= prices.size())
        return 0;  if(dp[ind][buy] != -1) return dp[ind][buy];
    if(buy == 1)
    return dp[ind][buy] = max(-prices[ind] + f(ind+1, 0, prices, dp),
                              0 + f(ind+1, 1, prices, dp));
    else
    return dp[ind][buy] = max(prices[ind] + f(ind+2, 1, prices, dp),
                              0 + f(ind+1, 0, prices));
}
int maxprofit(vector<int> &prices)
    {   int n = prices.size();
        vector<vector<int>> dp(n, vector<int>(2,-1));
        return f(0, 1, prices, dp);
    }
```

Tabulation :-

```
int maxProfit(vector<int> &prices)
    {   int n = prices.size();
        vector<vector<int>>dp(n+2, vector<int> dp(n, vector<int>(2,0));
        for(int ind=n-1; ind>=0; ind--)
        {
            for( int buy=0; buy<=1; buy++)
            {   if( buy == 1)
                dp[ind][buy] = max(-price[ind] + dp[ind+1][0]
                                   0+f(ind+1, 1, prices, dp));
                dp[ind+1][1]);
            else
            {
                dp[ind][buy] = max(prices[ind] + dp[ind+2][1],
                                   0 + f(ind+1, 0, prices, dp));
                dp[ind+1][0])
            }
        }
        return dp[0][1];
    }
```

@Ashish kumar Nayak

more optimation


```
int maxprofit(vector<int>& prices)
{
    int n = prices.size();
    vector<int> front2(2,0);
    vector<int> front1(2,0);
    vector<int> cur(2,0);
    for(int ind = n-1; ind>=0; ind--)
    {
        cur[1] = max(-prices[ind] + front1[0],
                     0 + front1[1]);
        cur[0] = max(prices[ind] + front2[1],
                     0 + front1[0]);
        front2 = front1;
        front1 = cur;
    }
    return cur[1];
}
```

T.C- O(N)
S.C- O(6) ≈ O(1)

@Aashish kumar Nayak

DP-40 Buy and sell stocks with Transaction fee

DP on stocks
6th problem

prices[] = {1, 3, 2, 8, 4, 9} fee = 2

B S B S

7-2 + 5-2 = 12-4 = (8) Net profit.

f(ind, buy)

{

if(ind == 0) return 0;

if(buy == 1)

max(-prices[ind] - fee + f(ind+1, 0), 0 + f(ind+1, 1));

else

max((price[ind] - fee) + f(ind+1, 1), 0 + f(ind+1, 0));

After the sell is done deduct the fee or we can also deduct the fee during buying.

Code :

```
int maximumprofit(int n, int fee, vector<int> &values)
{
    int aheadNotBuy, aheadBuy, curbuy, currNotbuy;
    aheadbuy aheadNotbuy = aheadbuy = 0;
    for(int ind = n-1; ind >= 0; ind--)
    {
        //sell
        currNotbuy = max(values[ind] + aheadbuy, 0 + aheadNotbuy);
        // buy
        curbuy = max(-values[ind] - fee + aheadBuy, 0 + aheadBuy);
        aheadBuy = curBuy;
        aheadNotBuy = currNotBuy;
    }
    return aheadBuy;
}
```

@Ashish Kumar Nayak

DP-41 Longest increasing Subsequence

arr[] : [10, 9, 2, 5, 3, 7, 101, 18]

{2, 3, 7, 101} → len=4

{2, 3, 7, 18} → len=4

arr[] = {8, 8, 8}

{8} → len = 1

Brute force

Let's try out various subsequences → power set (or) Recursion

in order to generate all subsequence.

(2^n) exponential — Check for increase

Store the longest.

Trying all ways

↳ Recursion

[10, ~~9~~]

prev=10 to check

Recurrence

(i) Express everything in terms of index

(ii) Explore all possibilities in subsequence — ✓ ✗

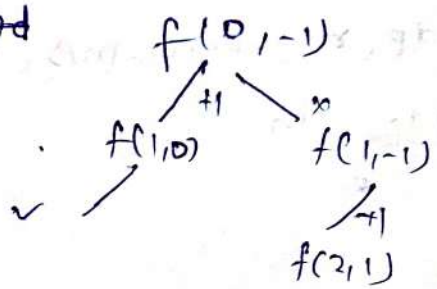
(iii) Take the max length

f(ind, prev_ind)

f(0,-1) → length of LIS starting from 0.

f(3,0) → length of LIS starting from ind index whose previous index is 0. → prev

f(ind



Base case :-

```
if( ind == n)
    return 0;
```

~~Recc~~ Recursion.

T.C. $O(2^n)$ for every index we have 2 options i.e $O(2^n)$ .. S.C - $O(N)$

take \ not-take

↓ overlapping sub-problem

Memoization

f(ind, prev_ind)

ind — 0 — (0-1)

Co-ordinate change :

0 1 2 3

T.C. - O(N X N)

S.C - O(N X N) + O(N)

code :-

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

Tabulation :

Rules.

(i) Base case ~~~ dp[n][n+1] = 0
∴ we already assign 0 to all so no need to write base case.

(ii) Changing parameter
ind = n-1 → 0
prev_ind = ind-1 → -1

(iii) copy the recurance
follow coordinated shift.

Code :

```
int longestIncreasingSubsequence(int arr[], int n)
{
  vector<vector<int>> dp(n+1, 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];
}
```

@Aashish Kumar Nayak

Space optimization :-

@Aashish Kumar Nayak

```
int longestIncreasingSubsequence(int arr[], int n)
{
    vector<int> next(n+1,0), cur(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+next[prev_ind+1];
            if(prev_ind==-1||arr[ind]>arr[prev_ind])
            {
                len=max(len, 1+next[ind+1]);
            }
            cur[prev_ind+1]=len;
        }
        next=cur;
    }
    return next[-1+1];
}
```

T.C. - $O(N^2)$

S.C. - $O(N) \times 2$

Example

5	4	11	1	16	8

dp[n]

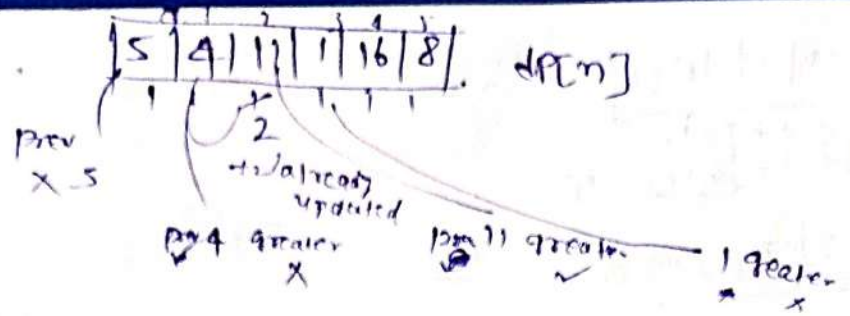
dp[i] → signifies the longest increasing subsequence that ends at index i.

{5, 16}

{4, 11, 16}

Ans → max of all dp index.

$i = 0$ to $n-1$ max(dp[i])




```
for (ind = 0 → n-1)
    for (prev = 0 → ind-1)
        if (curr[prev] < arr[ind])
            dp[ind] = max(1+dp[prev], dp[ind]);
```

Code :-

```
int longestIncreasingSubsequence(int arr[], int n)
{
    vector<int> dp(n,1);
    int maxi = 1;
    for(int i = 0; i<n; i++)
    {
        for(int prev = 0; prev<i; prev++)
        {
            if(arr[prev] < arr[i])
            {
                dp[i] = max(dp[i], 1+dp[prev]);
            }
        }
        maxi = max(maxi, dp[i]);
    }
    return maxi;
}
```

T.C. - $O(N^2)$
S.C. - $O(N)$

This solution will be required if we want to trace the LIS.

DP-42

print the subsequence

5	7	11	1	16	8
1	2	2	1	3	1	dp

| 0 | 1 | 2 | 3 | 4 | 5 | hash

4 → 2 → 0

16 11 5

reverse it → 5, 11, 16

Code :-

```
int longestIncreasingSubsequence(int arr[], int n)
{
  vector<int> dp(n,1), hash(n);
  int maxi = 1;
  int 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> temp;
temp.push_back(arr[lastIndex]);
while(hash[lastIndex] != lastIndex)
    {
        lastIndex = hash[lastIndex];
        temp.push_back(arr[lastIndex]);
    }
reverse(temp.begin(), temp.end());
for(auto it : temp)
    cout<<it<<" ";
cout<<endl;
return maxi;
}
```

T.C. - $O(N^2)$

DP-43 Longest Increasing Subsequence

In this lecture we will use Binary search

LIS using Binary search

[1, 7, 8, 4, 5, 6, -1, 9]

Increasing subsequences are

1, 4, 5, 6, 9 — longest

Approach :- 1 ~~4, 6, 9~~

Try to go every element and form subsequence

[1, 7, 8, 4, 5, 6, -1, 9]

[1, 7, 8, 9] len=4

[1, 4, 5, 6, 9] len=5 → longest

[-1, 9] len=2

— But a lot of space will be consumed

Intuition

- To avoid space we will overwrite instead of creating new array.

[1, 7, 8, 4, 5, 6, -1, 9]

[1, 4, 5, 6, 9]

[1, 4 — instead of making [1, 4 we will replace new overwrite in 1st array Replace 4 with 7

arr[] = {1, 4, 5, 4, 2, 8}

[1, 2, 4, 5, 8] len = 4 — space has been optimized.

Recursion in previous lec
Memoization
space optimization
Tabulation
Algorithmic approach

T.C — O(N²)
S.C. — O(N)
Not more

@Aashish Kumar Nayak

Binary Element el -> arr[i]

if(find arr[i] .

else find 1st greater of arr[i]

ed

lower_bound( )

→ it gives you 1st index of arr[i]
or
first index > arr[i]

Code :-

```
int longestIncreasingSubsequence(int arr[], int n)
  {  vector<int> temp;
     temp.push_back(arr[0]); int len = 1;
     for(int i=1; i<n; i++)
        { if(arr[i] > temp.back())
            { temp.push_back(arr[i]);
              len++;
            }
          else
            {
              int ind = lower_bound(temp.begin(), temp.end(),
                                    arr[i]);
              temp[ind] = arr[i];
            }
        }
     return temp.size();
  }  return len; // len will have less space than temp
```

lower bound.

size().

T.C. - O(N log N)
S.C. - O(N)

@Aashish kumar Nayak

DP-44 Largest Divisible Subset | longest increasing Subsequence

@Aashish kumar Nayak

arr[] = {1, 16, 7, 8, 4}

{1, 16, 8} — subsequence

{1, 8, 16}
{1, 16, 8}
{1, 7, 16}
} Subset not necessary to follow the order.

arr[i] % arr[j] == 0

or arr[j] % arr[i] == 0

lets take an example

{16, 8, 4} (16,8) (8,4) (16,4) Every pair is divisible

{1, 16, 8, 4} (1,16), (1,8), (1,4), (16,8), (16,4), (8,4)

All the pairs are divisible this can be possible ans

len = 4

print any answer you can jumble it and return.

{1,16,8,4} or {4, 16, 1, 8} ✓ ✓

intution

sort the array

{1, 4, 7, 8, 16}

{1, 4, 8, 16}

∵ 4 is divisible by 1 so insert it
7 is not divisible by 4.
∵ 8 is divisible by 4 and also if it is divisible by 4 then it will automatically divisible by 1 or previous elements.

len = 4

we will get longest Divisible increasing subsequence.

LIS code :-

```
for(i=1; i<n; i++)
  for(j=0; j<i; j++)
    if(arr[i] % arr[j] == 0)
      if(arr[i] < arr[i] && dp[i] < dp[j]+1)
        dp[i] = dp[j]+1
        hash[i] = j
```

because it's already sorted just check arr[i] % arr[j]; or arr[j] % arr[i];

Code :-

```
vector<int> divisibleSet(vector<int> &arr)
{
    int n = arr.size();
    vector<int> dp(n,1), hash(n);
    int maxi = 1;
    int lastIndex = 0;
    sort(arr.begin(), arr.end());
    for(int i=0; i<n; i++)
    {
        hash[i] = i;
        for(int prev=0; prev<i; prev++)
        {
            if(arr[i] % arr[prev] == 0 && 1+dp[prev] > dp[i])
            {
                dp[i] = 1+dp[prev];
                hash[i] = prev;
            }
        }
        if(dp[i] > maxi)
        {
            maxi = dp[i];
            lastIndex = i;
        }
    }
    vector<int> temp;
    temp.push_back(arr[lastIndex]);
    while(hash[lastIndex] != lastIndex)
    {
        lastIndex = hash[lastIndex];
        temp.push_back(arr[lastIndex]);
    }
    reverse(temp.begin(), temp.end());
    return;
}
```

T.C. $O(N^2)$

S.C - $O(N)$

@Aashish Kumar Nayak

DP-45 Longest String Chain | Longest increasing subsequence

words[] = {"a", "b", "ba", "bca", "bda", "bdca"}

{a, ba, bda} {b, ba, bca}

{a, ba, bca}

```
for(i=1 ⟶ n)
{ for (j=0 ; j<i ; j++)
    {
      if(arr[i]>arr[j] && dp[j]+1 > dp[i])
      { dp[i] = dp[j]+1;
      }
    }
    maxi = max(maxi, dp[i]);
return maxi;
```

one line change.

Compare(arr[i], arr[j])

bdca

bda

assume

arr[i] bcda ← +a arr[j] bcd

bcd

if both pointers reaches its end means match.

In this question there is sequence NOT sub-sequence so we can pick from any where.

["xbc", "pexbcr", "xb", "rxbc", "pcxbc"]

we will sort this array according to the length.

write comparator

```
bool comp(string &s1, string &s2)
    return s1.size() < s2.size();
```

@Aashish Kumar Nayak

Code :-

```
int longestStrChain(vector<string> &arr)
{
  sort(arr.begin(), arr.end(), comp);
  int n = arr.size();
  vector<int> dp(n, 1);
  int maxi = 1;
  for(int i=0; i<n; i++)
  {
    for(int prev=0; prev<i; prev++)
    {
      if(checkpossible(arr[i], arr[prev])
         && 1+dp[prev] > dp[i])
      {
        dp[i] = 1+dp[prev];
      }
    }
    if(dp[i] > maxi)
    {
      maxi = dp[i];
    }
  }
  return maxi;
}

bool comp(string &s1, string &s2)
{
  return s1.size() < s2.size();
}

bool checkpossible(string &s1, string &s2)
{
  if(s1.size() != s2.size()+1) return false;
  int first = 0;
  int second = 0;
  while(first < s1.size())
  {
    if(s1[first] == s2[second])
    {
      first++; second++;
    }
    else { first++; }
  }
  if(first == s1.size() && second == s2.size()) return true; else return false;
}
```

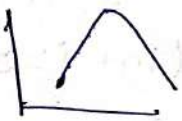
length of the string

T.C. - O(N^2)xl + O(nlogn) / sorting.

DP-46 Longest Bitonic Subsequence | LIS

arr[] : {1, 11, 2, 10, 4, 5, 2, 1}

Bitonic ⟶ increasing then decreasing. or it can be just increase or just decrease

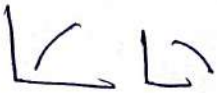


{1, 2, 10, 4, 2, 1}

{1, 2, 10, 5, 2, 1}

{1, 11, 10, 4, 2, 1}

{1, 11, 10, 5, 2, 1}



arr[] : {1, 10, 2, 3, 4, 5, 6}

because of subsequence {1, 2, 3, 4, 5, 6} len=6

arr[] = {5, 4, 3, 100, 2, 1}

{5, 4, 3, 2, 1} — len = 5

Both of them are also Bitonic

```
for(i=0; i<n; i++)
    dp[i]=1
    for(j=0; j<i; j++)
        if(arr[i]>arr[j] && dp[j]+1>dp[i])
            dp[i] = 1+dp[j];
```

↑ longest increasing subsequence till index i.

arr[] = {1, 11, 2, 10, 4, 5, 2, 1}

dp1[] = {1, 2, 2, 3, 3, 4, 2, 1}

dp2[] = {1, 2, 5, 3, 4, 3, 3, 2, 1}

10 is common

2 + 4 − 1 = 6 Ans

2 + 4 = 7 Ans

@Aashish Kumar Nayak

Code:-

```
int longestBitonicSequence(vector<int> &arr, int n)
{ vector<int> dp1(n,1);
  for(int i=0; i<n; i++)
  { for(int prev=0; prev<i; prev++)
    { if(arr[prev] < arr[i] && 1+dp1[prev] > dp1[i])
      { dp1[i] = 1+dp1[prev];
      }
    }
  }
  int maxi=0;
  vector<int> dp2(n,1);
  for(int i=n-1; i>=0; i--)
  { for(int prev=n-1; prev>i; prev--)
    { if(arr[prev] < arr[i] && 1+dp2[prev] > dp2[i])
      { dp2[i] = 1+dp2[prev];
      }
    }
    maxi = max(maxi, dp1[i]+dp2[i]-1);
  }
```

avoid this by using this ↓

```
  int maxi=0;
  for(int i=0; i<n; i++)
  { maxi = max(maxi, dp1[i]+dp2[i]-1);
  }
  return maxi;
}
```

Same O(N) T.C. (B,D)

T.C. - O(N)

@Ashish Kumar Nayak

DP-47 Number of longest increasing subsequence

arr[] = {1, 8, 5, 4, 7}

{1, 3, 4, 7}
itself + its LIS {1, 3, 5, 7} — ② subsequences of longest length

arr[] → {1, 8, 5, 4, 3, 2, 6, 7, 10, 8, 9}
(5 attached with 1)

dp[] → 1 x2 x2 x2 x2 x3 3
cnt[] → 1 1 1 1 1 2 3 4 5

1, 5

↳ itself of length 1 and cnt is 1, 1 can be attached with 4 so increase cnt by 1 and dp remain

1,5,7 1,6,7
1,4,7 5,6,7
1,3,7 4,6,7
1,2,7 3,6,7
1,6,7 2,6,7

{1, 3, 5, 4, 7}
dp[] = 1, 2, 3, 3, 4
cnt[] = 1, 1, 1, 1, 2

Code :-
```
int findNumberOfLIS(vector<int> &arr)
{ vector<int> dp(n,1), cnt(n,1);
  int maxi = 1; int n = arr.size();
  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];
        cnt[i] = cnt[j];
```

```
else if (arr[prev] < arr[i] && 1 + dp[prev] == dp[i])
{
    cnt[i] += cnt[prev];
}
}
maxi = max(maxi, dp[i]);
}
int cnt = 0;  int nos = 0;
for(int i = 0; i < n; i++)
{ if(dp[i] == maxi)
    nos += cnt[i];
}
return nos;
}
```

DP - 48 Matrix chain Multiplication [MCM] Partition DP starts

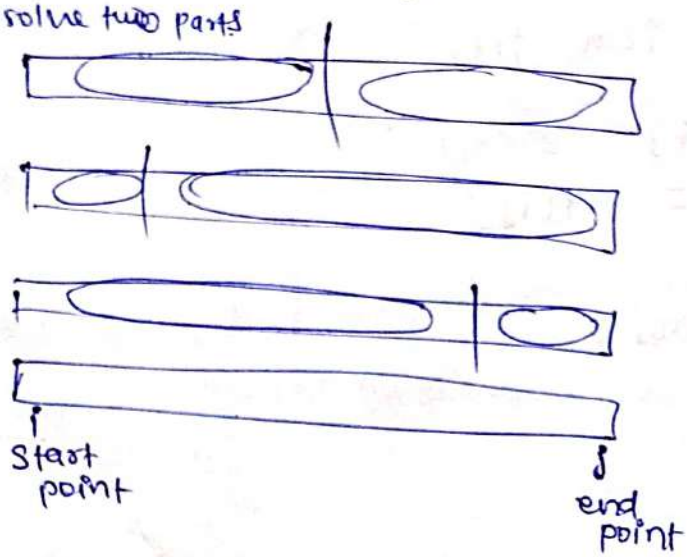
New pattern - [pattern DP] Tough pattern

Solve a problem in a pattern

$(1+2+3)\times(5)$

$(1+2)+(3\times5)$

We have to solve two parts



MCM → matrix chain multiplication

B = 30x5

L = 5x60

A [ ]$_{10\times30}$ B [ ]$_{30\times5}$

= 10 x 30 x 5 = 1500 operations required to multiply the matrix.

$\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}_{2\times2} \times \begin{bmatrix}2\\ 3\end{bmatrix}_{2\times1} = \begin{bmatrix}1\times2 + 2\times3\\ 3\times2 + 1\times3\end{bmatrix}$

$2\times2\times1$ = 4 operations

(A B C)

(A B) C

10×30 30×5

A = 10x30

B = 30x5

C = 5x60

[ ]$_{10\times5}$ [ ]$_{5\times60}$ + (10x5x60)

= 1500 + 3000 = 4500

@Ashish Kumar Nayak

A(BC)

(30x60) 5x60

9000

[]10x30 []30x60

= 10x30x60 + 9000

= 18000 + 9000

= 27000

The minimum operation required given the N matrix dimension

ABCD

A(B(CD)) AB(CD)

Given arr[] → [10, 20, 30, 40, 50]

↓ dimensions of N-1 matrixes N=5

A → 10x20, B → 20x30, C → 30x40, D → 40x50

1st → A[0]xA[1] 2nd A[1]xA[2] 3rd → A[2]xA[3] ith → A[i-1]xA[i]

Tell me the no. of operations required to multiply the Matrices.

Approach Partition DP

take the 4 guys and give one - the ans.

ABCD

(AB)(CD) (A)(BCD) (ABC)(D) → take the minimal of them

1. Start with entire block/array F(i, j) start point end point
2. Try all partition → Run a loop to try all partitions
3. Return the best possible 2 partition

[10, 20, 30, 40, 50]

A B C D

i j

↓ ↓

1 n-1

f(1, 4) → return the minimum multiplications to multiply matrix 1 → 4

Base case :-

```
f(i,j)
{ if(i==j)
    return 0;
```

Try all partions

{ 10, 20, 30, 40, 50 }
(A) (B C D)

( ) ( )

( ) ( )

$K = (i \xrightarrow{1} j-1)$ (3)

$K=1$ $f(i,K)$, $f(K+1,j)$ (1, 2, 4)

$K=2$ $f(1,2)$, $f(3,4)$

$K=3$ $f(1,3)$, $f(4,4)$

$K = (i+1 \rightarrow j)$

$f(i,K-1)$, $f(K,j)$

for ($K \rightarrow i \rightarrow j-1$)

{ steps = A[i-1] x arr[K] x arr[j]) + f(i,K) + f(K+1,j)

f(1,1) f(2,4) (BCD)

A = 10x20

B = 20x30

C = 30x40

D = 40x50

(10x20) (20x50)

(10x50) 10x20x50 operation or steps

B C D

(20x30) (30x40) (40x50)

B C D

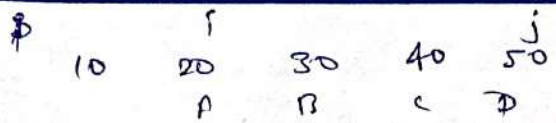
20x30 30x50

(20x50)

(20x40) (40x50)

(20x50)

Aashish Kumar Nayak



f(AB) + f(CD) +

10×20 30×30 40×50

20×30

30×50

10×30

10×50

No. of steps

= 10×50×30

A[i-1] × A[k] × A[j]

Recursion Tree

(10, 20, 30, 40, 50)

A B C D

i j

single matrix No multiply

A [20] [30, 40, 50]

i j i j

i j B C D

AB CD A B C D

20 30 20 30 40 50 50

i j i j i j i j

(10 * 20 × 30) + f + f

B C D

30 40 50

i j

[B] [CD] [BC] [D]

Then take min of all.

Code

Code :=

```
int f(int i, int j, vector<int> &arr)
{ if(i == j) return 0;
int mini = 1eg;
for(int k = i; k < j; k++)
  { int steps = arr[i-1] * arr[k] * arr[j] + f(i,k,arr,dp)
                + f(k+1, j, arr,dp);
    if( steps < mini )
        mini = steps;
  }
  return mini; return dp[i][j] = mini;
}

int matrixMultiplication( vector<int> &arr, int N)
{ vector<vector<int>> dp(N, vector<int>(N, -1));
  return f(1, N-1, arr, dp);
}
```

T.C. - Exponential

Apply Memoization

changing variables f(i,j)

DP-49 Matrix chain multiplication | Bottom-up | Tabulation

Rules :-

1. copy the base case
2. write ~~row~~ changing states.

~~f(i,j)~~

dp[N][N]

~~for~~ i = (n-1 ⟶ 1)

j :- ___

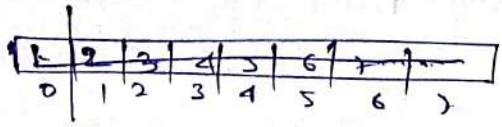
Code :-

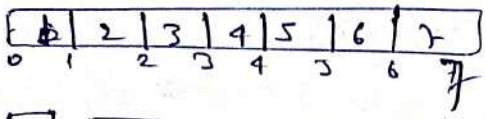
```
int matrixmultiplication(vector<int> &arr, int N)
{
    int dp[N][N];
    for(int i=N-1; i>=1; i--)
    {
        for(int j= i+1; j<N; j++)
        {
            int mini = 1e9;
            for(int k= i; k<j; k++)
            {
                int steps = arr[i-1]*arr[k]*arr[j]
                 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

DP-50 Minimum cost to cut the stick

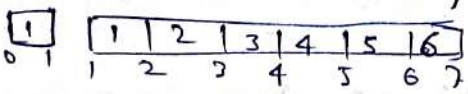
arr[] = [1, 3, 4, 5] n = 7

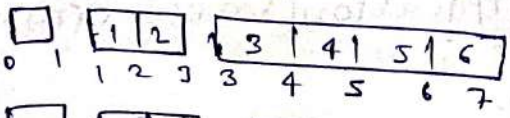
cost = length of the stick you are cutting

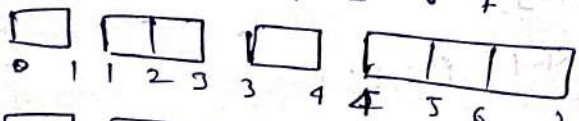


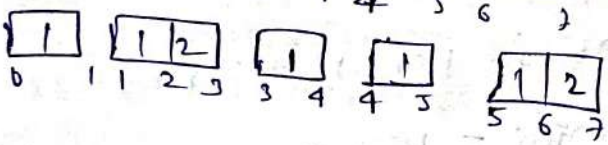


7 + 6 + 4 + 3 = 20

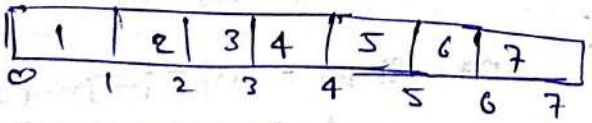


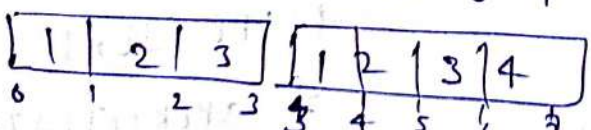




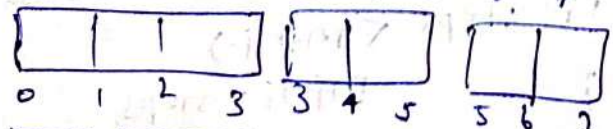


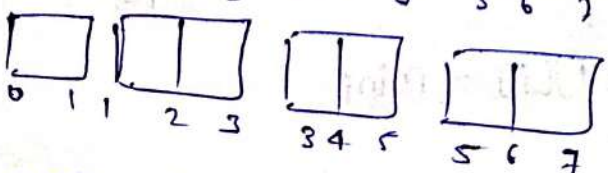
{ 3, 5, 1, 4 }



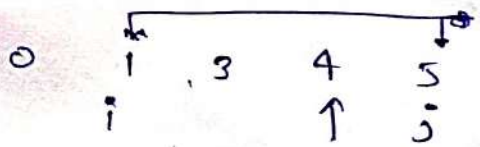


7 + 4 + 3 + 3 = 16





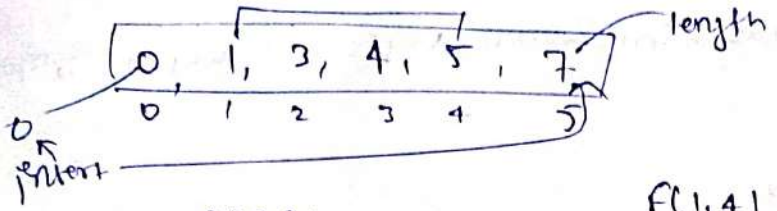
array is sorted



cost = length of that stick in which 4 is.....

cuts[j+1] − cuts[i−1]

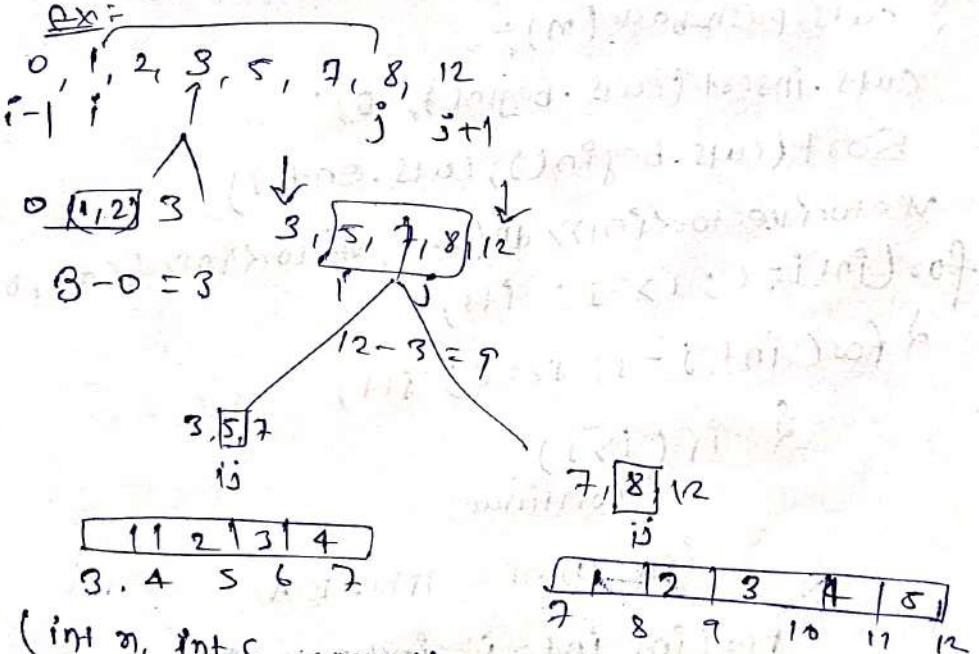
7 − 0 = 7



f(i,j) f(1,4)

```
if(i>j)
   return 0;
mini = INT_MAX;
for(ind = i → j)
{ cost = cuts[j+1] - cuts[i-1]
        + f(i, ind-1) + f(ind+1, j)
  mini = min(mini, cost);
}
return mini;
}
```

Ex:-



Code

```
int cost(int n, int c, vector<int> &cuts,
{ cuts.push_back(n);
  cuts.insert(cuts.begin(), 0);
  sort(cuts.begin(), cuts.end());
  return f(1, c, cuts, dp);
}
```

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

@Aashish Kumar Nayak

```
int f(int i, int j, vector<int>& cuts, vector<vector<int>>& dp)
{ if(i > j) return 0;
    if(dp[i][j] != -1)
        return dp[i][j];
    int mini = INT_MAX;
    for(int ind = i; ind <= j; ind++)
    { int cost = cuts[j+1] - cuts[i-1] + f(i, ind-1, cuts, dp)
                 + f(ind+1, j, cuts, dp);
      mini = min(mini, cost);
    }
    return dp[i][j] = mini;
}
```

T.C. $(M^2 \times M) \simeq O(M^3)$

S.C. $= O(M^2) + \simeq O(M^3)$ Stack space Ass

Tabulation :-

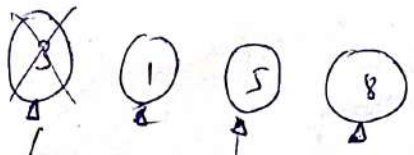
```
int cost(int n, int c, vector<int>& cuts)
{ cuts.push_back(n);
  cuts.insert(cuts.begin(), 0);
  sort(cuts.begin(), cuts.end());
  vector<vector<int>> dp(c+2, vector<int>(c+2, 0));
  for(int i = c; i >= 1; i--)
  { for(int j = 1; j <= c; j++)
    { if(i > j)
        continue;
      int mini = INT_MAX;
      for(int ind = i; ind <= j; ind++)
      {
        int cost = cuts[j+1] - cuts[i-1] + dp[i][ind-1] +
                   dp[ind+1][j];
        mini = min(mini, cost);
      }
      dp[i][j] = mini;
    }
  }
  return dp[1][c];
}
```

@Aashish kumar Nayak

DP-51 Brust Ballons | partion DP

arr[] = {3, 1, 5, 8} n = 4

Summation of those coins ↑↑



if we burst 5 → 1 (Left) × 5 × 8 (right) = 40

1 × 3 × 1 = 3

for 1, now 3 is already burst

1 × 1 × 5 = 5

for 5

1 × 5 × 8 = 40

[8] 1 × 8 × 1 = 8

1	[3, ~~1~~, 5, 8]	3 × 1 × 5 = 15
5	[3, ~~5~~, 8]	3 × 5 × 8 = 120
3	[~~3~~, 8]	1 × 3 × 8 = 24
8	[8]	1 × 8 × 1 = 8
		167 → maximum

~~Maximal~~ ~~start from~~ ~~min~~ value ~~to burst the~~ Balloon.

[3, 1, 5, 8]

i ... j can be anyone. so we can think as MCM

[b_1, b_2, b_3, b_4, b_5, b_6]

$(b_3 \times b_4 \times b_5)$ + $(b_1 b_2 b_3)$ + $(b_5 b_6)$

Subproblems

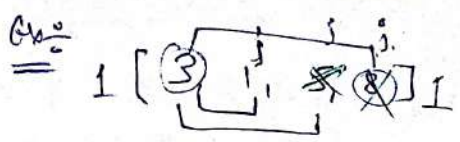
This will not work if solving independently

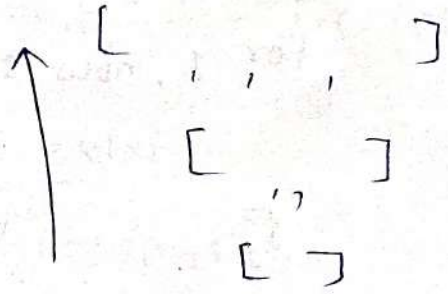
[b1, b2, b3, b4, b5, b6]

{ b1, b2, b3, b5, b6 }

if we take subpart as an individual subproblem and try to solve for b3 then we can't become it depends on other portion b4. i.e. b4.

start thinking in opposite direction





[3, X, 8]

[X, 8]

[8]

3 x 1 x 5 = 15

3 x 5 x 8 = 120

→ 1 x 3 x 8 = 24

← 1 x 8 x 1 = 8

{ b1, b2, b3, b4, b5, b6 }

i X ind j

= (a[i-1] x a[ind] x a[j+1]) + f(j, ind-1) + f(ind+1, j)

{b1, b4} or {b2, b4} or {b3, b4} or 2nd last Burst

[ ]

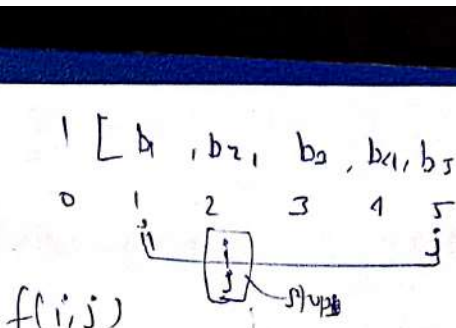
[ ]

[ ]

[b4]

= 1 x b4 x 1

(last Burst)




```
f(i,j)
{
  if( i >= j )
      return 0;
  for( ind = i -> j )  maxi = INT_MIN;
  {
    cost = a[i-1] x a[ind] x a[j+1] + f(i, ind-1) + f(ind+1, j)
    mini = MIN(mini, cost);
  }
  return mini;
```

i = j − 1 subproblem (last subproblem)

Recursion

T.C. ⟶ T.C. — Exponential

↓ Memoization

T.C. → N × N × M ≃ N^3

S.C. → $O(N^2)$ + $O(N)$ ASC

[N+1]

Code :-

```
int f( int i; int j, vector<int> & a , vector<vector<int>> &dp)
{  if(i>j) return 0;
   if( dp[i][j] != -1)
        return dp[i][j];
   int maxi = INT_MAX;
   for(int ind = i; ind <= j ; ind++)
      { int cost = a[i-1] * a[ind] * a[j+1] +
                   f(i, ind-1, a, dp) + f(ind+1, j, a, dp);
        maxi = max(maxi, cost);
      }
   return dp[i][j] = maxi;
}
```

```
int n = a.size();
a.push_back(1);
a.insert(a.begin(), 1);
vector<vector<int>> dp(n+1, vector<int>(n+1, -1));
return f(1, n, a, dp);
}
```

Tabulation :

④

```
int maxCoins(vector<int>& a)
{
    int n = a.size();
    a.push_back(1);
    a.insert(a.begin(), 1);
    vector<vector<int>> dp(n+2, vector<int>(n+2, 0));
    for(int i = n; i >= 1; i--)
    {
        for(int j = 1; j <= n; j++)
        {
            if(i > j)
                continue;
            int maxi = INT_MIN;
            for(int ind = i; ind <= j; ind++)
            {
                int cost = a[i-1] * a[ind] * a[j+1]
                    + dp[i][ind-1] + dp[ind+1][j];
                maxi = max(maxi, cost);
            }
            dp[i][j] = maxi;
        }
    }
    return dp[1][n];
}
```

TC:- $O(N^3)$

S.C. $O(N^2)$

@Aashish Kumar Nayak

DP-52

Boolean Expression to True | partition DP

Compression = "T|T&F"

or → |
& → and
XOR → ^

"T&F"

"F"

You have to perform in such a way you will get True

"T|T&F"

"T|F" ① → no. of ways

"T"

i j
"T^F|F&F^T|F"

(T)(F|T&F^T|F)

( )|( )

( )&( )

( )^( )

( )|( )

Similar to MCM

T ^ F | T & F ^ T | F

f(i, ind-1) partition (ind) f(ind+1, j)

left & right

both has to be true.

I'm looking for no. of ways left comes true & no. of ways in which right is true.

x + y.
left right

exp (left) | (Right)

T = x1 T = x2 = x1 x x2

F = x3 = x1 x x3

T ^ T = F

F ^ F = F

T ^ F = T

F ^ T = T

@Aashish Kumar Nayak

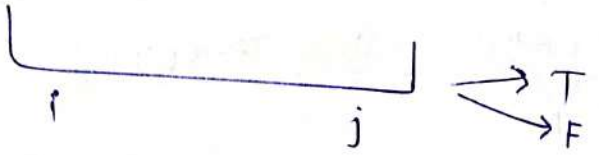
left ∧ right

T = x_1 T = x_2

F = x_3 F = x_4

$x_1 \vee x_4$

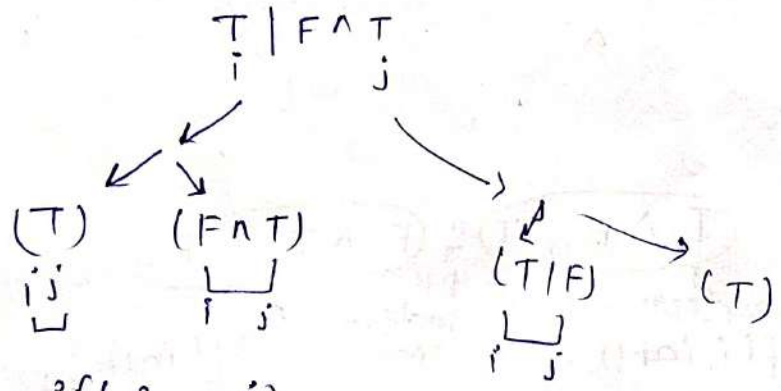
$x_3 \vee x_2$



In How many ways we can make it T or F.

~~f(i,j) is True~~

```
f(i,j, isTrue)
{ if(i>j)
    return 0;
```




```
if(i==j)
  { if(isTrue==1) return a[i]=='T';
    else return a[i]=='F';
```

looking for false.

→ 1 is false
0 is true.

Now and try to make partition.

```
int ways=0
for(ind i+1, ind<=j; ind=ind+2)
  { LT = f(i, ind-1, 1);
    LF = f(i, ind-1, 0);
    RT = f(ind+1, j, 1);
    RF = f(ind+1, j, 0);
```

ways = 0

if(a[ind] == 'T')

if(isTrue) ways += LTxRT; else ways += (LTxLF)+(LFxRT)+(RFxLF);

else if '^'

{

else if '|'

{

else

{

return ways;

So we are applying recursion

T.C. — O(exponential)

f(i, j, isTrue)

dp[N][N][2]

Code :-

```
int mod = 1000000007;
long long f(int i, int j, int isTrue, string &exp, vector<vector<vector<long long>>> &dp);
{ if(i>j) return 0;
  if(i==j)
  { if(isTrue)
    { return exp[i]=='T';
    }
    else return exp[i] == 'F';
  }
  long long ways = 0;
  if(dp[i][j][isTrue] != -1) return dp[i][j][isTrue];
for(int ind = i+1; i<=j-1; ind+=2)
{ long long lT = f(i, ind-1, 1, exp, dp);
  long long lF = f(i, ind-1, 0, exp, dp);
  long long rT = f(ind+1, j, 1, exp, dp);
  long long rF = f(ind+1, j, 0, exp, dp);
```

```
if(exp[ind] == '&')
{ if(isTrue) ways = (ways + (rT*lT)%.mod;
  else ways = (ways + (rT*lF)%.mod + (rF*lT)%.mod +
                                    (rF*lF)%.mod))%.mod;
else if(exp[ind] == '|')
{
if(isTrue)
ways = (ways + (lT*rT)%.mod + (lT*rF)%.mod +
                              (lF*rT)%.mod)%.mod;
}
else{
    ways = (ways + (rF*lF)%.mod)%.mod;
}
}
else
{ if(isTrue)
  { ways = (ways + (lT*rF)%.mod + (lF*rT)%.mod)
  }                                          %.mod;
else
  { ways = (ways + (lT*rT)%.mod + (lF*rF)%.mod)
  }                                          %.mod;
}}
return dp[i][j][isTrue] = ways;
int evaluateExp(string & exp)
{ int n = exp.size();
vector<vector<vector<long long>>> dp(n, vector<vector<long long>>
(n, vector<ll>(2, -1)));
return f(0, n-1, 1, exp, dp);
}
```

DP-53 Palindrome partitioning - II | Front partition

Given String = bababcbadcede

small string str = aabb n=4

3 partition a|a|b|b n-1 partition

↑ ↑ ↑ ↑

Every individual substring will be palindrome.

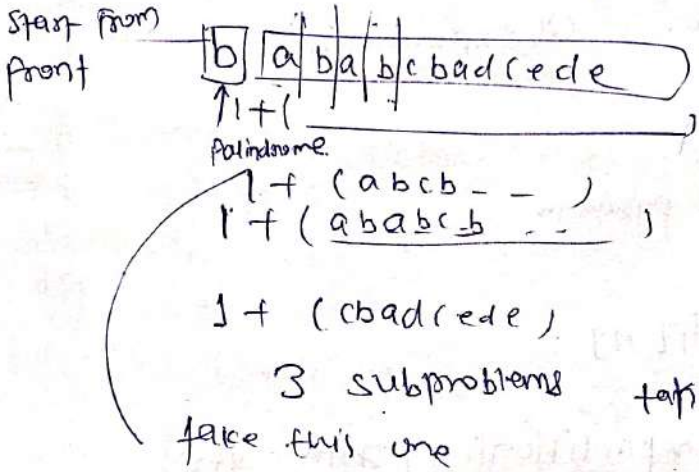
aa|bb

↑ ↗

pal

a|a|b|b

Such problems are generally solved using front partition



Writing a recurrance :-

1. Express everything in terms of index.
2. Express all the ~~ways~~ possibilities.
3. Take the min of all possibilities
4. Write the base case.

Give me the min. cut to make all substring palindrome.

```
f(i)
{ if( i == n) return 0;
                temp = " ";   if(dp[i] != -1) return dp[i];
for(j=i ; j<n; j++)
    {
      temp += s[j];
          if( isPalindrome (temp))
              cost = 1+F(j+1);
              minCost = min( minCost, cost);
          }
}       return minCost;
```

dp[i] =>

Recursion → T.C. — O(Exponential)

↓ Memoization

Changing parameter

0 → N

dp[N]

Code :

```
int palindromePartitioning (string str)
{ int n = str.size();
  vector<int> dp(n, -1);
  return f(0, n, str, dp) -1;
}
int f(int i, int n, string & str, vector<int> & dp)
{ if(i == n) return 0;
  if(dp[i] != -1) return dp[i];
  int minCost = INT_MAX;
  for(int j=i; j<n; j++)
    { if( ispalindrome(i, j, str))
      { int cost = 1 + f(j+1, n, str, dp);
        minCost = min( minCost, cost);
      }
  }return dp[i] = minCost;
}
```

temp = b
temp = ba
temp = bab

@Aashish Kumar Nayak

```
bool ispalindrome(int i, int j, string s)
{ while(i<j)
    { if(s[i] != s[j])
        return false;
      i++;
      j--;
    }
  return true;
}
```

T.C. $O(N \log N) = O(N^2)$

S.C. $O(N) + O(N)$ Ass.

Tabulation :

① Base case $(i == n)$ → 0

② $i = n-1 \longrightarrow 0$ $dp[n] = 0$;

③ Copy the recurrence

```
int palindromepartitioning(string str)
{ int n = str.size();
  vector<int> dp(n+1,0);
  dp[n] = 0;
  for(int i=n-1; i>=0; i--)
    { int mincost = INT_MAX;
      for(int j=i; j<n; j++)
        { if(ispalindrome(i, j, str))
          { int cost = 1 + dp[j+1];
            mincost = min(mincost, cost);
          }
        }
      dp[i] = mincost;
    }
  return dp[0]-1;
}
```

T.C. – $O(N^2)$

S.C – $O(N)$

@Aashish kumar Nayak

DP-54 Partition Array for Maximum Sum / Front Partition

arr[] = [1, 15, 7, 9, 2, 5, 10] K = 3
0 1 2 3 4 5 6

all elements of partition converted into max element

[15, 15, 15 | 9, 9, 9, | 10, 10]

Sum = 77

[1 15 7 | 9 | 2 5 10]

[15 15 15 9 10 10 10]

sm = 84 ↑↑

front partition logic ⟷ Maximum

We have various ways → Try recursion

Rules :

(i) Express everything in terms of index.
(ii) Try partition possible from that index.
(iii) Take the best partition.

[0, 1, 2, 3, 4, 5, 6]
[i, 15, 7, 9, 2, 5, 10]
 ↑
 ind

f(3)

f(0)
give me the max sum if we have the array from 0.

Base case : Try out all partitions from that index

Base case :
Whenever we are crossing the k we will stop

```
f(ind)
{
    Base case:
    maxAns = INT_MIN;
    len = 0; maxi = INT_MIN;
    for(j = ind; j < min(n, ind+k); j++)
    {
        len++;
        maxi = max(maxi, arr[j]);
        sum = (len x max) + f(j+1);
        maxAns = max(maxAns, sum);
    }
    return maxAns;
}
```

15, 15 7
j=1 j=2 j=3

T.C. O(exponential)
↓ Memorization
1 changing parameter
T.C. - O(N) x O(K)
S.C. - O(N) + O(N)

@Mallesh Kumar Nayak

```
int f(int ind, vector<int> &num, int k, vector<vector<int>> &dp)
{
    if(ind == num.size())
        return 0;
    int len = 0; if(dp[ind] != -1) return dp[ind];
    int maxi = INT_MAX;
    int maxAns = INT_MIN;
    for(int j = ind; j < min(ind+k, n); j++)
    {
        len++;
        maxi = max(maxi, num[j]);
        int sum = len * maxi + f(j+1, num, k, dp);
        maxAns = max(maxAns, sum);
    }
    return dp[ind] = maxAns;
}

int maximumSubarray(vector<int> &num, int k)
{ int n = num.size();
  vector<int> dp(n, -1);
  return f(0, num, k, dp);
}
```

Tabulation :-

(i) Base case

if(ind == n) return 0

i.e dp[n] = 0

(ii) Changing Variable

0 to n ; this time opposite ind : n-1 to 0

(iii) copy the recursion.

@Ashish Kumar Nayak

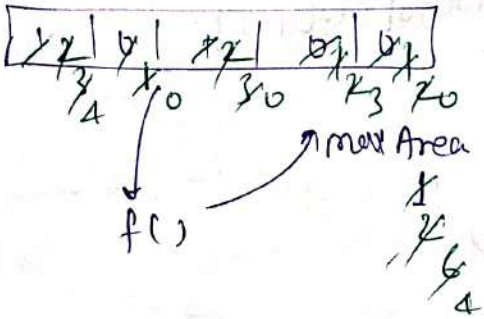
```
int maximumSubarray(vector<int> &num, int k)
{  int n = num.size();
   vector<int> dp(n+1,0);
   for(int ind = n-1; ind >= 0; ind--)
      {  int len = 0;
         int maxi = INT_MIN;
         int maxAns = INT_MIN;
       for(int j = ind; j < min(ind+k, n); j++)
            {  len++;
               maxi = max(maxi, num[j]);
               int sum = len * maxi + dp[j+1];
               maxAns = max(maxAns, sum);
            }
         dp[ind] = maxAns;
      }
      return dp[0];
}
```

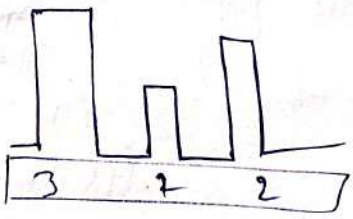
DP-55 Maximum Rectangle area with all 1s | DP on Rectangles

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

$2 \times 3 = 6$ (maximum)

We will solve this question using Histogram





if we find 0 then it becomes 0
else previous height will be added.

max = 6

Code :-

```
int maximalAreaofSubmatrixofAll(vector<vector<int>> &mat,
                                int n, int m)
{
    int maxArea = 0;
    vector<int> height(m, 0);
    for(int i=0; i<n; i++)
    {
        for(int j = 0; j<m; j++)
        {
            if(mat[i][j] == 1)
                height[j]++;
            else
                height[j] = 0;
        }
        int area = largestRectangleArea(height);
        maxArea = max(area, maxArea);
    }
    return maxArea;
}
```

@Ashish Kumar Nayak

```
int largestRectangleArea(vector<int>& histo)
{
    stack<int> st;
    int maxA = 0;
    int n = histo.size();
    for(int i = 0; i<=n ; i++)
    {
        while(!st.empty() && (i==n || histo[st.top()]>=
                                        histo[i]))
        {
            int height = histo[st.top()];
            st.top();
            st.pop();
            int width;
            if(st.empty())
                width = i;
            else
                width = i - st.top() - 1;
            maxA = max(maxA, width * height);
        }
        st.push(i);
    }
    return maxA;
}
```

T.C. ~~O(N×M)~~

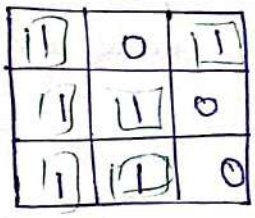
T.C. $O(N \times (m+n))$

S.C. $O(N)$

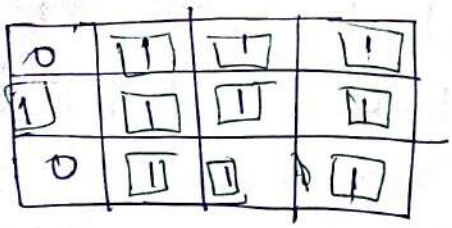
We are trying to remember the heights

that why this is kept in dp.

DP-56 Count No. of square Submatrices

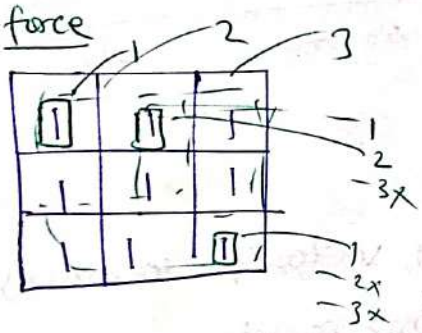


Size 1 — 6
Size 2 — 1



size 1 — 10
size 2 — 4
size 3 — 1

Brute force



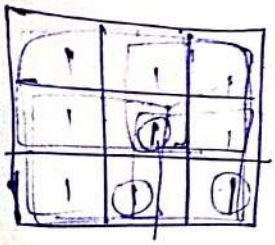
We can do better using dp.

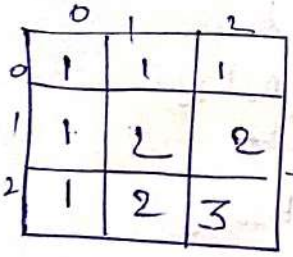
Try to create dp of similar size

dp[i][j]

How many squares end at Right Bottom at (i,j)

Try



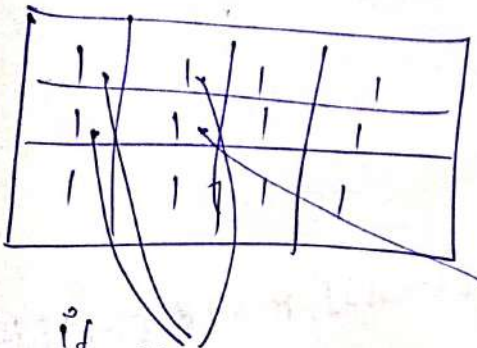


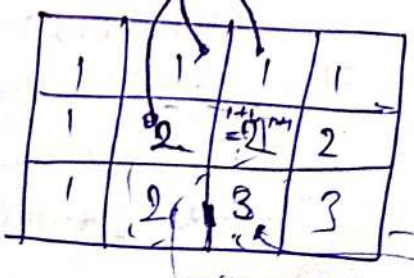
1 sq. he himself
+ he is right most bottom guy
= 2

sum all of them we will get our ans.

but How we will fill this array.

take minimal of three and add 1





first row and first column themselves are sq. of size 1.

if there 3 are 1 and if this is 1. then it will form square of size 2.

Now add of of them, and return the ans.

1	1	0
1	1	1
1	1	0

→

1	1	0
1	2	1
1	2	0

min = 0+1

— now add all of them.

min(1,1,1) = 1+1 = 2

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

upper. diagonal. left.

Code:-

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

add all of them in dp matrix and return.