DYNAMIC PROGRAMMING

Handwritten Notes of Striver(TUF) Playlist

by: Aashish Kumar Nayak

NIT Srinagar

AASHISH KUMAR NAYAK

aashishkumar.nayak

Scanned with OKEN Scanner

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 single 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

Scanned with OKEN Scanner

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 optimization
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 ~~Bit~~ 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

Scanned with OKEN Scanner

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, SC ↳ TC, SC → final soln

pre requisite
Recursion playlist
1st 10 vids, 6th & 7th imp

① fibonacci No.

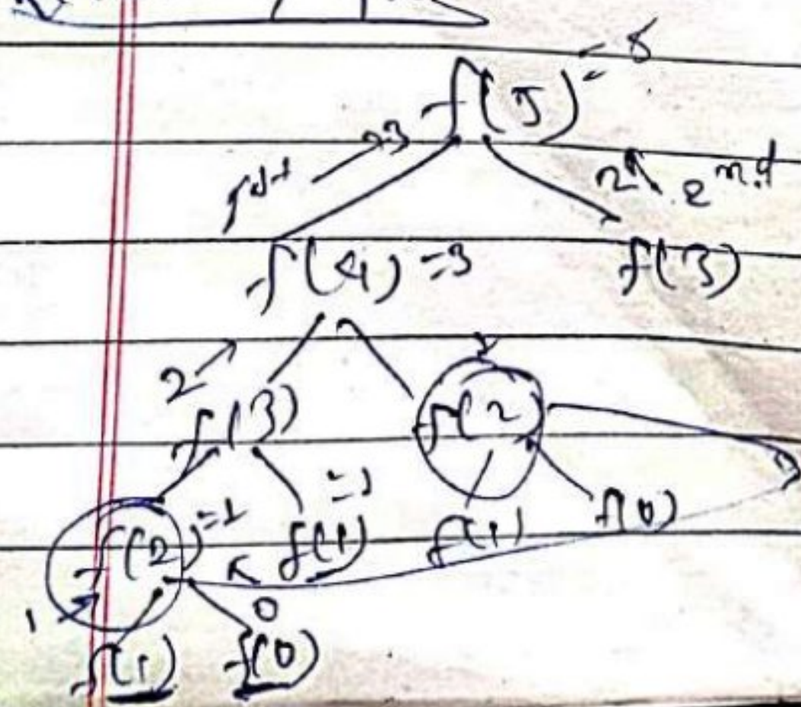
0 1 1 2 3 5 8 13 21
0 1 2 3 4 5 6 7

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

$f(4) =$

```
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.

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5
		1	2	3	5

initialy = -1

dp[n+1]

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

(1) convert Recursion into Memorization.
(2) dp[n]

Recursion → DP

3 steps

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

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

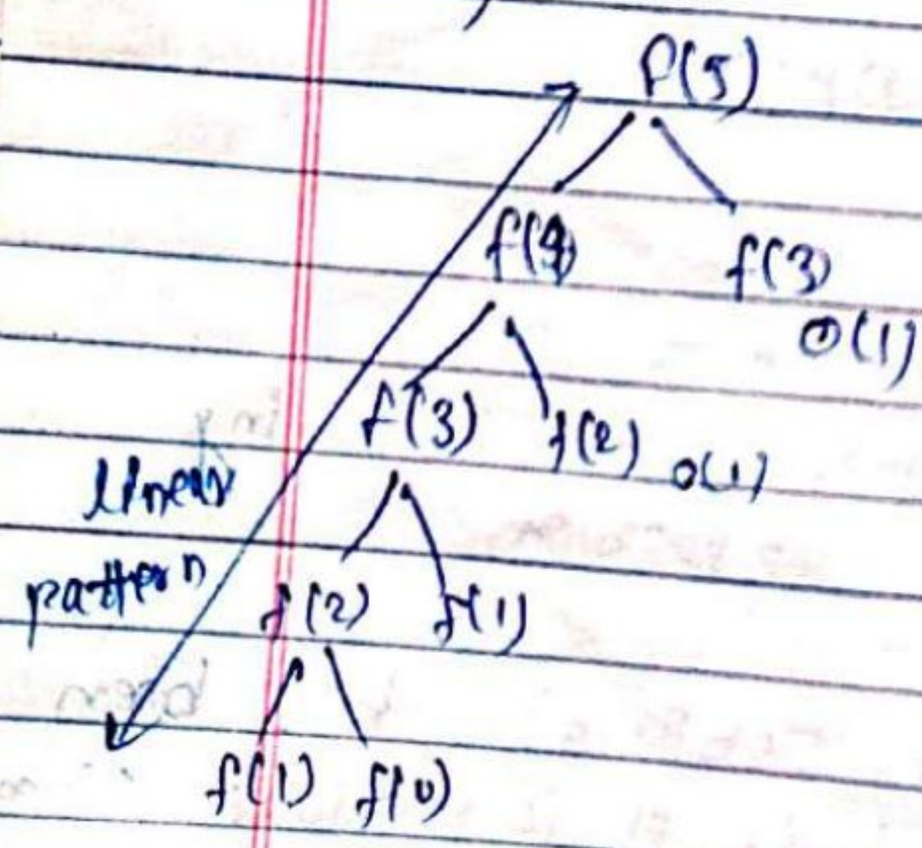
Step 2:- Checking if the subproblem has been previously solved. if it is previously solved

Step 3:- 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);
    cout<< f(n,dp);
    return 0;
}
```



TC = O(N)

SC = O(N) + O(N)

@Aashish kumar Nayak

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)

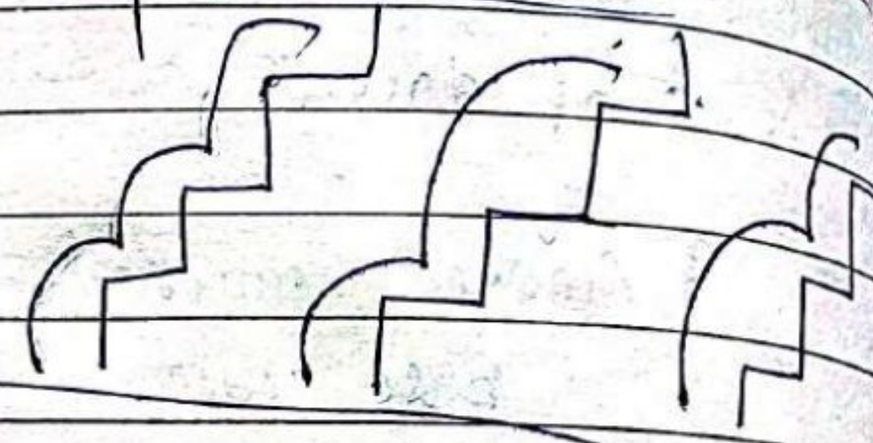
Intuitive solution

```
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;
```

@Ashish Kumar Nayak

PAGE NO.:
DATE: / /

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

all possible ways

Imp.

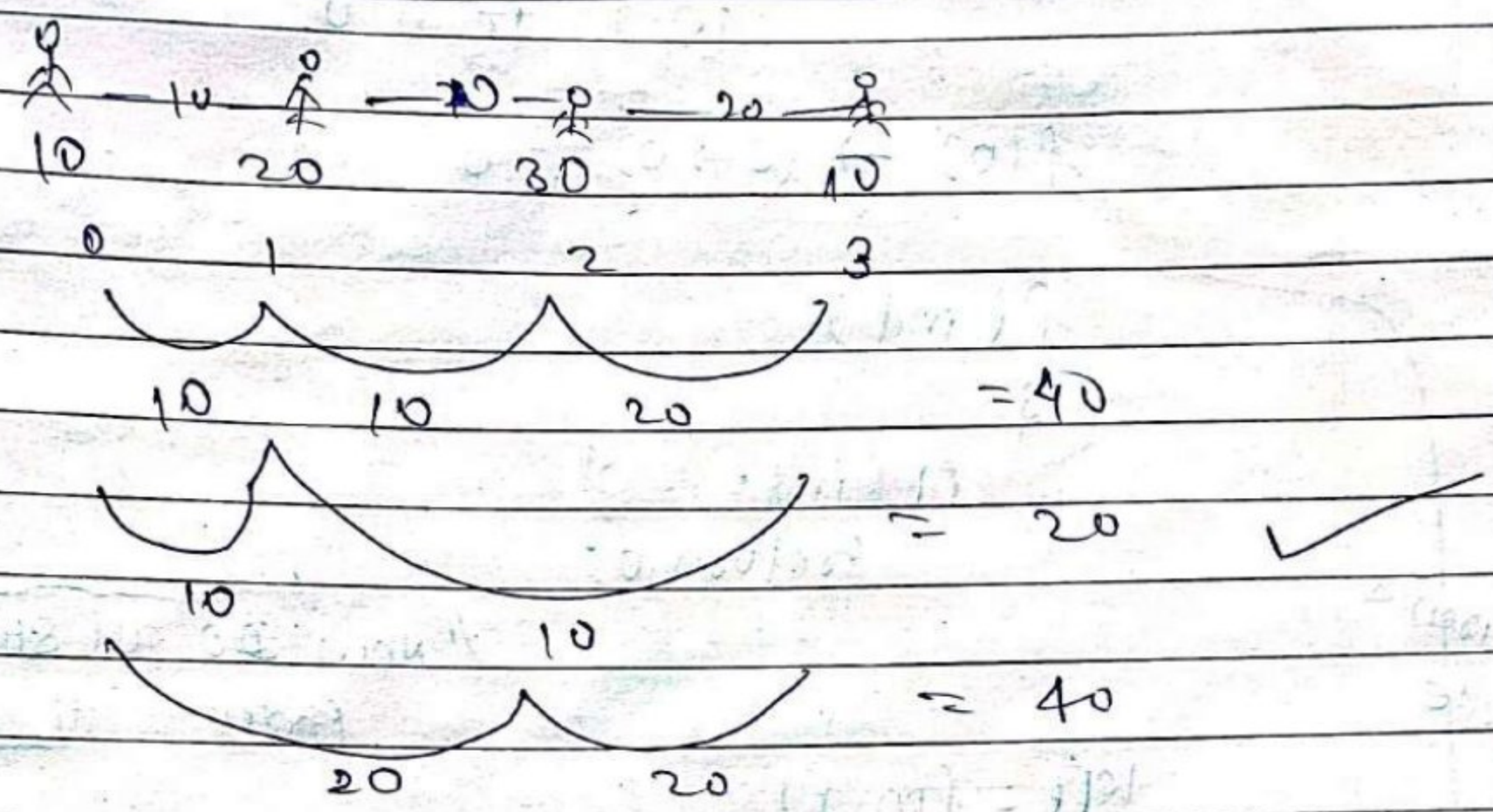
Shortcut

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

@Aashish Kumar Nayak

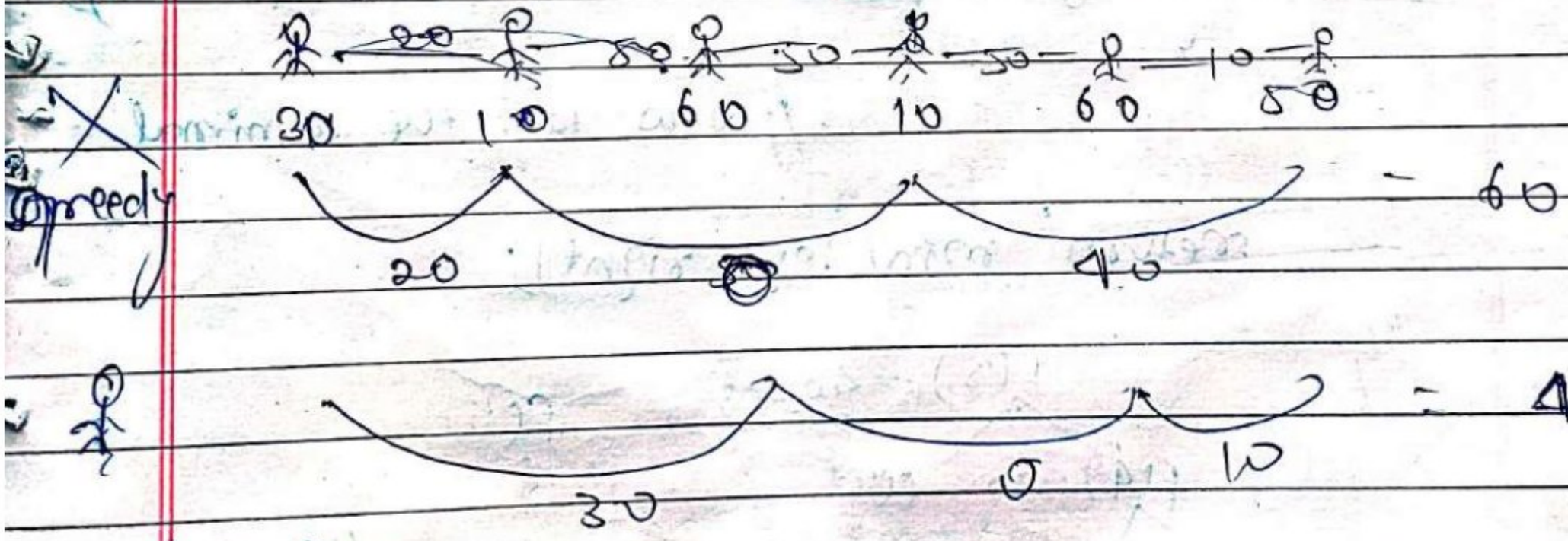
PAGE NO.:
DATE: / /

Frog Jump problem :



All possible ways, best ways, Means Recursion / DP

Why a greedy soln 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.

Step 3: take @ the min (all steps).

Aashish Kumar Nayak

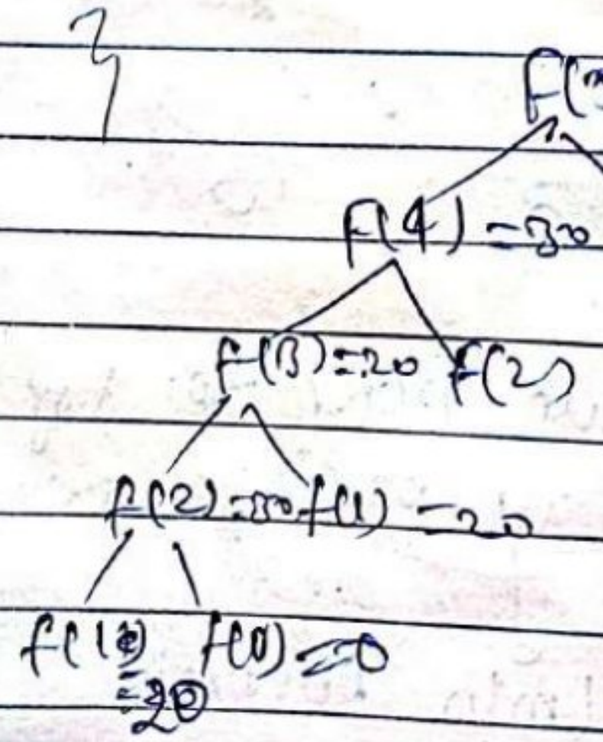
PAGE NO.:
DATE: / /

$f(n-1) \rightarrow$ 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(n-1)~~
    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

and so on

19/8

19/8

29/8

@ Aashish kumar Nayak

PAGE NO.:
DATE: / /

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

f(1)

{

left = f(0) + abs(20)

right = INT_MAX —— ∵ index 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) → 20)

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: / /

Now 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

$\boxed{dp[n+1] := }$ — 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

Lets code

```
int f(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 min(left, right);
    [return dp[ind] = min(left, right);]
}

int frogjump(int n, vector<int> &height)
{
    [vector<int> dp(n+1, -1);]   — (1) step
    return f(n-1, height, dp)
}
```

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

Now optimized Memoization

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

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

(@Aashish Kumar Nayak)

PAGE NO.:
DATE: / /

Memoization — Tabulation
(Top-down) (Bottom-up)

int dp[n] → for zero based index

Tabulation Method

```
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)~~
array dp — ~~Recursion~~ 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;
}
```

TC = O(N)

SC = ~~O(N)~~ + ~~O(N)~~
~~array dp~~ ~~Recursion~~

∴ SC = O(1)

both are reduced

Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Recurrence R 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~~ ~~{ if~~

```
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);
  }
}
```

PAGE NO.:
DATE: / /

dP[i] = min steps;

}

print (dP[n-1]);

TC → O(N×K)
SC → O(N)

Can't optimize further

as SC - O(K) + O(N)

MAXIMUM SUM OF NON-ADJACENT ELEMENTS

lets try out all subsequences with the given constraints

Try out all means Recursion.

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.

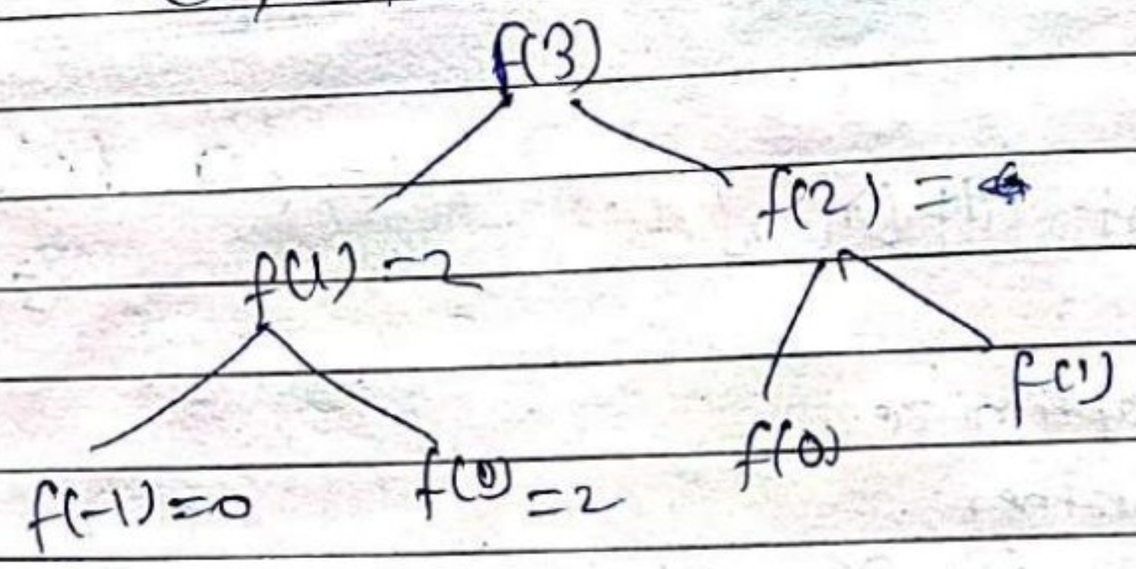
pick subsequence with no adjacent element.

@Ashish Kumar Nayak

Nayak

PAGE NO.:
DATE: / /

0 1 2 3
2, 1, 4, 9



f(1)
{

f(1) ⇒ 2 sm [0 — 1]

Pick = a[1] + f(-1) = 1 + 0 = 1

Not Pick = 0 + f(0) = 2

Max(Pick, Not Pick) ∴ f(1) = 2

f(3)
{

Pick = a[3] + f(1) = 9 + 2 = 11

Not Pick = 0 + f(2) = 6

Max(Pick, Not Pick) ∴ f(3) = 11

f(2)
{ Pick = a[2] + f(0) = 4 + 2 = 6

Not Pick = 0 + f(1) = 2

Max(Pick, Not Pick) ∴ f(2) = 6

TC - O(N)

SC - O(N) + O(N)

Ashish Kumar Nayak

PAGE NO.:
DATE: / /

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 reduce save from TLE

PAGE NO.:
DATE: / /

Tabulation

```
int dP[0] = a[0];

int nrg = 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[ind];
    if( i>1 )
        take = take + prev2;
    Non take = 0 + prev;
```

Aashish Kumar Nayak

PAGE NO.:
DATE: / /

```
curi = Max( take, Non take);
    prev2 = prev;
    prev = curi;
}
    print (prev);
```

```
int Max_sum_Non_adjac( 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

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 = ~~get~~ 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(maximumnoadj(temp1), maximumnoadj(temp2))
}
```

PAGE NO.:
DATE: / /

NINJA TRAINING Problem

whenever

Greedy fails

↓

Try all possible ways

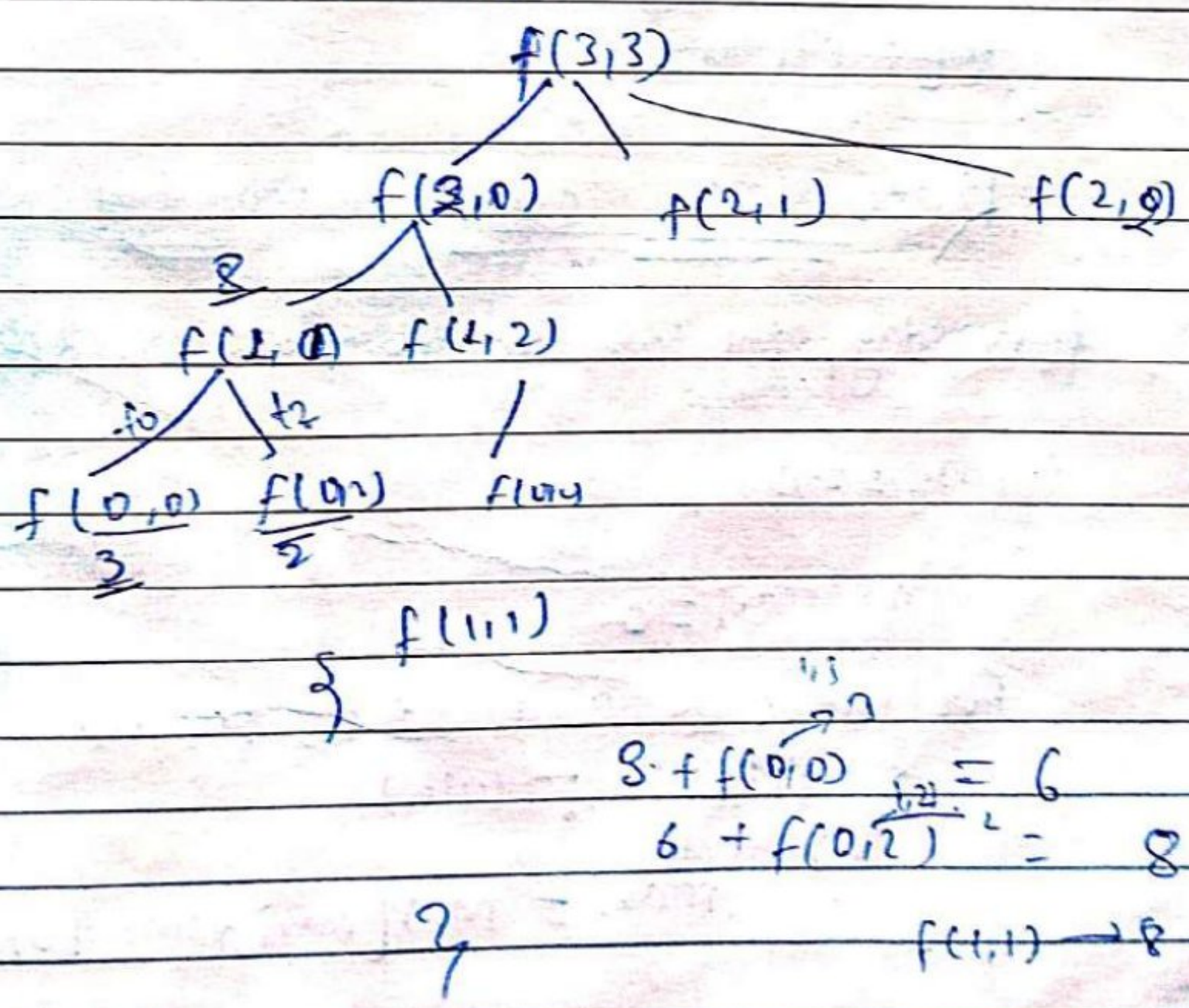
↓

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: / /

$f(1,2)$ — 6

$f(0,0)$ — 3 $\quad f(0,1)$

$f(1,2)$

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

?

$f(2,0)$ — 8

$\{ 1 + f(1,1) = 9 \}$
$\{ 6 + f(1,2) = 12 \}$ $\rightarrow$ 12

$f(2,1)$

Overlapping sub-problem
↓
Memoization

day — N

last — (0,1,2,3) — 4

changing parameter
ind, last

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 maxi;
}   return dP[day][last]= maxi;

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

TC - O(N×4)×3

SC - O(N) + O(N×4)

~~Tab~~ 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: / /

Fab

Tabulation

```
int minsatraing( 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 (point[0][0], points[0][1]);
    dp[0][3] = max (point[0][0], point[0][1], point[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)

PAGE NO.:
DATE: / /

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[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

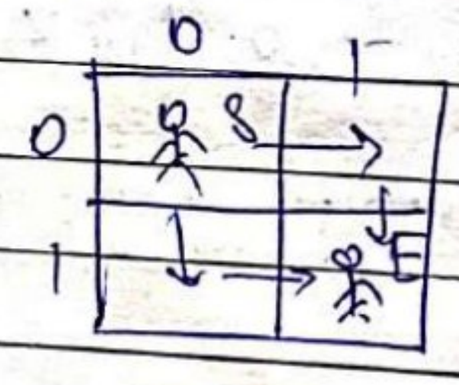
DP on Grids / 2D Matrix

- → 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

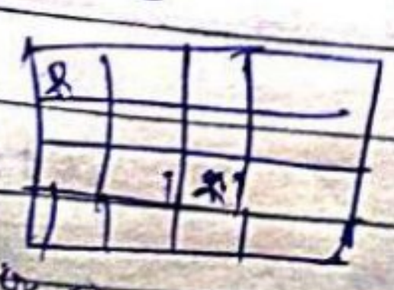
Means apply Recursion

How to write Recursion

1. Express in terms of index (i,j) row col
2. Explore/Do 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: / /

```
int 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] =

↑ upper
← left
2 path

TC → $(2^{m \times n})$

SC → (path Length)

$(m-1)+(n-1)$

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

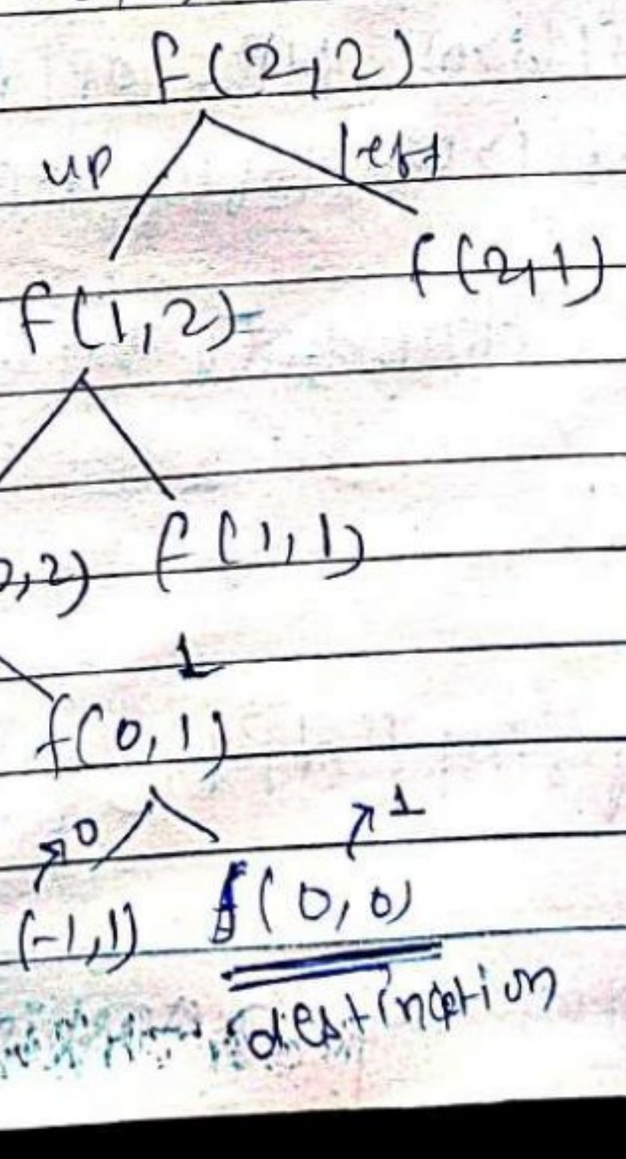
Recursion → DP

Memoization

overlapping sub problem.

	0	1	2
0	•		
1			
2			•

Let m = 3, n = 3




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

→ 1

TC → O(N×m)

SC = $\underbrace{(n-1)+(m-1)}_{} + \underbrace{O(N \times m)}_{DP}$

PAGE NO.:
DATE: / /

Tabulation :-

```
dP[n][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)

@Aashish kumar Nayak

PAGE NO.:
DATE: / /

→ If there is a previous row & previous column, we can space optimized 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

PAGE NO.:
DATE: / /

```
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

@Aashish Kumar Nayak

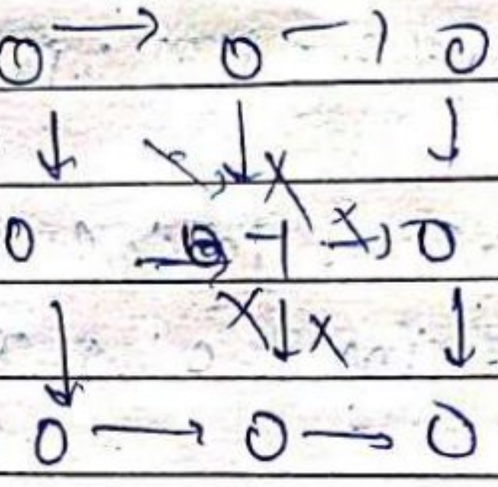
PAGE NO.:
DATE: / /

```
int uniquepath(int m, int n)
{
  vector<int> prev(n,0);
  for(int i=0; i<n; i++)
    {
      vector<int> cur(n,0);
      for(int j=0; j<m; 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 Mayak

PAGE NO.:
DATE: / /

Unique paths 2 DP on Grid with Maze obstacle



0 ways

```
if 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;
```

@Ashish Kumar Nayak

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

2x3

(0,0) → (0,2)

5 + 11 + 5 + 2 = 23 ✓

5 + 9 + 5 + 2 = 21 → min

Why not follow greedy

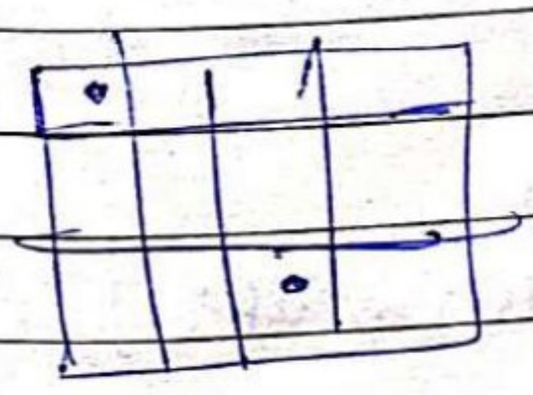
10	8	2
10	5	100
1	1	2

Greedy = 10 + 8 + 2 + 100 + 2

Min = 10 + 10 + 1 + 1 + 2 ✓

Try out all paths → Tell me the path which gives has min cost

(Recursion)



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

PAGE NO.:
DATE: / /

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

  if(i<0 || j<0)
      return INT_MAX;
  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);
      dp[i][j] =
    return min(up, left);
}
```

Recursion → Memorization
overlapping
sub problem

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);
```

@Ashish Kumar Nayak

PAGE NO.:
DATE: / /

```
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);
}
```

v(n,-1) v(n, vector<int>(n,-1))

Tabulation :-

vector<vector<int>> &grid(n,

```
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 Mayak

PAGE NO.:
DATE: / /

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

    else left += 1e9;

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

© Aashish Kumar Nayak

Scanned with OKEN Scanner

Space optimization

PAGE NO.:
DATE: / /

```
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];
}
```

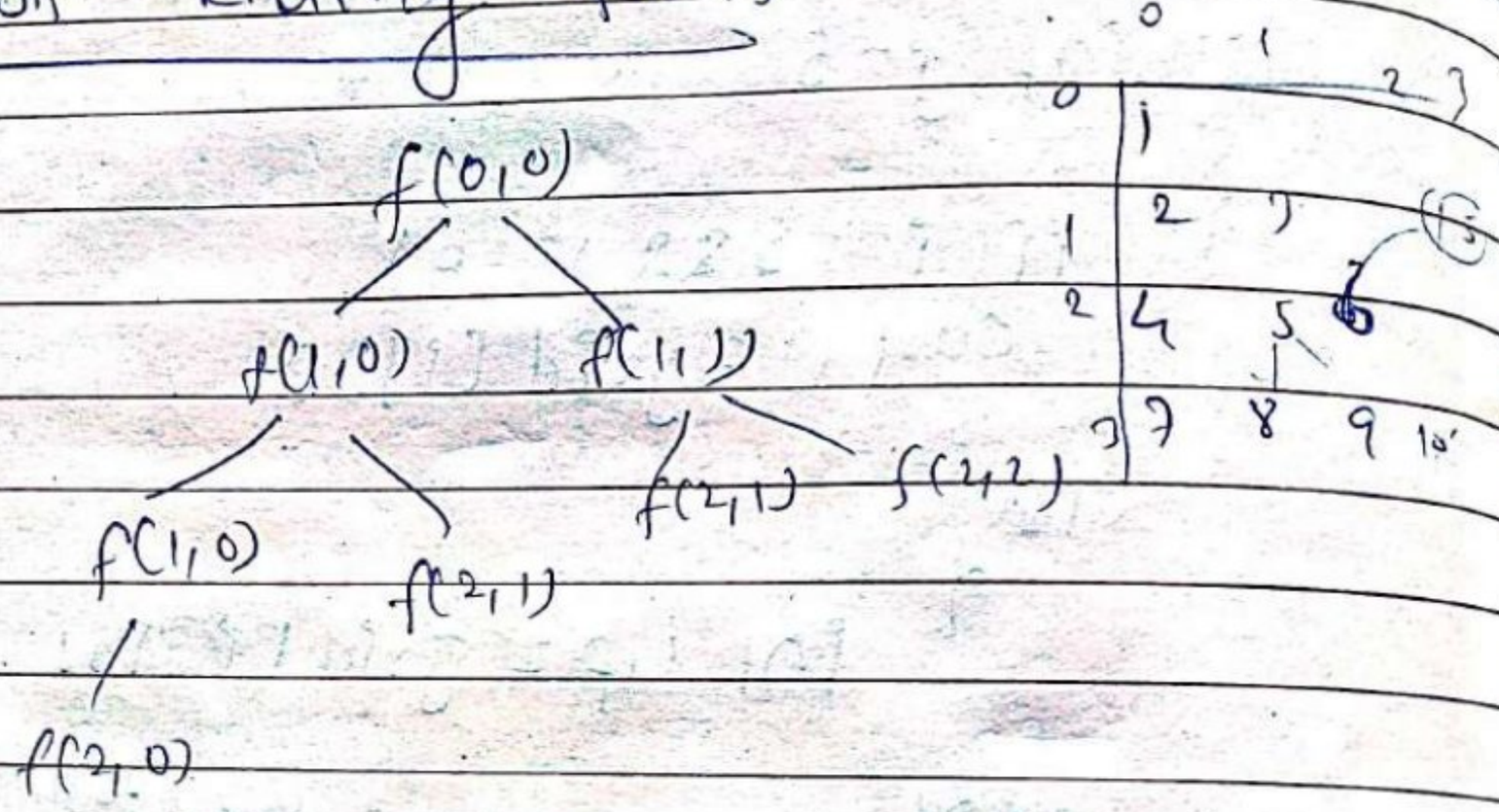
@Aashish Kumar Nagar

@ Aashish Kumar Nayak

PAGE NO.:
DATE: / /

D-12

Triangle fixed starting point and Variable Ending point




```
int f(int i, int j, vector<vector<int>> & triangle, int
    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 minsumTriangle (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 minsumtriangle( 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 = triang[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 Minsumpath(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

PAGE NO.:
DATE: / /

D-12

MINIMUM/MAXIMUM FALLING PATH SUM:

	2	10	4
100	3	2	1
1	1	20	2
1	2	2	1

Greedy → fails (becaus of uniformity)

Greedy will pass only when elements are uniform.

We can not apply greedy where uniformity is not there.

	0	1	2
0	1	1	100
1	2	2	3
2	3	100	2

3x3

f(2,0)

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

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

f(1,0)

{

ld = 2+f(0,-1) → -1eg

u = 2+f(0,0)

rd = 2+f(0,1)

Ashish Kumar Nayak

PAGE NO.:
DATE: / /

Recursion code

```
int f(int i, int j, vector<vector<int>> & matrix, vector<vector
                <int>> &dp)
{   if( j<=0 || j>= matrix[0] .size() )
        return -1e9;
    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 = -1e9;
    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][0];
    }
    for (int i = 1; i<n; i++)
    {
        for (int j = 0; j<n; j++)
        {
                                  prev[j]
            int u = matrix[i][j] + dp[i-1][j];
            int ld = matrix[i][j] + dp[i-1][j];
            int rd = matrix[i][j] + dp[i][
            int ld = matrix[i][j];
        if( j-1 >=0 )
            ld += dp[i-1][j-1];
        else
                                  prev[j-1];
            ld += -1e8;
            int rd = matrix[i][j];
            if( j+1 <m)
                rd += dp[i-1][j+1];
            else rd += -1e8;
                                  prev[j+1]
            dp[i][j] = max( u, max (ld, rd));
                                  curr[j]
        }
        prev = curr;
```

PAGE NO.:
DATE: / /

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

prev[j]

Space optimization code :-

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

Just replace dP[j][j] → prev[j]

& make → prev = curr;

and → return ~~prev~~ .prev

in this case

~~return maxi = max(u, max(u)~~

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

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

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

Recursion Recursion

take together

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

Alice Bob

Base case

i) out of Boundary

ii) last row case (Destination)

Boundary are always write first

```
f(i1, j1, i2, j2)
{ if( j1<0 || j1>=m || j2<0 || j2>=m)
        return -1e8;
```

PAGE NO.:
DATE: / /

↙↓↘ means they reach at same time.

```
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

↙↓↘

$(i+1, j-1)$ $(i+1, j)$ $(i+1, j+1)$

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)
      { a[i][j1] + f(i+1, j1+dj1, j2+dj2) }
    else
      { a[i][j1] + a[i][j2] + f(i+1, j1+dj1, j2+dj2) }
  }
}
return maxi;
```

fail

TC → $3^n \times 3^n$ = exponential

SC → $O(N)$ ASC

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<=0 || 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; dj2 <= +1; dj1++)
    {
      for( int dj2 =-1; dj <= +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++)
```

@ Aashish Kumar Nayak

PAGE NO.:
DATE: / /

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

    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];
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];
}
```

fail

out

ons

DP-14

SUBSET SUM EQUA TO K

DP in subsequences / subsets
↓
contiguous / Non contiguous

[1, 3, 2] → [1, 2], [3, 2]

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

Power set — BIT manipulation

Recursion

① Express (ind, target)

② Explore possibilities of the index.

f(n-1, target)

Tabulation

① Base case

② Form the Nested loops in

ind → [1, n-1]

target → (1 → target)

arr → [2, 3, 1, 1] target = 5

f(3, 4)

index target

f(2, 3) — (2, 4)

f(1, 3)

f(1, 2)

f(0, 2)

f(-1, 0) (-1, 2)

@Aashish Kumar Nayak

Recursion

PAGE NO.:
DATE: / /

```
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);
}
```

$TC = O(2^n)$

$SC = O(N)$

@Aashish Kumar Nayak

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)

fail

ut

Aashish Kumar Nayak

DP-15

Partion Equal subset sum

PAGE NO.:
DATE: / /

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

{2, 3, 5}
subset 1

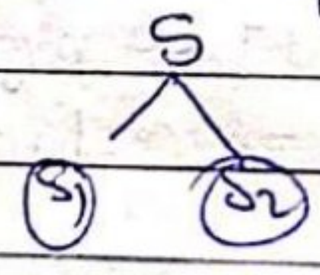
{3, 3, 4}
subset 2

subset with n sum

Exactly 2 subset

$S_1 = S_2$

Entire array sum = S



$S_1 = S_2 = S/2$

S = total sum

if S = odd $\not\rightarrow$ (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 subsetsumTOK( n, target, arr);
}
```

@Ashish Kumar Nayak

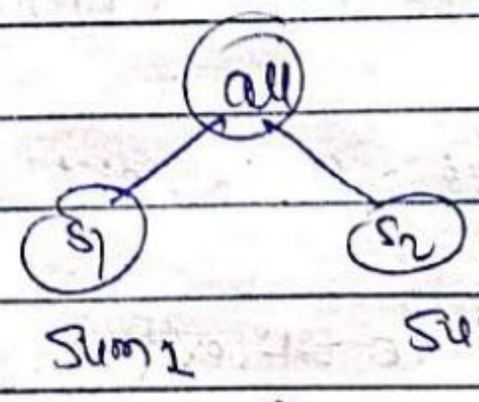
PAGE NO.:
DATE: / /

DP-16

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



abs|sum1 − sum2| is minimum

Subset sum → Tabulation

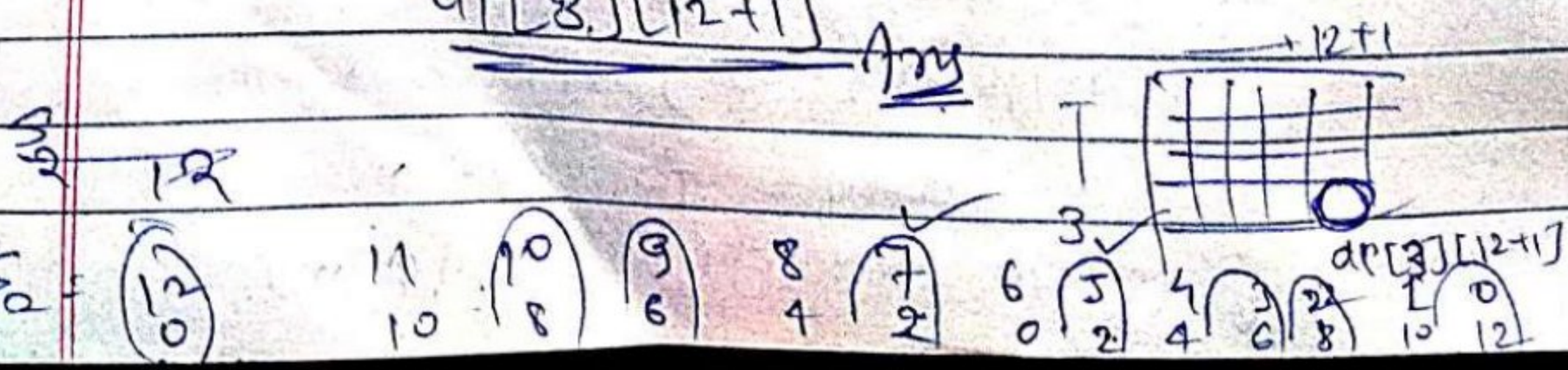
if we check for a target = K
We can derive if every Possible target between (1 & K) is ✓/✗

[3, 2, 7]

S_1 min 0 = 0
S_1 max = 12

all possible S_1
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
✓ ✗ ✓ ✓ ✗ ✓ ✗ ✓ ✗ ✓ ✓ ✗ ✓

dp[3][12+1] Ans



PAGE NO.:
DATE: / /

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

@Aashish Kumar Nayak

DP-17

PAGE NO.:
DATE: / /

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 Always

return 1 if true

return 0 if false

Recurrance

1.) Express in terms of (index, target)

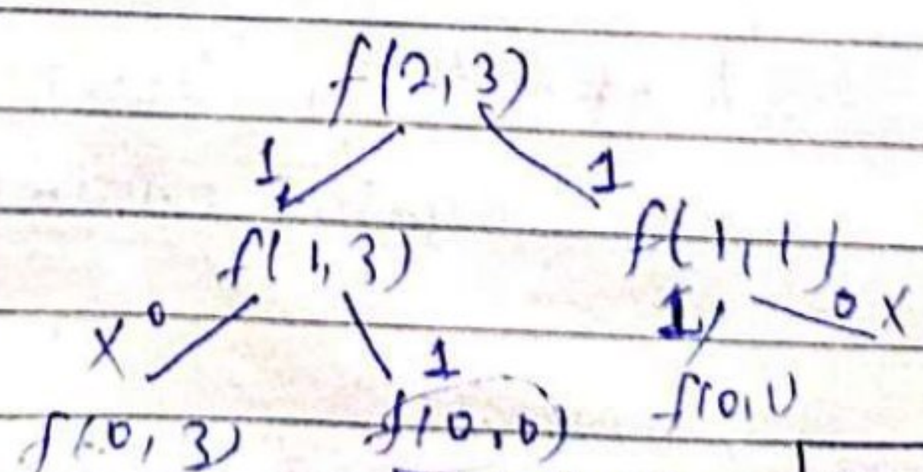
2.) Explore all the possibilities

3.) Sum up all the possibilities & return.

0	1	2	3
1	2	2	3

f(n-1, target) (n-1)

```
not_pick = f(ind-1, target);
pick = 0;

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

return pick + notpick;
}
```



@Aashish Kumar Nayak

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)

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Recursion code:

```
int f(int ind, int sum, 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[i] dp = return~~
            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), curr(tar+1);
    prev[0] = curr[0] = 1;
    if(num[0] <= tar)
        prev[num[0]] = 1;

    for(int ind = 1; index < 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]];

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

18

PAGE NO.:
DATE: / /

Count partitions with given sum difference :

~~Give~~ arr → {0, 0, 1} sum = 1

$\boxed{\text{ans} = 1}$ → 4

{0, 1}

{0, 1}

{0, 0, 1}

{1}

no. of zeros → 2

pow (2, n) × ~~ans~~ 1

power set $2^2 \times 1$

$4 \times 1 = 4$ subset

$S_1 >= S_2$

& $\boxed{S_1 - S_2 = D}$

$S_1 = \text{totsum} - S_2$

$D = \text{totsum} - S_2 - S_2$

$\boxed{D = \text{totsum} - 2S_2}$

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

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

DP-17 → modified targets

Totsum − D >= 0

totsum − D has to be even

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

Recursion + Memoization 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]) r
        return1;
        return0;
    }
    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);
}
```

© Aashish Kumar Nayak

Scanned with OKEN Scanner

PAGE NO.:
DATE: / /

```
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);
}
```

@Aashish Kumar Nayak

PAGE NO.:
DATE: / /

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
```

Scanned with OKEN Scanner

space optimization code

PAGE NO.:
DATE: / /

```
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
```

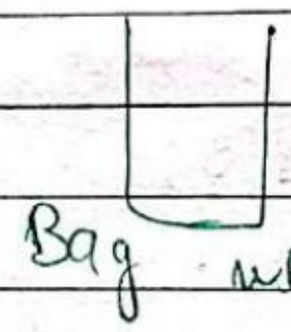
DP-19

PAGE NO.:
DATE 30/05/22

0-1 Knapsack

$n = 3$

wt. →	3	4	5
val →	30	50	60



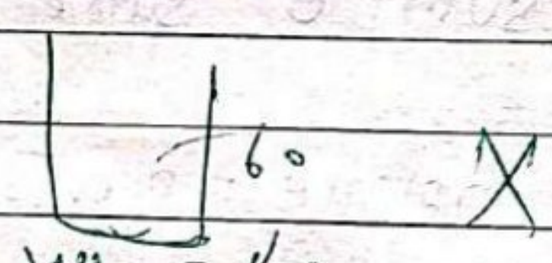
5 + 3 = 8 weight
val = 60 + 30
= 90

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, w)
ii.) Explore all possibilities
iii.) looking for maxim

3	4	5
30	50	60

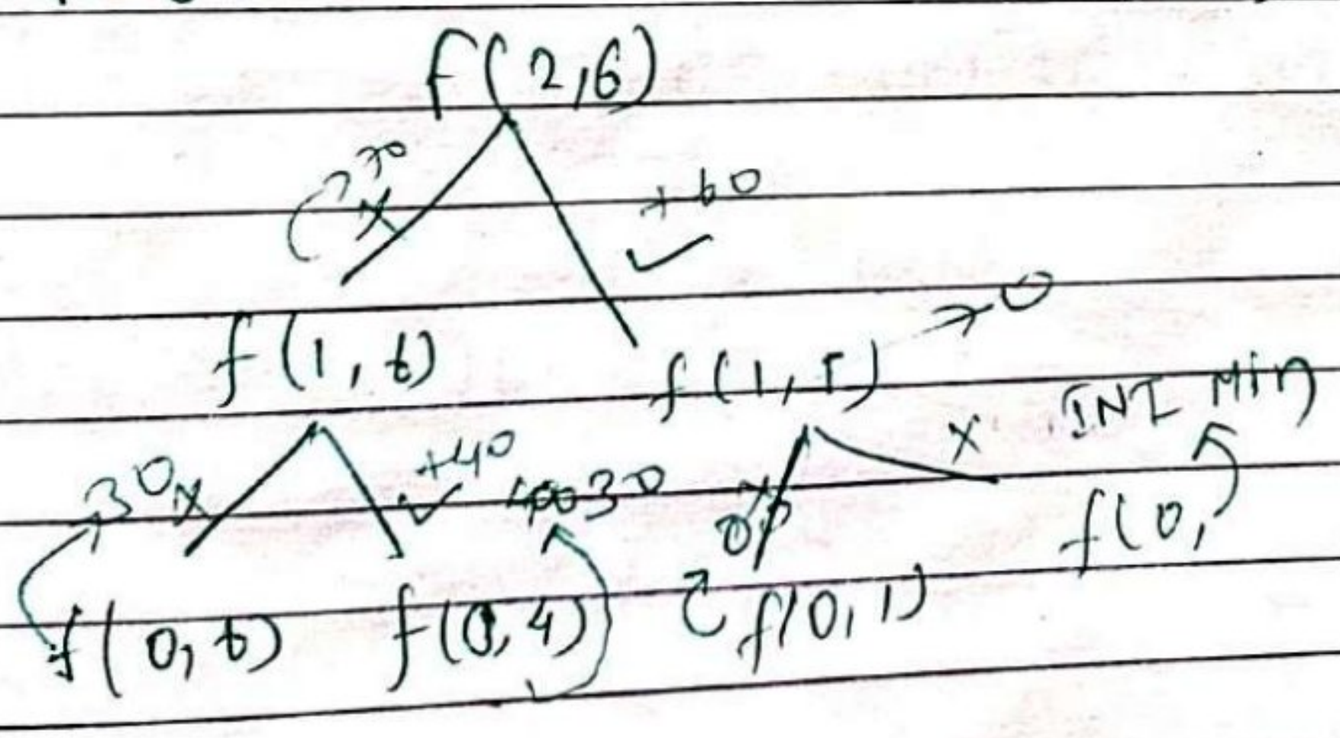
PAGE NO.:
DATE: / /

```
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 →	3	2	5
Val	30	40	60
Wt = 6			

wt → 6
val → 100
0

f(0,w)



Memoization.

ind, w

dP[ind+1][w+1];

TC = O(n×w)

SP = O(N×w) + O(N)

Tabulation

@Aashish Kumar Nayak

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];

© Aashish Kumar Nayak

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);
}
```

@Aashish 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

Scanned with OKEN Scanner

Space optimization

```
int knapsack( vector<int> wt, vector<int> val, int n, int
weight)
{
    vector<int> prev( weigh +1, 0), cur(weight+1, 0);
    ~~vector<vector<ir~~
    ~~vector<vector<int>> dp(n, vector<int> (maxweight +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 = cur;
  }
    return prev[weight];
}
```

@Aashish Kumar Nayak

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];
}
```

@Aashish Kumar Nayak

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 recurence in terms of index.
2. Express or do all steps.
3. Min. of all.

Knapsack

@Aashish Kumar Nayak

Whenever there is statement like

{ infinite supply, multiple case } it will stand at same index

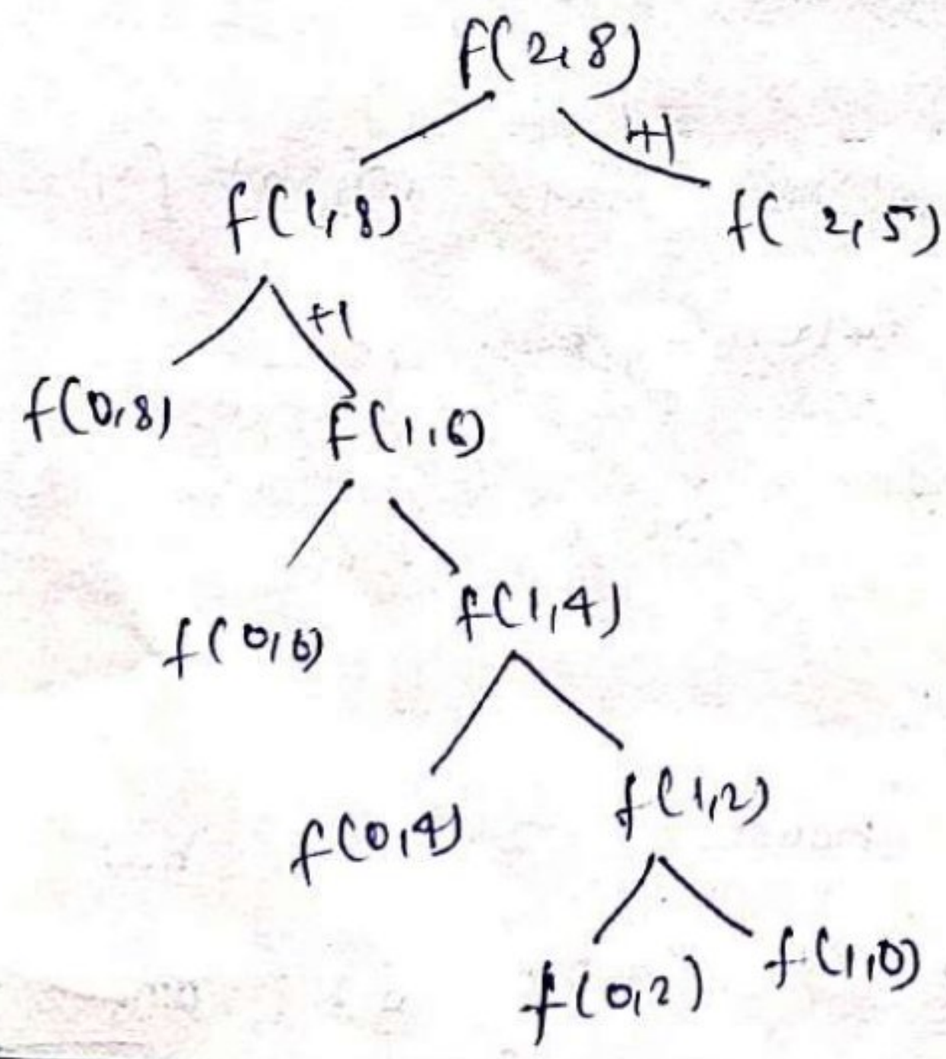
T.C. → $O(2^n)$ it will not go back.

S.C. → $O(N)$ - recursion

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

  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, NOT_take);
}
```

✗ $\boxed{6}$ a[0]	target = 7
✓ $\boxed{7}$	target = 7
$\boxed{4}$	target = 12

Base case



T.C. → $\gg O(2^n)$

S.C. → $\gg O(N)$

Exponential.

Now we will Memoize it,

changing parameters → ind, target

dp[ind][target] = Now

T.C. = $O(N \times T)$

S.C = $O(N \times T) + O(T)$ Auxillary Space

@Aashish Kumar Nayak

Tabulation

*** Any memoization can be converted into tabulation.

1. Base case
2. find target and write it in opposite fashions.
3. copy the turbulance.

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 min (take, notTake);
}  → 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, num, 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=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 not-take = 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, not take);
       }
  }
  int ans = dp[n-1][target];
    if( ans >= 1e9)
          return -1;
    return ans;
}
```

@Ashish Kumar Nayak

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] = leq;
    }
  for(int ind = 1; ind < n; ind++)
    { for(int T = 0; T<=target; T++)
      { int not_Take = 0 + prev[T];
        int take = INT_max;
        if(nums[ind] <= T)
        {
          take = 1 + cur[T - nums[ind]];
        }
        cur[T] = min(take, not_take);
      }
      prev = cur;
  int ans = prev[target];
   if (ans >= leq)
      return -1;
   return ans;
}
```

@ Aashish Kumar Nayak

Target sum problem | DP on subsequence

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

target = 3

$\overset{+}{1}\ \overset{+}{2}\ \overset{-}{3}\ \overset{-}{1} = -1$

$1 \quad 3 \quad 0 \quad -1$

count all the ways to get target.

$\overset{-}{1}\ \overset{+}{2}\ \overset{+}{3}\ \overset{-}{1} = 3$

$-1 \quad 1 \quad 4 \quad 3$

2 ways

$\overset{+}{1}\ \overset{-}{2}\ \overset{+}{3}\ \overset{+}{4} = 3$

$\boxed{S1 - S2 = D}$

sum partition

$\overset{-}{1}\ \overset{+}{2}\ \overset{+}{3}\ \overset{-}{1}$

[1 2 3 1]

$\underbrace{3+2}_{S_1}\ \underbrace{-1-1}_{S_2}$

$5 - 2 = 3$

$S_1 \quad S_2$

<u>same problem like in DP-18</u>

<u>Code</u>:

```
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;
}
```

@Aashish Kumar Nayak

```
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];
}
```

@Aashish Kumar Nayak

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

figure out total no. of ways → Try out all the ways
↓
Recursion

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

  f4 ___
  f2 ___

  return (_ + _);
}
```

Recurance :-

1. Express → (ind, T)
2. Explore all possibility → not Take / take
3. Sum all possibilities and return.

{ In all the DP subsequen. problem.

| 1 | 2 | 3 |
0 1 2

f(2, 4)
ind T

The 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;
```

✓ $\boxed{4}$ arr[0] — tar = 4

× $\boxed{3}$ — tar = 4

✓ $\boxed{2}$ — tar = 4

tar % arr[0] that 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. = $O(N \times T)$

S.C = $O(N \times T) + O(\text{target})$ — 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 recurrence.

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);
}
```

@ Ashish Kumar Nayak

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

@Aashish kumar Nayak

DP-23 unbounded Knapsack

Prerequisite
DP-19 — 0/1 knapsack
in this there was only one occurances.

wt → {2, 4, 6}
val → {5, 11, 13}

W = 10

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

Ex:-
13 wt → 6 11 wt → 4

wt → 6
10

wt → 2 × 2
+ 5 × 2 = 23

wt → 4 × 2 + wt 2
11 × 2 + 5 = $\boxed{27}$ maximum possible value.

(wt → 2) × 5 = 10 wt
5 × 5 = 25 val

Base case we write for 0

val 10
wt 5
arr[0]

W = 8
if W = 2 ×
int $\left(\frac{W}{wt[0]}\right) \times val[0]$

```
f(ind, W)
{   if(ind == 0)
        return [ W/wt[0] × 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 recurance.

>> O(W)

@Ashish Kumar Nayak

so 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)
{
    int 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 unbounded knapsack (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);
}
```

@Aashish Kumar Nayak

Tabulation :-

~~vec~~

```
int unboundedknapsack(int n, int W, vector<int> &val,
                      vector<int> &wt)
{ vector<vector<int>> dp(n), vector<int>(wt+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 notpick = 0 + dp[ind-1][w];
      int pick = 0;
      if(wt[ind] <= w)
      { take = val[ind] + dp[ind][w - wt[ind]];
      }
      dp[ind][w] = max(take, notTake);
    }
  }
  return dp[n-1][W];
}
```

@Aashish kumar verma

Space optimization

int-1, int — so space optimization 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]= max(take,notTake);
                Cur[w] = max(take, notTake);
            }
R           // prev = cur; //Remove this line
        }
    return prev[w];
}
```

Remove.

//prev[w - wt[ind]]

prev | 3 | 4 | 5 | 6 | 7 | 1 | 2 |

notTake = 3, 4 cur

take =

than we rewrite.

1D array optimization

@Aashish Kumar Nayak

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

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 sum 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 for 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 x 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);
}
```

$n-1 \longrightarrow 0$

✓ $\boxed{6}$ —— $N = 12$

Price[0] → rod_length = 1

looking to make it.

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

~~Pricing~~ price = N x 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 x N)

S.C. → O(N x N) + O(Target)

Ans

Mem Tabulation :-

1. Base case
2. Changing parameter — ind, N
3. copy recurrence

Base case.

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

~~for~~ ind → 1 ⟶ n-1

N ⟶ 0 ⟶ N

(copy the recurrence)

@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++)
       {
```

@Aeshish Kumar Nayak

Scanned with OKEN Scanner

```
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];
        // prev[N-rodLength]
      }
      cur[N] = max(take, notTake);
    }
    //prev[N]
  }
  ~~return dp~~ 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 string
→ 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~~ methods to print ~~all~~ the subset.

S_1 = "adebc" —— 2^5 subsequences

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

S_2 = "dcadb" —— 2^5 subsequences

d, c, a, d, b, dc, da, dd, db, - - — adb ← 3

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 parameter to generate them.

@ Ashish Kumar Mayak

Scanned with OKEN Scanner

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

@Aashish Kumar Nayak

Rules to write the recurrence

(i) Express everything in terms of index.
(ii) Explore possibilities on that index.
(iii) take the best among them.

Lets take an example

d == d

ac | ce

ind2 ind1

if match

acd | ced

ind1 ind2

$1 + f(ind1-1, ind2-1)$;

ac | ce

if not match

ac | ce $\xrightarrow{+1}$ $f(ind1-1, ind2-1)$ it goes -ve

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

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

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

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

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

match

not match

0 1 2 | 0 1 2
acd | ced $f(2,2) = 2$

$+1 + 1 = 2$

ac | ce $f(1,1)$

a | ce $f(0,1)$ 0

ac | e 1 $f(1,0)$

1 $f(0,-1)$ 0

| ce $f(-1,1)$

a | e $f(0,1)$

0 0

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

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

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

Code :-

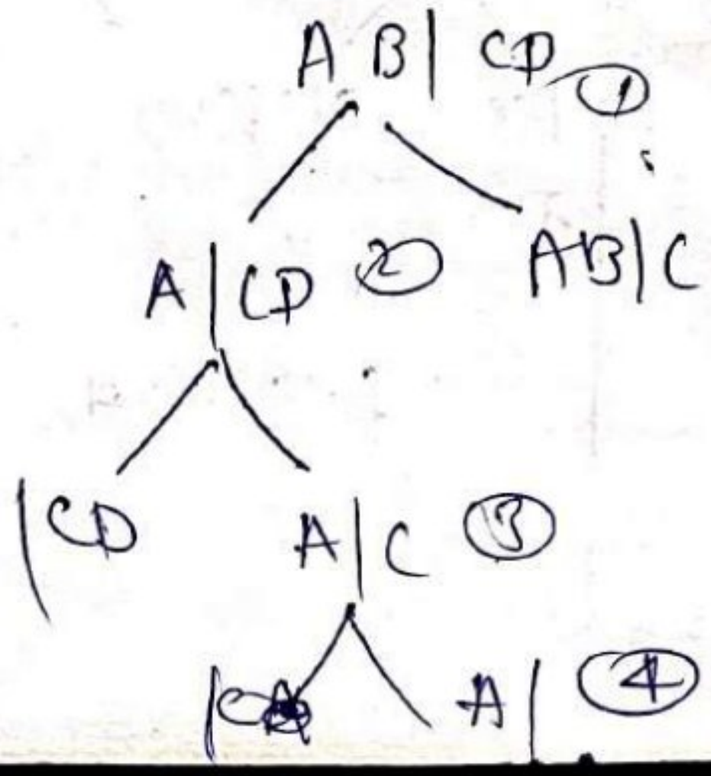
```
int f(int i, int j, string &s, string &t, vector<vector<int>> &dp)
                i==0      j==0
{
    if(i<0 || j<0)
        return 0;
    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j])
        return 1 + f(i-1, j-1, s, t);
    return dp[i][j] = max(f(i-1, j, s, t, dp), f(i, j-1, s, t,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

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.

@Aashish Kumar Nayak

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];
}
```

@Ashish Kumar Nayak

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];
}
```

©Aashish Kumar Nayak

D6-26 print longest common subsequences :-

S_1 = "abcde" S_2 = "bdget"

len(lsc) = 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

move (3,2)

come here.

match

dp[5][5] = 3

abcde Bdgek

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

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 if( dp[i][j] = max(dp[i-1][j]
  }
```

@Aashish Kumar Nayak

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

while(i>0 && j>0) for(i = 0 → len) s += "q";

$	$	$

{ 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 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]);
      }
   }
}
```

@Aashish Kumar Nayak

Scanned with OKEN Scanner

```
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. $O(n+m)$

@Aashish Kumar Nayak

DP-27 longest common substring :-

S_1 = "ab[jk]lp" S_2 = "a[jl]p"

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

Breaking the consecutive rule

(+1)

ac | ax

a | ax ac | a

(+1)

| ax a | a c | a ac |

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

no need to do a lookup 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.

Scanned with OKEN Scanner

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;
}
```

@ Ashish Kumar Nayak

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;
}
```

@Aashish Kumar Nayak

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 babcbc ab "

S_2 = " ba c bcba b b " — 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);
}
```

@Aashish Kumar Nayak

DP-29 Minimum insertions to make a string palindrome:

prerequisite → LCS
→ LPS

S = "abcaa"

↓

abcaa aa cba
reverse

max operation = len(S)

$\boxed{S + rev(S)}$ — 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

a b c a b a

we will match if not match then we will place copy of that in right part.

$\boxed{\text{code}}$ in ni $\boxed{\text{max}}$

$\boxed{n \text{ — longest palindromic subsequence}}$

size of string.

@Aashish Kumar Nayak

Code :-

```
int minInsertion(string &str)
{ return str.size() - longestpalindromesubsequence(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];
}
```

@Aashish Kumar Nayak

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
(b, d: x)

lets delete b & d

a c — 2 deletion

now insert n

$\boxed{anc}$ — 1 insertion

3 operations

longest common subsequences.

deletions = n − len(lcs)
insertions = m − len(lcs)

$\boxed{n+m - 2\times len(lcs)}$ → 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(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];
}
```

@ Aashish Kumar Nayak

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.

~~"bruu~~ "bgruoote" — len = 8

S_1 = "bleed" S_2 = "blue"

"bleued"

fig. out lengths

print ≠

Common guys should be taken Once

common guy in two string means $\underline{LCS}(_,_)$

↓

it tells me longest common guys

"bgruoote"

S_1 S_2

blue (rt) goo

$(n+m) - len(LCS)$

print string

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

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

we will create dP table.

@Aashish Kumar Nayak

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(1,1) = 1

common guys get added by 1

etouorbg

reverse ~~the~~ it

gbrouote → len(8)

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

@ Ashish Kumar Nayak

$\boxed{i>0}$ $\boxed{j>0}$ @Aashish Kumar Nayak

String 1 has some length

S_2 has some length

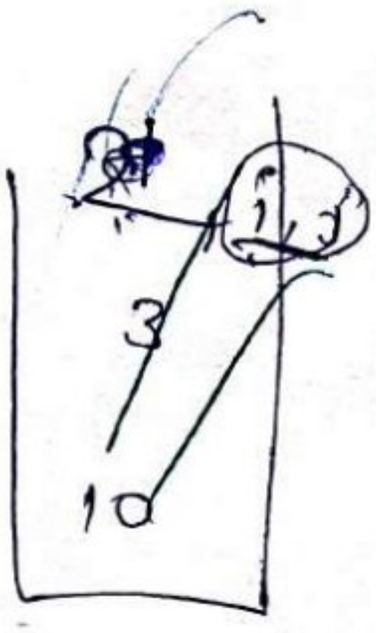
```
while (i>0) ans += s1[i-1]; i--
while (j>0) ans += s2[j-1], j--
```

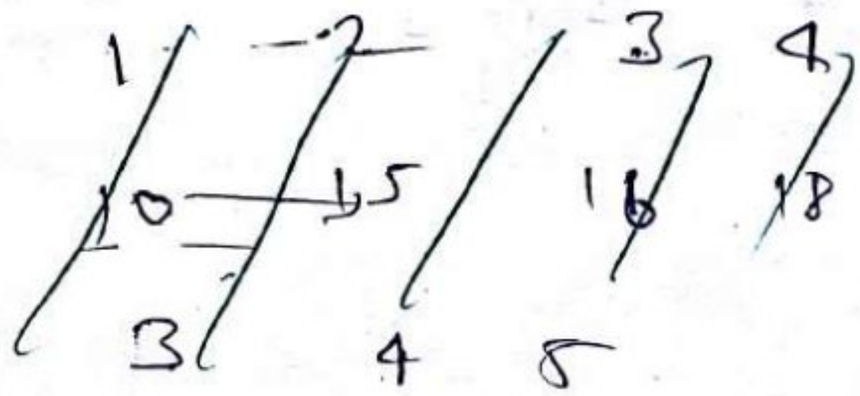
Code :-

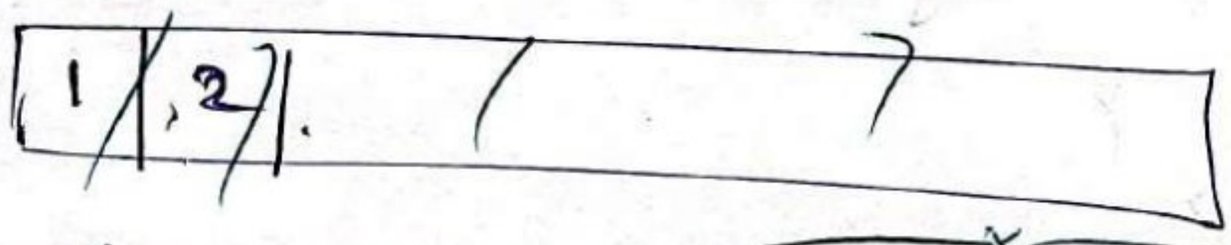
```
string shortestSuperSequence( string s, string t)
{   int n = s.size();
    int m = t.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
    for( int j = 0; j<=m ; j++)
        dp[0][j] = 0;
    for( int i=0; i<=n; i++
        dp[i][0] = 0;
    for(int i = 1; i<=n; i++)
    { for(int j=1;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

@Aashish Kumar Nayak







~~DP on~~

Distinct subsequences

$s = $ "babgbag" $\quad s_2 = $ "bag"

occurance of s_2 in s_1 in babgbag → 5 Subs Distinct ways

Differents methods of comparing

Trying all the ways → Recursion

Count the no. of ways

```
f()
{
    Base case ↑' ↓0
    f1()
    f2()
    return f1()+f2();
```

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

How to write recurrance

(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

Aashish Kumar Nayak

$f(i,j)$ $\quad$ S_1 = ~~bagg~~ babgbag (i) n-1

S_2 = bag (m-1)

$f(n-1, m-1)$ — 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

SC → O(N+m)

S_1 → 2^n, S_2 → 2^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) + f(i-1, j);~~
        return f( i-1, j-1, s, t, dp) + f(i-1, j, s, t, dp);

return ~~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> dp(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.

Base case $\boxed{j==0}$ → 1

no string

s_2

$j==0$ → 0

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

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<int>> dp(n+1, vector<int>(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 dp[n][m];
}
```

(int) connected back

@Ashish Kumar Nayak

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];
  }; }
```

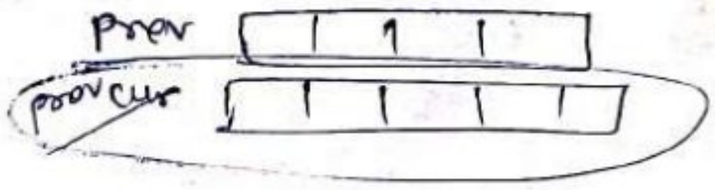
single array optimization

i = 1 → n

j = 1 → m

if() → cur[j] = prev[j-1] + prev[j]

else → cur[j] = prev[j]




```
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];
}
```

@Aashish Kumar Nayak

DP-33.

Edit Distance

S_1 = "horse" S_2 = "ros"

1. insert
2. Remove
3. Replace

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

---

① 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

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 — match — ok
ros
not match min delete
not match end
two case
replace or insert the same char

→ if not matching
→ insert of the same character
→ delete & try finding somewhere else
→ replace & match.

@Aashish Kumar Nayak

Try all ways → Recursion @Aashish Kumar Nayak

How to write Recurrance?

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

f(i, j)

horse (i)
ros (j)

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

f(i, j)
{
if(s1[i] == s2[j]) s1[0----i] to s2[0---j]
return 0 + f(i-1, j-1); // if they are matching shrink both string and we dont require any steps.
}

1 + f(i, j-1); // insert // if not matching i can insert
1 + f(i-1, j); // delete
1 + f(i-1, j-1); // replace

S_1 gets exhausted return (j+1) steps

horse
ros
f(-1, 1) → min operations to convert empty string to a s2[0---1] "ros"

S_2 gets exhausted return (i+1) operations

S_1 = "horse" (0 1 2)
S_2 = "ros" (-1 0 1 2)
f(2, -1) → min operations to convert "hor" → " "
3 operations

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) $\boxed{i==0}$ return j — Here j can be anything.

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

$\boxed{j==0}$ return i

for(i=0 → n) dp[i][0] = i;

@Ashish Kumar Nayak

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];
}
```

space optimization

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

@Aashish Kumar Nayak

Base Cases

$dp[0][j] = j$ → can be written in single array

0	1	2	3	4 - - j

$dp[i][0] = i$ → can be written in single array.

0				
1				
2				
3				
4				

```
int editDistance(string str1, string str2)
{
    int n = str1.size();
    int m = str2.size();
~~for (vector~~
    vector<int> prev(m+1, 0), curr(m+1, 0);
    for(int j=0; j<=m; j++)
        prev[j] = j;
    for(int i=1; i<=n; i++)
    {   curr[0] = i;
        for(int j=1; j<=m; j++)
        {   if(str1[i-1] == str2[j-1])
            ~~curr~~ curr[j] = prev[j-1];
            else
                curr[j] = 1+min( prev[j],
                                 min(curr[j-1],
                                     prev[j-1]));
        }
        prev = curr;
    }
    return prev[m];
}
```

@Aashish Kumar Nayak

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

S_1 = "?ay"
S_2 = "day"
all three chars are matching
return true

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

S_1 = "ab*cd"
S_2 = "abdefcd"
return true

S_1 = "cb?d"
S_2 = "abcd"
return false

S_1 = "**abcd"
S_2 = "abcd"
return true

1st thing which comes in brain string matching → we can do using recurrence

Problem faced →
ab*cd
abdefcd
for this star we dont know how much of length of other string it will cover

so we have to try all possible ways

len = 0, 1, 2 → means recursion

so we will try string matching by recursion.

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

Scanned with OKEN Scanner

s1[0 - - - 4]
s2[0 - - - 6]

$f(m-1, m-1) \rightarrow$

$f(4, 6) \rightarrow (T/F)$

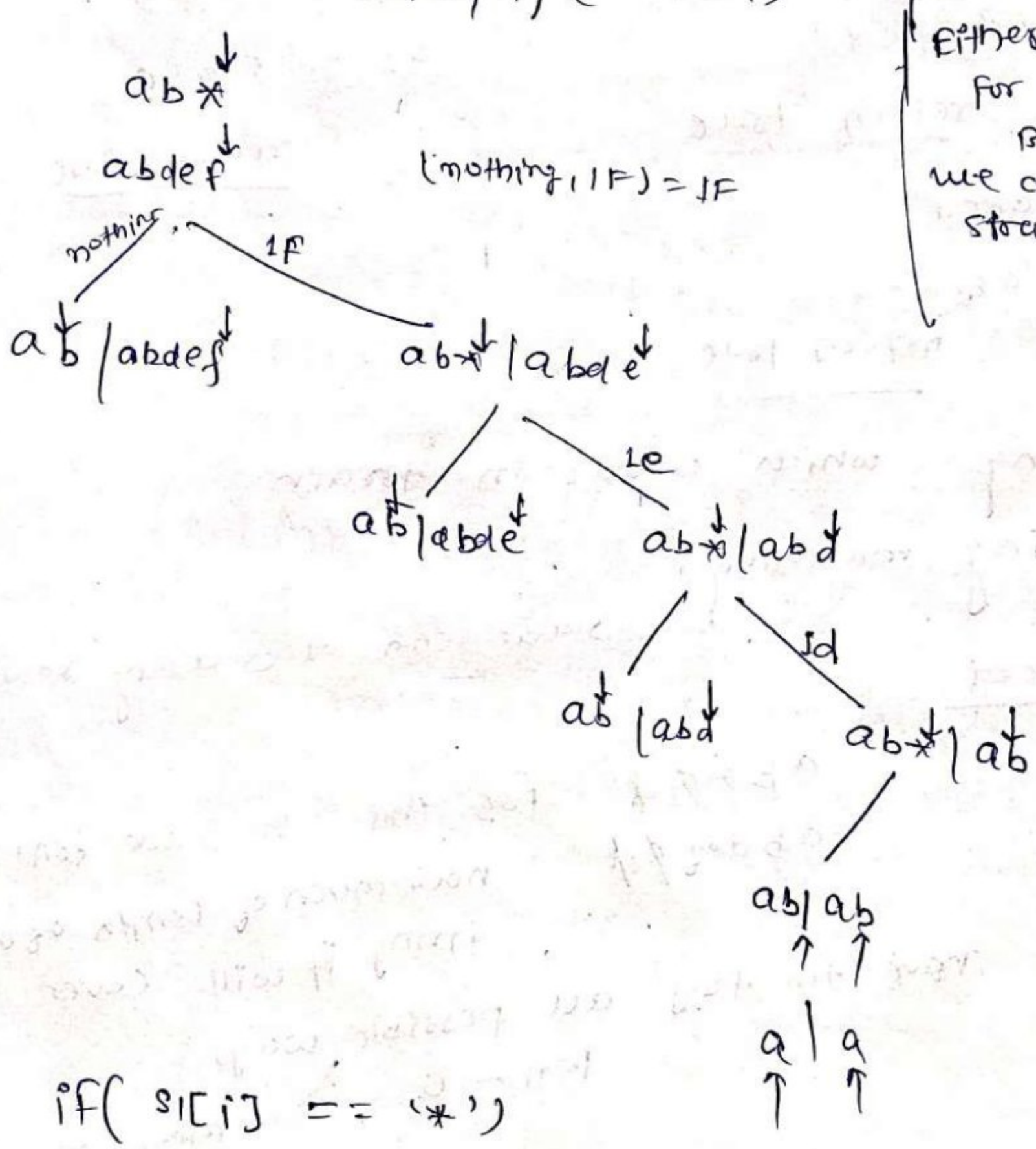
0	1	2	3	4
a	b	*	c	d

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

@Aashish Kumar Nayak

Base case → return true/false

Either S_1 gets exhausted

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

if ($\boxed{i<0 \ \&\& \ j<0}$) 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 if has to be all "*"

for (ii = 0 → i)

if (S1[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 (ai) (———)

<u>Recursion</u>

* <u>T.C. — O(exponential)</u>

<u>S.C. — O(N+M)</u>

Auxillary Stack space

↓

After <u>memoization</u>

T.C. O(N×m)

S.C. O(N×m) + O(N+m)

↑

<u>Ass</u>

@ Ashish Kumar Nayak

```
~~if(~~ 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++)   ii=1
{ if( Pattern[ii-1] != '*')
return false;
}
return true;
}
if (dp[i][j] != -1) return dp[i][j];
if (Pattern[i-1] == text[j] || pattern[i-1] == '?')
{
return ~~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 wildMatchingCard( 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);
}
```

@Aashish Kumar Nayak

Tabulation :-

@Aashish Kumar Nayak

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

① Base cases
② changing the parameter
③ copy the recurance

```
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]
```

@Aashish Kumar Nayak

```
        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;
              }
           }
         ~~dp[i][0] = flag;~~  cur[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] = false~~
```

```
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];
}
```

@Aashish Kumar Nayak

Scanned with OKEN Scanner

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

6 problems

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

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

Buy → 1
sell → 6
, only done once

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)

mi

{ 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

$\underbrace{1, 5}_{4} + \underbrace{3, 6}_{3} = 7$

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;

}

<u>Base case</u>

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.

<u>dp[ind][buy]</u> T.C. - O(1

@Aashish Kumar Nayak

```
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 ~~profit~~ 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 - $O(N \times 2) + O(N)$

1. Base Case
2. i $\rightarrow$ (n-1 $\rightarrow$ 0)
   Buy 0 $\rightarrow$ 1
3. copy the recurance.

@Aashish kumar Nayak

~~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];
}
```

@Aashish Kumar Nayak

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];
}
```

@Aashish Kumar Nayak

DP-37 Buy and sell stocks - III

Option stocks

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

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

2 + 3 + 3 = 8

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

sell int →

Recursion

T.C. → exponential

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

overlapping subproblems

Memoization

@Ashish Kumar Nayak

ind → 0 — n-1
buy → 0/1
cap → 0/1/2
ind → N

three changing parameter | 3D DP

dp[N][2][3] N×2×3

T.C. - N×2×3

S.C. - O(N×2×3) + O(N)

Asc recursion stack

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);
}
```

@Aashish Kumar Nayak

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

②

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

③ answer will be initial call i.e dp[0][1][2]

@Aashish kumar Nayak

Scanned with OKEN Scanner

```
int MaxProfit(vector<int> & prices, int n)
{ vector<vector<vector<int>>> dp(n, vector<vector<int>>(2, vector<int>(3,0)));
```

So we dont write base cases. if there would be something different like 1, 0 then we may write.

(.. already dp is full with 0.

```
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].
}
```

@Aashish kumar Nayak

Scanned with OKEN Scanner

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];
}
```

N×2×3

day Buy cap

dp [N×4]

B	S	B	S
4	1	1	1
0	1	2	3

transaction

four variables

X

Not intuitive

method in next question

DP-38 Buy and sell stocks - IV | DP on stocks

At max K transactions

K can be anything

Code is similar only change is K

Replace 2 with K

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);
}
```

@Aashish Kumar Nayak

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

@Aashish kumar Nayak

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.

@Aashish Kumar Nayak

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 ✗ B
B ✓

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

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

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

ind+2 — that day we can buy.

only this minor change on DP on stocks II problem we solved the DP on stock V

@Aashish Kumar Nayak

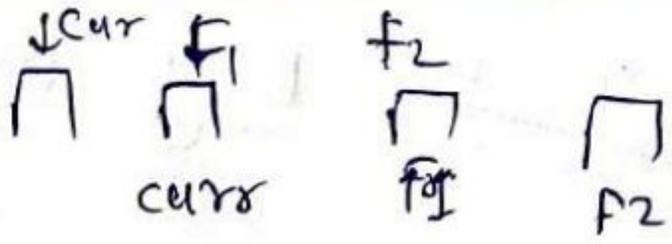
```
int f(int ind, int buy, vector<int> &prices, vector<vector<int>>&dp)
{
    if(ind >= prices.size())
        return 0;  if(dp[ind][buy] != -1) return dp[ind][buy];
    if(buy == 1)
    return dp[ind][buy] = max[-prices[ind] + f(ind+1, 0, prices, dp),
                              0 + f(ind+1, 1, prices, 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, 0, prices, dp),
        }
        else
        {
        dp[ind][buy] = max(prices[ind] + dp[ind+2][1],
                           0 + f(ind+1, 0, prices, dp)).
    }  }
    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. — $O(6) \approx O(1)$

@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(-price[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));

or

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

@Aashish 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 try out various subsequences → power set | or Recursion

↓

(2^n) Check for increase

exponential

Store the longest.

in order to generate all subsequence.

Trying all ways

↳ Recursion

[10, ~~9~~

prev=10 to check

Recurrance

(i) Express everything in terms of index

(ii) explore all possibilities in subsequence — ✓ ✗

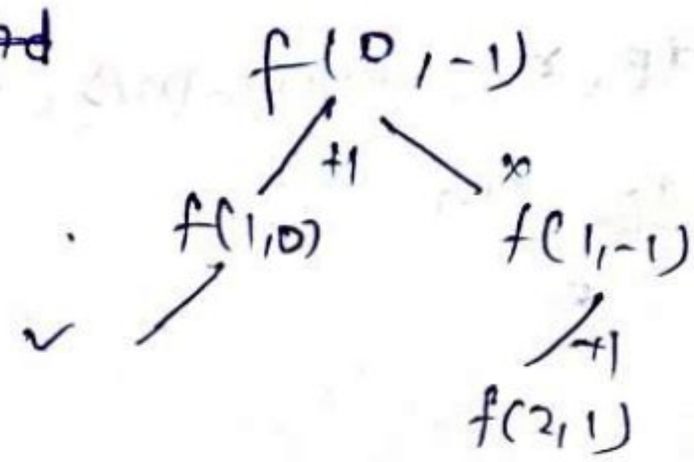
(iii) Take the max length

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~~



@Aashish Kumar Nayak

Base case :-

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

~~Recu~~ 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 — (n-1)

co-ordinate change

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

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

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

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 coordinates shift.

Code :

```
int longestIncreasingSubsequence(int arr[], int n)
{ vector<vector<int>> ~~dp(n+1, 0)~~;
                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 longestIncreaseSubsequence (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}

{3, 11, 16}

LIS → max of all dp index.

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

5	4	12	1	16	8

dp[n]

prev
x = 5

2
already updated

4 greater ✗

11 greater ✓

1 greater ✗

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 require if we want to trace the LIS.

@Aashish Kumar Nayak

DP-42

5	9	11	1	16	8

1	1	2	1	3	1

dp

0	1	2	3	2	5

hash

print the subsequence

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;
        }
    }
```

@Ashish Kumar Nayak

```
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;
}
```

$\boxed{T.C. - O(N^2)}$

@Aashish Kumar Nayak

DP-43 Longest Increasing Subsequence

In this lecture we will use Binary search $1 \le N \le 10^5$

LIS using Binary search

Recursion
Memoization
Space optimization
Tabulation
Algorithmic approach

in previous lect

T.C – $O(N^2)$
S.C. – $O(N)$
Still more

[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]

Eg1,

[1, 7, 8, 4, 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 new [1, 4 we will replace in 1st array Replace 4 with 7

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

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

Aashish Kumar Nayak

Binary Element el -> arr[i]

if(find arr[i]).

else find 1st greater of arr[i]

or

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 take less space from temp size()
```

lower bound.

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)
len = 4 — All the pairs are divisible this can be possible ans

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}
len = 4 — ans

∵ 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.

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[j] < arr[i]~~ && dp[i] < dp[j]+1)
      dp[i] = dp[j]+1
      hash[i] = j
  }
}
```

because it's already sorted just change 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" }

b c d

add b add d

{a, ba, bda}

add a add c

{b, ba, bca}

{a, ba, bca}

for(i=1 → n)

one line change.

{ for (j=0; j<i; j++) Compare(arr[i], arr[j])

{ if(~~arr[i] > arr[j]~~ dp[j]+1 > dp[i])

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

}

maxi = max(maxi, dp[i]);

}

return maxi;

b d c a

b d a

assume

arr[i]

b c d a

+a

arr[j]

b c d

b c d

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" (2), "pcxbcf" (5), "xb" (1), "cxbc" (3), "pcxbc" (4)]

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. - $O(N^2) \times l + O(n \log n)$ sorting.

DP-46 Longest Bitonic Subsequence | LIS

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

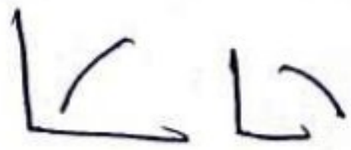
Bitonic → increasing then decreasing. or it can be just increase or just decrease

{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[i]+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, 2, 4, 3, 3, 2, 1}

2 → common

1 6 3 6 5 6 3 1

3 + 4 = 7 Ans

10 is common

1 = 6 Ans

@Ashish 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.

T.C. - O(N²)

@Ashish Kumar Nayak

DP-47 Number of longest increasing subsequence

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

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

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

dp[] → 1 ~~2~~ ~~2~~ ~~2~~ ~~2~~ ~~2~~ 3
cnt[] → 1 1 1 1 1 ~~2~~ 1 9
~~4~~ 5

4

1, 5

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

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, 1

{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];
      }
```

@Ashish Kumar Nayak

```
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;
}
```

@Aashish Kumar Nayak

DP - 48 Matrix chain Multiplication (MCM) Partition DP starts

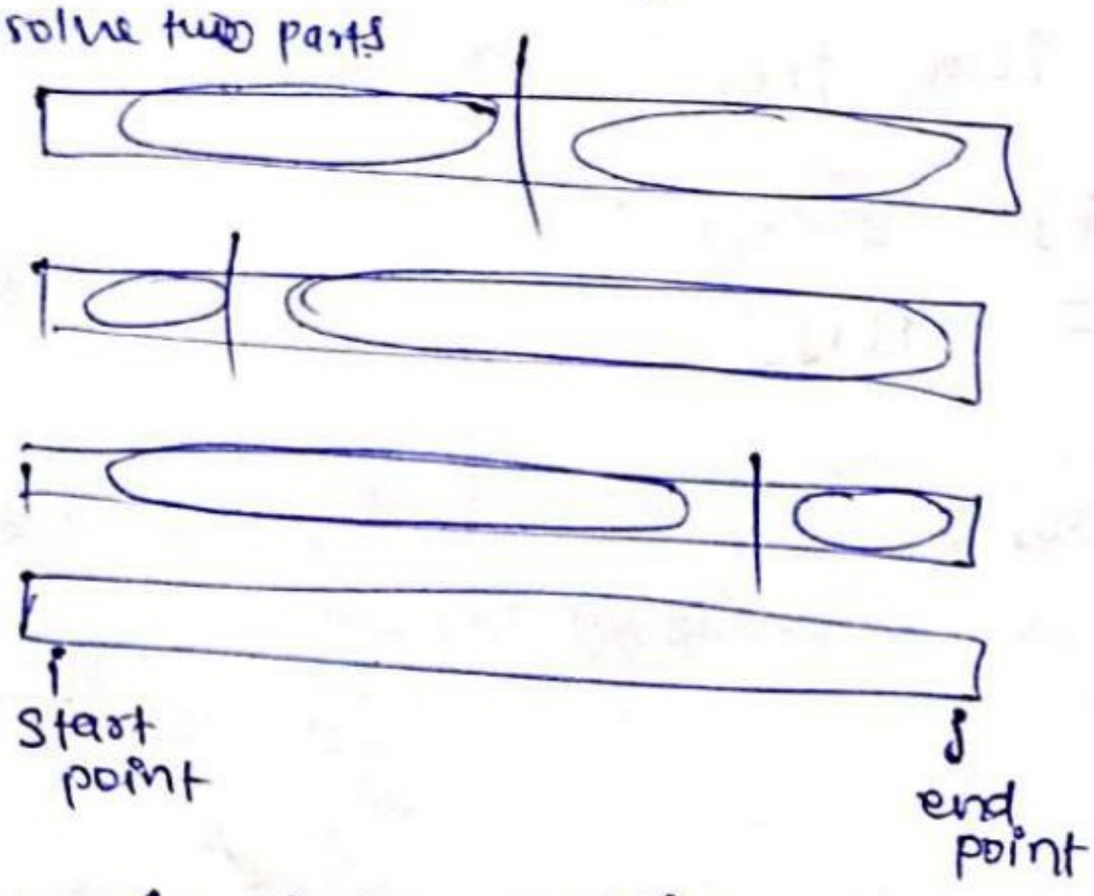
New pattern - [pattern DP] Tough pattern

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

$B = 30\times5$

$L = 5\times60$

$A \quad B$

$[\ ]_{10\times30} \quad [\ ]_{30\times5}$

$= 10\times30\times5 = 1500$ 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

$(A\ B\ C)$

$(AB)\ C$

$10\times30 \quad 30\times5$

$A = 10\times30$

$B = 30\times5$

$C = 5\times60$

@Ashish Kumar Nayak

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

$= 1500 + 3000 = 4500$

A(BC)

$[]_{10\times30}$ $[]_{30\times60}$ $(30\times\underline{5})$ $\underline{5}\times60)$

9000

$= 10\times30\times60 + 9000$

$= 18000 + 9000$

$= 27000$

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 → 10×20, B → 20×30, C → 30×40, D → 40×50

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

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

Approach Partition 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.
   ~~i — start point~~
3. Return the best possible 2 partition

[10, 20, 30, 40, 50]

A B C D

i → 1, j → n-1

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

© Aashish Kumar Nayak

Base case :-

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

Try all partions

$\{10, \overset{i}{20}, 30, 40, \overset{j}{50}\}$

(A) (B C D)

( ) ( )

( ) ( )

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

$K=1 \quad f(\overset{1}{i}, \overset{1}{K}), \; f(\overset{2}{K+1}, \overset{4}{j})$

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

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

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

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

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

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

$f(1,1) \quad f(2,4)$

(B C D)

$(10 \times 20) \quad (20 \times 50)$

(10×50) $\boxed{10 \times 20 \times 50}$ operations or steps

B C D

$(20 \times 30) \quad (30 \times 40) \quad (40 \times 50)$

B C D

$20 \times 30 \quad 30 \times 50$

$(20 \times 40) \quad (40 \times 50)$

$(20 \times 50) \quad (20 \times 50)$

$A = 10 \times 20$

$B = 20 \times 30$

$C = 30 \times 40$

$D = 40 \times 50$

Aashish Kumar Nayak

i j

10 20 30 40 50

A B C D

f(AB) + f(CD)

10×20 30×40 40×50

20×30

30×50

10×30

10×50

No. of steps

= 10×50×50

A[i-1]×A[k]×A[j]

Recursion Tree

(10, 20, 30, 40, 50)

A B C D

i j

∵ single matrix No multiply → 0

A

[20] [30, 40, 50]

i j

i j

B C D

AB CD

20 30 40 50

i j i j

A B C D

20 30 40 50

i j

(10 * 20×50) + f + f

B C D

30 40 50

i j

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

0 0

then take min of all.

Cong

@Aashish Kumar Nayak

Code :-

```
int f(int i, int j, vector<int> &arr)
{ if(i==j) return 0;
 int mini = 1e9;
 for(int k=i; k<j; k++)
   { int steps = arr[i-1] * arr[k] * arr[j] + f(i,k, arr,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:-

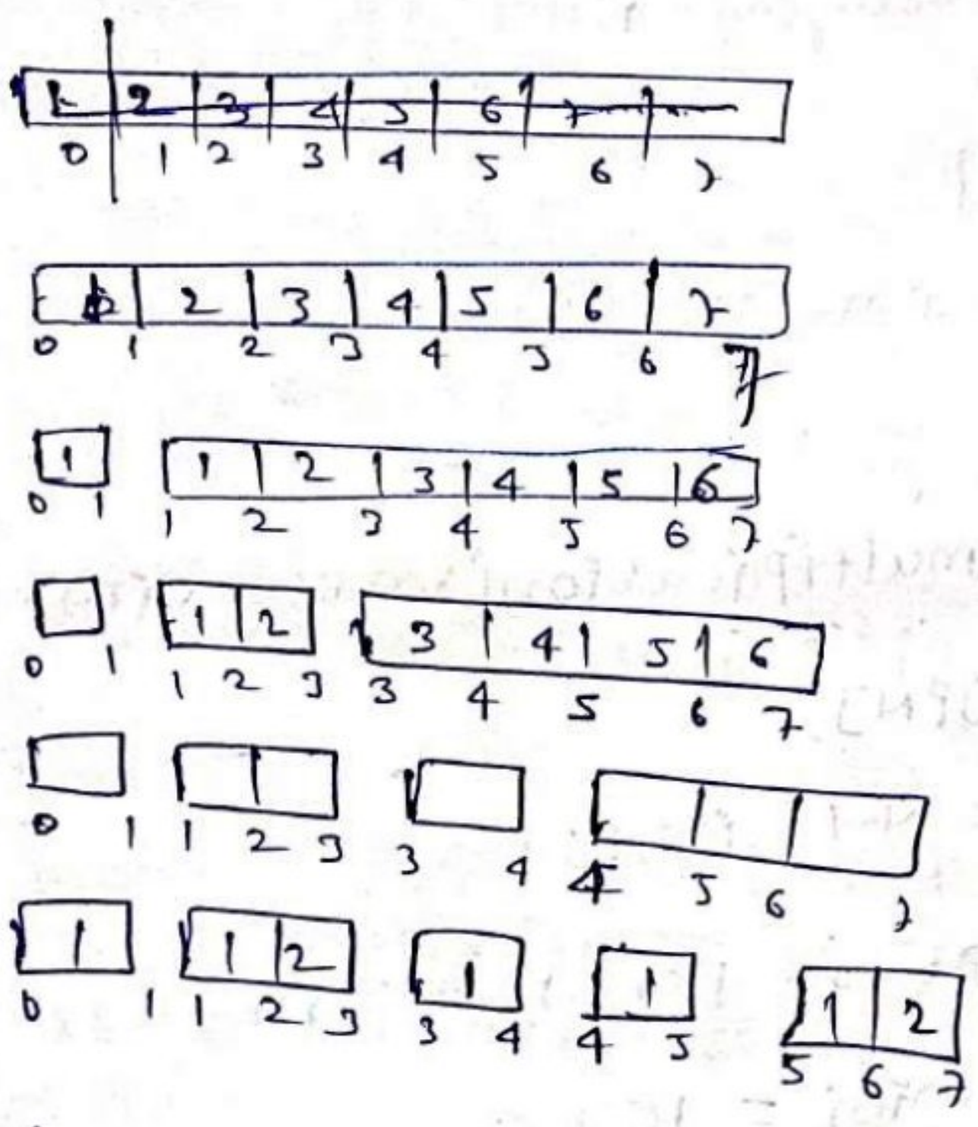
```
int matrixmultiplication(vector<int> &arr, int N)
{
    int dp[N][N];
    for(int i = N-1; i>=1; i--)
    {
        for(int j = i+1; j<N; j++)
        {
            int mini = INF;
            for(int k = i; k<j; k++)
            {
                int steps = arr[i-1]*arr[k]*arr[j]
             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


Scanned with OKEN Scanner

DP-50 Minimum cost to cut the stick

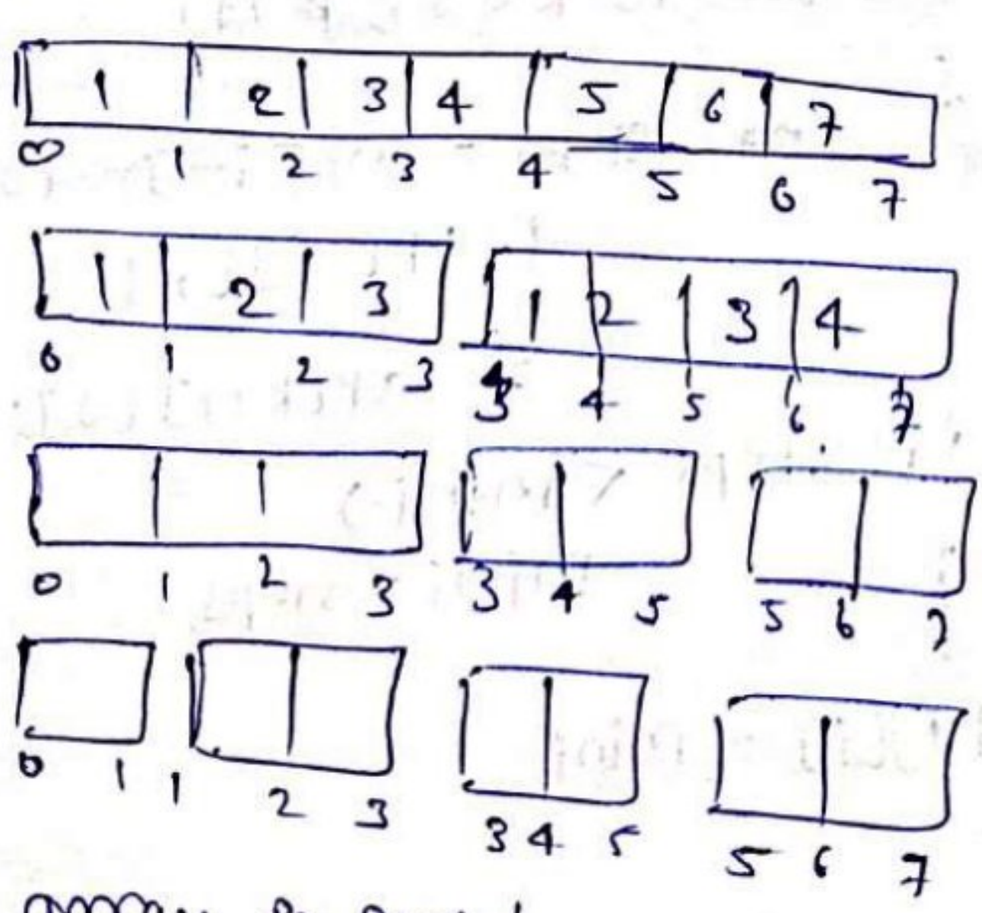
arr[] = [1, 3, 4, 5] n = 7

cost = length of the stick you are cutting



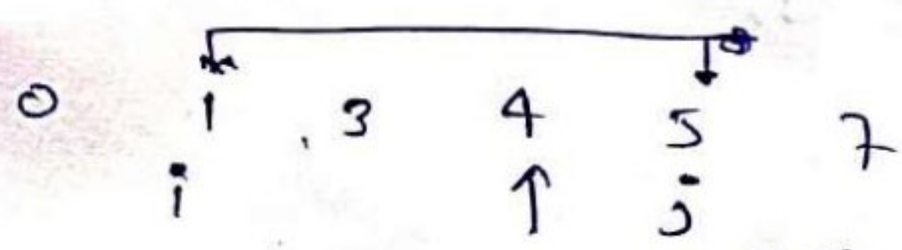
7 + 6 + 4 + 3 = 20

{3, 5, 1, 4}



7 + 4 + 3 + 2 = 16

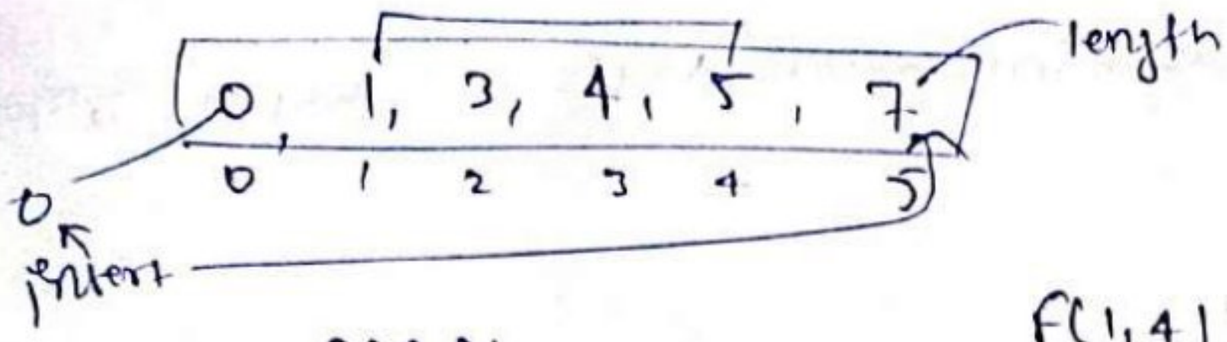
array is sorted



cuts[j+1] − cuts[i−1]

7 − 0 = 7

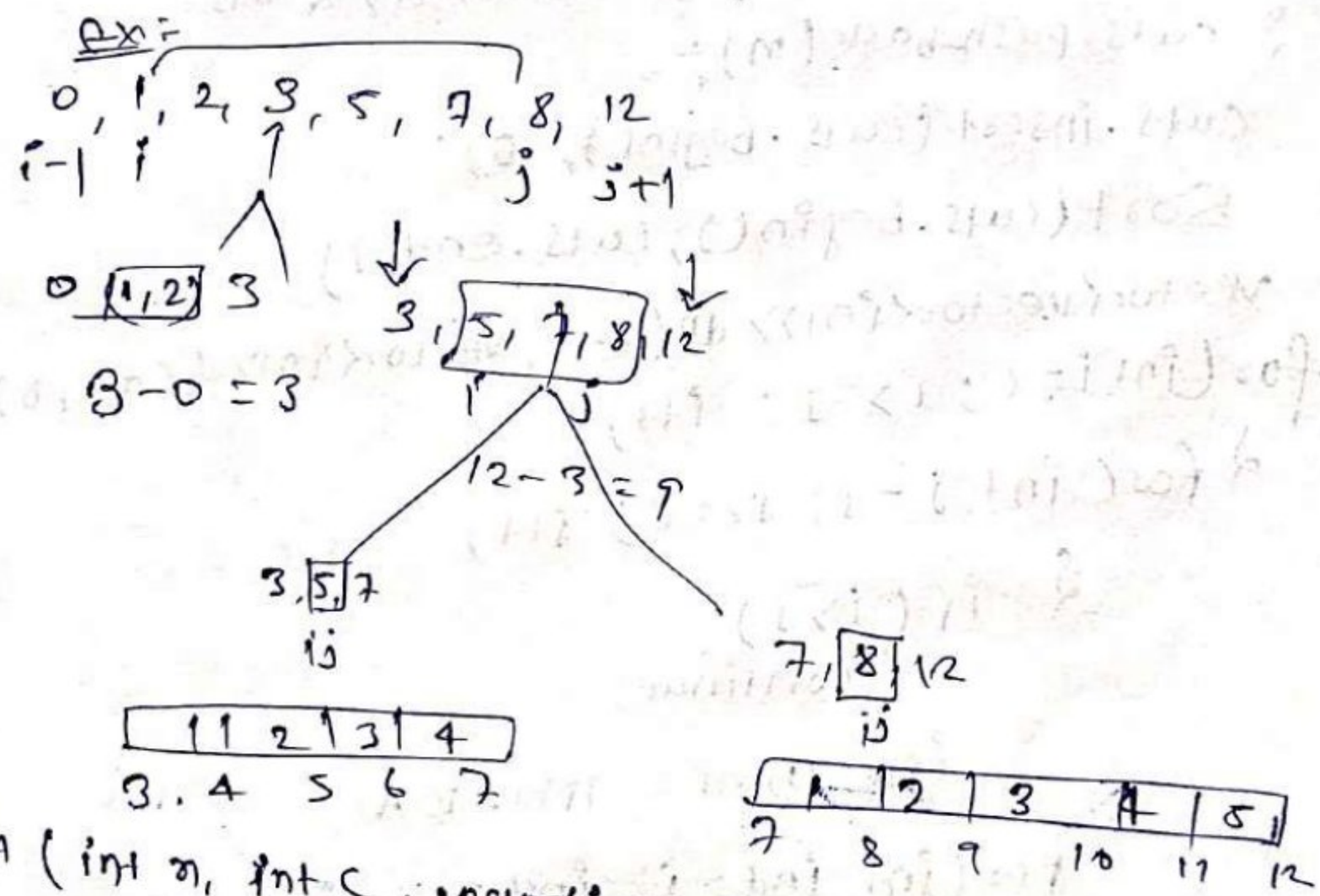
cost = length of that stick in which 4 is.....



f(1,4)

```
f(i,j)
    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(~~ vector<vector<int>> dp(c+1, vector<int>(c+1,-1));
        return f(1, c, cuts, dp);
    }
```

@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. $= O(M^2)$ + stack space

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];
}
```

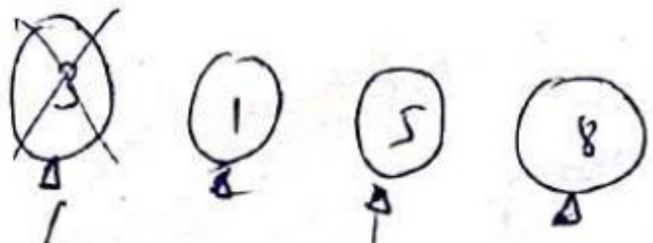
@Aashish Kumar Nayak

DP-51

Brust Ballons | partion DP

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

Summation of those coins ↑↑



if we burst 5

Left right

1×5×8 = 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

~~Mount~~ ~~Start from min value to Burst the Balon~~.

[3, 1, 5, 8]

i j

can be any one. 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

@Aashish Kumar Nayak

$[(b_1, b_2, b_3), \boxed{b_4}, b_5, b_6]$

$\{b_1, b_2, b_3, b_5, b_6\}$

if we take subpart as an individual subproblem and try to solve for b_3 then we can't become it. depends on other portion ~~b_4~~ i.e. b_4.

start thinking in opposite direction

Ex: $1\,[\,3 \;\; \cancel{7} \;\; \cancel{8}\,]\,1$

$[\,3, \cancel{7}, 8\,]$ $\quad 3\times1\times5 = 15$

$[\cancel{7}, 8\,]$ $\quad 3\times5\times8 = 120$

$[\cancel{8}]$ $\quad \rightarrow 1\times3\times8 = 24$

$\rightarrow 1\times8\times1 = 8$

Ex: $\{b_1, b_2, b_3, b_4, b_5, b_6\}$

$i \quad x \quad j$ (ind)

$= (a[i-1]\times a[ind]\times a[j+1]) + f(i, ind-1) + f(ind+1, j)$

$\{b_1, b_4\}$ or $\{b_2, b_4\}$ or $\{b_3, b_4\}$ or 2nd last Burst.

$[b_4]$ $\quad = 1\times b_4\times1$

(last Burst)

@Ashish Kumar Nayak

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

0 1 2 3 4 5

i j

f(1, 5) or f(1, 4)

```
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

i j

[N+1]

T.C. → N×N×M ≃ N^3

S.C. → $O(N^2)$ + $O(N)$ ASC

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;
}
```

@ Aashish Kumar Nayak

```
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];
}
```

T.C. $O(N^3)$

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

@Aashish Kumar Nayak

DP-52

Boolean Expression to True / partion DP

Compression = "T | T & F"

"T & F"

"F"

or → |
& → and
XOR → ^

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

"T | T & F"

"T | F" ① → no. of ways

"T"

"T ^ F | F & F ^ T | F"

(T) ^ (F | T & F ^ T | F) Similar to MCM

( ) | ( )

( ) & ( )

( ) ^ ( )

( ) | ( )

Top (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 ~~ind~~ left comes true or no. of ways in which right is true.

x + y.
left right

left | Right

T = x_1 T = x_2 = $x_1 \times x_2$

F = x_3 = $x_1 \times x_3$

T ^ T = F

F ^ F = F

T ^ F = T

F ^ T = T

@Aashish Kumar Nayak

left ∧ right

$T = x_1$ $T = x_2$

$F = x_3$ $F = x_4$

$x_1 \vee x_4$

$x_3 \vee x_2$

$\underbrace{\qquad\qquad}_{i \qquad j}$ → T, → F

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;
```

T | F ∧ T
i j

(T) (F ∧ T) (T | F) (T)
i j i j i j

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

looking for false.

→ 1 is false
0 is true.

go and try to make partition int ways ⇒

```
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);
```

@Aashish Kumar Nayak

~~ways = 0~~

if(a[ind] == '&')

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]

<u>Code</u>:

```
int mod = 1000000007;
long long f(int i, int j, int istrue, string &exp, vector<vector<vector<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);
```

@Ashish Kumar Nayak

```
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 +
                 rF(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 ways; 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);
}
```

@Ashish Kumar Nayak

DP-53 Palindrome partitioning - II | Front partition

Given string = bababcbadcede

small string str = aabb n=4

a|a|b|b n-1 partition

3 partion

every individual substring will be palindrome.

aa|bb

pal

a|a|b|b

Such problems are generally solved using front partition

Start from front

b | a b a b c b a d c e d e

1 + ( )

Palindrome

1 + (abcb - -)

1 + (ababcb - -)

1 + (cbadcede)

3 subproblems take the minimum no. of cuts.

take this one

1 + (abcbadcede)

1 + (dcede)

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.

@Aashish Kumar Nayak

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

Recursion → T.C. — O(Exponential)

↓ Memoization

Changing parameter

$0 \rightarrow N$

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;
}
```

@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. $O(N \times N) = O(N^2)$

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

Tabulation :-

int (1) Base case $(i == n)$ $\rightarrow 0$

(2) $i = n-1 \rightarrow 0$ $dp[n] = 0$;

(3) copy the recurrance

```
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

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

Sum = 77

[1 15 7 | 9 | 2 5 10]

[15 15 15 9 10 10 10]

all elements of partition converted into max element

sm = 84 ↑↑

Maximum

front partion logic ⟷

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.

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

↑ ind

f(0)

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

f(3)

~~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~~ 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 5
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)

Scanned with OKEN Scanner

```
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-1 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];
}
```

@Aashish Kumar Nayak

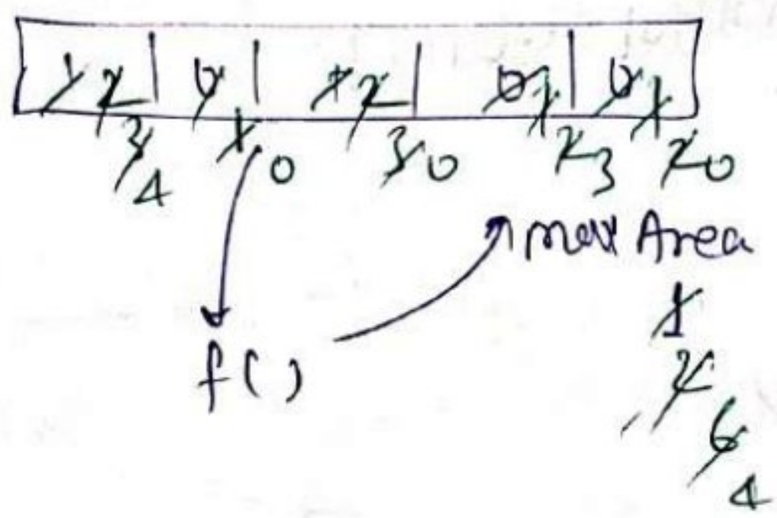
DP-55

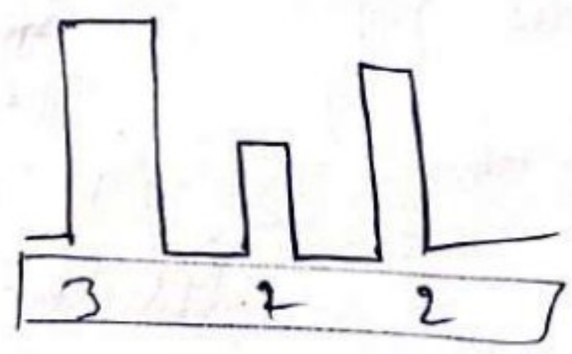
Maximum Rectangle area with all 1s | DP on Rectangles

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





max = 6

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

Code:

```
int maximalAreaofSubmatrixofAll( vector<vector<int> &mat,
                                 int, 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
thats why this is kept in dp.

@Ashish Kumar Nayak

Scanned with OKEN Scanner

DP-56 Count No. of square Submatrices

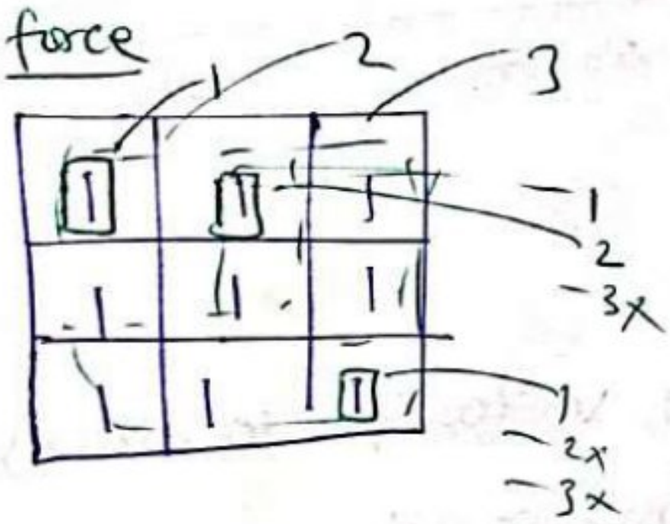
1	0	1
1	1	0
1	1	0

Size 1 — 6
Size 2 — 1

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

size 1 — 10
size 2 — 4
size 3 — 1

Brute force



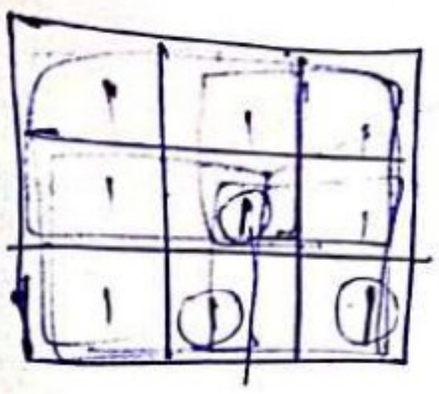
We can do better using dp.

Try to create dp of similar size

dP[i][j]

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

Try



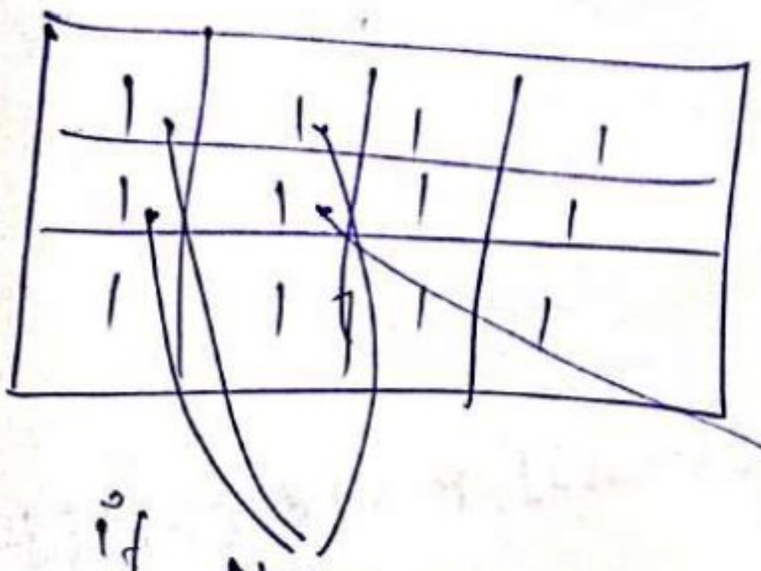
	0	1	2
0	1	1	1
1	1	2	2
2	1	2	3

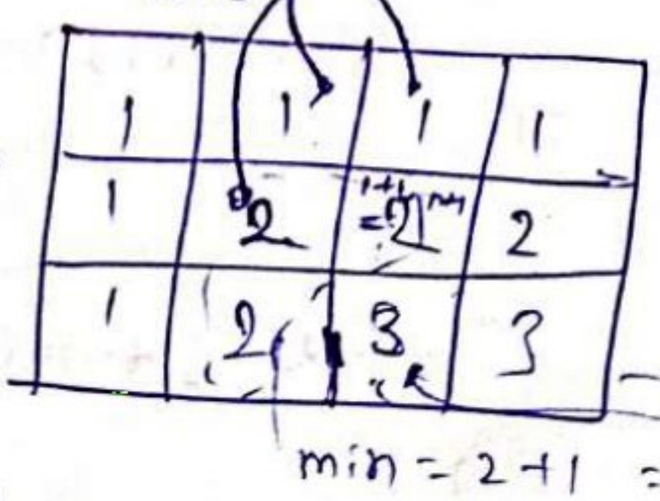
1 sq. he himself
+ he is right most bottom
= 2

sum all of them we will get our ans.

but How we will fill this array.

take minimal of other and adds





min = 2+1 = 3

first row and first column thog themselves are i.e. 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.

@Aashish Kumar Nayak

@Aashish Kumar Nayak

1	1	0
1	1	1
1	1	0

→

1	1	0
1	2	min=0+1 1
1	2	0

— Now add all of them.

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

$$dp[i][j] = \min(\underbrace{dp[i-1][j]}_{\text{upper}}, \underbrace{dp[i-1][j-1]}_{\text{diagonal}}, \underbrace{dp[i][j-1]}_{\text{left}}) + 1$$

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.