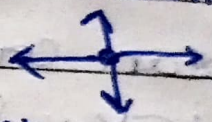
DP

DELTA Pg No. Date / /

DP on Grid

1. whenever we talk about dp on grid
2. what's the pattern

- we have given a matrix of n x m or may be n x n.
- we have to reach on (n-1, m-1) by traversing on grid.
- Also we have some sort of direction given like you can move right, left, up, down
- And at last we have to find maximum or minimum according to question



- game goes on here approach

4 write down base case

if(i<0 || j<0 || i>n-1 || j>m-1) return

- this may be vary with question to question.

→ now talk about direction

up = solve(i-1, j)

down = solve(i+1, j)

right = solve(i, j+1)

left = solve(i-1, j)

- return it according to question ask

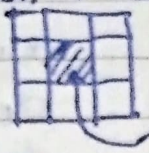
NoteBook

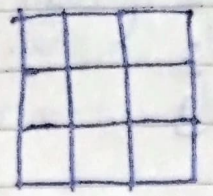
Sign.

Page No.
Date : / /

Top-down approach

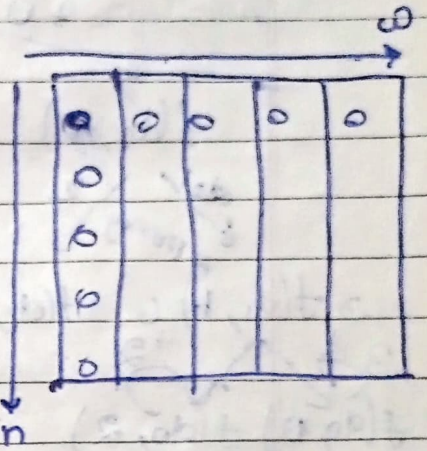
Recursive → Base Condition + RC

memoized → RC +  (-1) hai ki value hai
(DP)

In Top down →  only this no RC
(Best) and ⇒ DP
this is better bcz
Don't give stack overflow

~~tundey/Stand-alp problem~~

In Top down approach



Code Studio> Ninja training

→ Ninja plan N day learning, he have 3 activities and each having some points of each day, he can't do the same activity in two consicutive day

→ given 2D array of N*3 point

→ mtlb ninja ko max output chiye or greedy maxim de bhi sakta hai or nhi bhi jo abhi tak sikha so ninja pr hum dp lagenge because we have

→ optimal
→ Choose
→ max

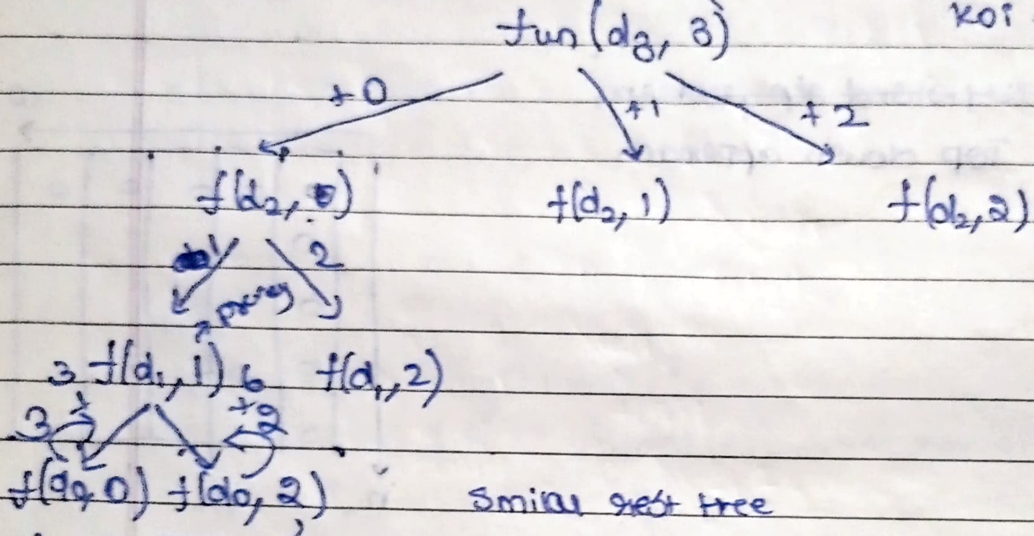
n=4

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

ninja ko aaj ka batana ke liye ki same task do din na ho jaye hume previous ka bhi to record rakhna padega ki previous day usne konsa kaam kiya hai.

→ mai niche se start karta hu

recursion tree



overlapping occur here.

at base

d_0 pr hu ab mujhe maximum lena hai jaha se mujhe aage ki nhi sochni

so we have to two parameters

first	second
days	last task that done.
↓	{0, 1, 2, 3}
N	first task, 2nd task, 3rd task, none task

So dp[N][4]

Page No.
Date : / /

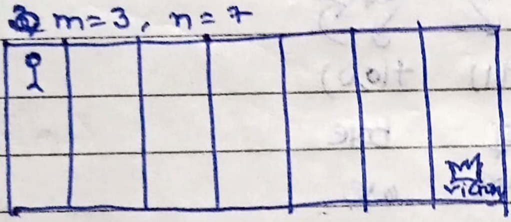
Sudo code

```
f(day, last)
    if(day==0)
        int maxi=0
        for(i=0 -> total day here 3 task)
            if(i != last){
                maxi=max(maxi, point[0][i]);
            }
        }
        return maxi
    }
    if(dp[day][last]!=-1) return dp[day][last];
    int max
    -> Same code for day not =0
```

Unique path

→ A bot at (0,0) initially, have to reach last row ke last column or & he move right & Down only.

m=3, n=7



give all unique path to reach at this

→ How to write recursive
↳ Express on term of (row, col)
↳ Explore/DO all stuff
↳ sum up all ways/max/min

f(i, j) → (m-1, n-1)

(0,0) → (m-1, n-1) tak jana hai mujhe.

Base case

agar ((i==0 || j==0) return 1

if(i<0 || j<0) return 0; } out of band case boundary ke bhar kaha se milega

A se B jana hai to diagonal condition hai upper ya same or left

Page No.
Date: / /

```
up = f(i-1, j);
left = f(i, j-1);
return up + left;
}
```

(B)

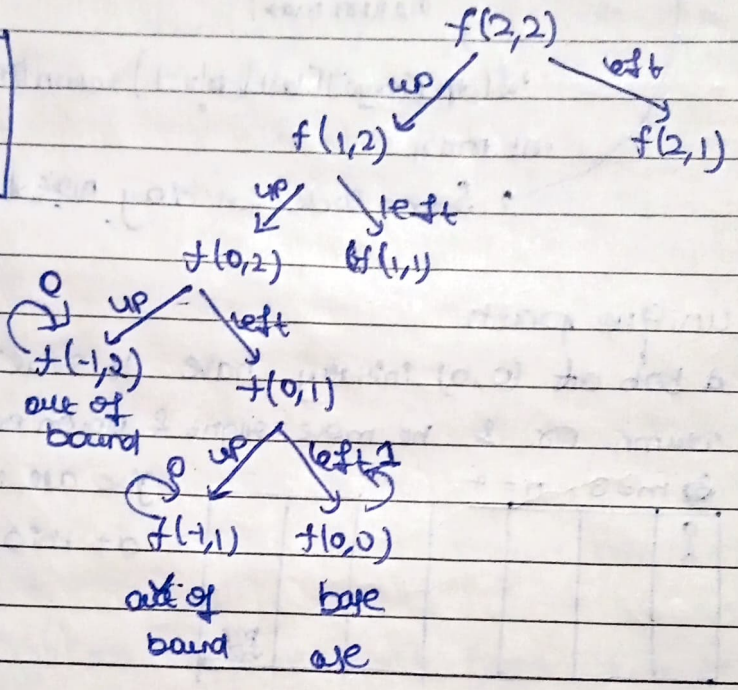
(A)

recursion

Tc → for every grid have take two option up, left. up, left so $(2^{m \times n})$

m=3, n=3

	0	1	2
0	0,0	0,1	0,2
1	1,0	1,1	1,2
2	2,0	2,1	2,2



Here overlapping occur if we see

so overlapping

memoization

Parameters ij

(i, j)

max (m-1) max (n-1)

so dp[m] x [n] will get this

So here

TC → O(N x m)

SC → O((n-1) + (m-1)) + O(n x m)

DP

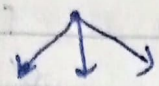
Here for every both of bob Alice have 3 choice like for [j=0] bob Alice choice -1, 0, +1 or [j=-1] Alice can choice -1, 0, +1 similarly for row

3×3 path.

Cherry Pick II

3	1	1
2	5	1
1	5	5
2	1	1

moves → (i+1, j-1), (i+1, j), (i+1, j+1)



→ Pass cell can pick up all cherries & cell become empty
→ both at same cell, one take cherries
→ cannot move outside grid, both have to reach bottom

max (All path by Alice + All path by bob)

↓ recursion ↓ recursion [Move together]

Rules

① Explore everything in terms of (i_1, j_1) & (i_2, j_2)
② Explore all the paths ↙ ↓ ↘
③ max sum $f(0,0, 0, m-1)$
alice standing → bob

$f(i_1, j_1, j_2)$ → because both are starting from same row

// base case → out of bound and row of both are same in future also so it is common

if ($i_1 \geq m$ || $j_1 < 0$ || $j_2 < 0$ || $j_2 \geq m$)
return -1e8

// reach destination
if ($i == n-1$)
{ // both can be at one location only one count
if ($j_1 == j_2$) return $a[i][j_2]$
else return $a[i][j_1] + a[i][j_2]$
}

int maxi

for ($d_1 \to -1$ to $+1$)
for ($d_2 \to -1 \to +1$)
{ if ($j_1 == j_2$) maxi = max(maxi, $a[i][j_1] + f(i+1, j_1+d_1, j_2+d_2)$)
else maxi = max(maxi, $a[i][j_1] + a[i][j_2] + f(\ ) + j_1 + d_1 + j_2 + d_2$)

TC → $(3^n \times 3^n)$ ⇒ exponential } overlapping
SC → $O(N)$

DELTA | Pg No. | Date

Dp on Subsequence

- pattern
  - → when you have an array and you have to find element which sum up to given to target element

1	2	3	4

target = 4

{1, 3} {4}

- → Here we have to take element which we can consider & skip element which not.
- → take & not-take Problem

take = arr[i] + solve(

- have to find base case first

if (i == n-1)

if (arr[i] == target), consider it also

}

take = arr[i] + solve(i+1, target - arr[i]

not_take = solve(i+1, target)

return max(take || not_take)

Sign.

Date / /

Subset sum equals to target

0 1 2 3
arr = [1, 2, 3, 4] target = 4

(To hume base check karna ki target nhi hua hume. check karna hai ki nhi ki nhi aisa subseq jiska sum 4 ho)

f(ind, target)
// target achived (base)
if (target == 0) return True
if (ind == 0) return (a[0] == target) → If only one element

bool not-taken = f(ind-1, target)
bool taken = false;
if (target >= a[ind])
taken = f(ind-1, target - a[ind]);
return (taken) (or not taken)
}

currently → TC → 2^n
add dp[ind][target] in above SC → O(n)
memoization → ind target
dp $[10^3+1] \times [10^3+1]$ → given

Tabulation
① Base cases
② form the nested loop
ind → (1 → n-1)
target → (1 → target)

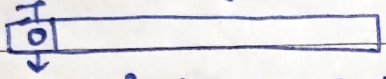
n-1
↙
n-2
↙
n-3
↙
0 → already done
target
↙
target - nums[0]
↙
0

Page No.
Date : / /

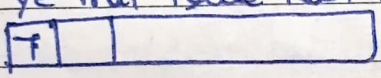
```
for(i=0 → n-1)
    d[i][0] = true;        → target 0 hai to
    d[0][a[0]] = true;     → ek element hai arr mai to
    for(i=1 → n-1)
      for(target = 1 → k)
        bool not take = dp[ind-1][target];
        bool take = false
        if(arr[ind] <= target) take = dp[ind-1][target-arr[i
        dp[ind][target] = take | not Take
      }
    }
    return dp[n-1][k];
}
```

Here also we can optimise its space complexity

Har time mai aage badha



ye mai True kar bi skta hu bake case



curr → curr mai bhi zero wala mai True kar hi sk so

space

```
bool subset( int n, int k, vector<int> &arr){
    vector<bool> prev(k+1,0), curr(k+1,0);
    prev[0] = curr[0] = true
    prev[arr[0]] = true
    for( ind → 1 → n,)
      for( target 1 → i<=K)
        bool not taken = prev[target]
        bool taken = false
        if(arr[ind]<=target) taken = prev[target-arr[in
          dp[curr[target] = take | not Take
        }
        prev = curr;
    }
    return prev[k];
```

Page No.
Date : / /

counts subset with sum k

(arr, k)
target

target
1, 1, 4, 5, [5]
{1,4}
{1,4}
{5}
} 3 ways

recurrence rule
1) Express in terms of (ind, target)
2) Explore all possibilities
3) sum all possibilities & return

start from f(n-1, S)

1) if (i==0) return 0/1; same
if (S==0) return 1

if (i==0) return (a[i]==S)

not_pick = f(ind-1, S)

pick=0
if (a[ind]<=S) pick=f(ind-1, S-a[ind])

initialization
if (a[0]<=tar) dp[0][a[0]]=1

return pick + not_pick;
}

Recursion TC → O(2^n)
agar recursion hai upper to overlapp karega hi so use memoization

Tabulation
① Base case
② Look at the changing Parameter & write Nested loop
③ Copy the recursion.

Another dp on Subsequence Problem

Page No.
Date : / /

Coin Change II

1	2	3

target = 4

any element can be used any no. of times

<1,1,1,1> = 4 → 1 way
<1,3> = 4 → 1 way
~~<3,1> = 4 → 1 way~~
<2,2> = 4 → 1 way
<1,1,2> = 4 → 1 way
~~<2,1,1> = 4 → 1 way~~
~~<1,2,1> = 4 → 1 way~~

} 5 ways

recursion
1 → base case
2 → convert all in i, T format
3 → sum up all possibilities & return

```
f(ind, T)
{
  // base case
  if(ind == 0)
  {
    return (T % a[0] == 0);
  }
  not_take = f(ind-1, T);
  take = 0
  if(arr[ind] <= T) take = f(ind, T-a[ind])
  return not_take + take;
}
```

Dp on String

- whenever we ~~have~~ are gonna heard about dp on string
- first thought that would gonna be come in mind fast is longest common subsequence

S_1 = adebc S_2 : dcadb

- we can't change order

so longest is adb

so what it's trigger ~~as~~ How we can start it

adebc ↑ i   dcadb ↑ j

```
if( S1[i] ==S2[j]
return 1 + solve(i-1, j-1).
else{
      return 0+ max (solve(i-1,j), solve(i,j-1))
```

// base case

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

	0	d	c	a	d	b
0	0	0	0	0	0	0
a	0	0	0	1	1	1
d	0	1	1	1	2	2
e	0	1	1	1	2	2
b	0	1	1	1	2	3
c	0	1	2	2	2	3

Its memo is

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

for (int i=1 ; i<=n ; i++)
    for(int j=1 ; j<=m ; j++)
        if(s1[i] == s2[j]){
            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];
```

• → minimum number of steps required to make word_1 to word_2

w_1 = sea ($n-1$) w_2 = eat ($m-1$)

```
int LCS = solve(n-1, m-1)
return (n - LCS) + (m - LCS)
```

LSC → code is same as before above

→ minimum insertion to make string palindromic

S_1 = abcaa

- create a new string S_2 which is reverse of S_1 & just do LCS

→ Stocks Problem

Best time to Buy & Sell Stocks II

[7, 1, 5, 3, 6, 4]

day = 0

profit = 0 → buy

- at each step we have two option take it or not take it ; int profit = 0

if (buy == 0)

profit = max(solve(a, i+1, 0, ap), -a[i] + solve(a, i+1, 1, ap)) (not take)

if (buy == 1) { we have to sell it now we have two option also

sell it currently or later

profit = max(solve(a, i+1, 1, ap) + a[i], solve(a, i+1, 0, ap))

DELTA Pg No.
Date / /

Dp on LIS

array = [10, 9, 2, 5, 3, 7, 101, 18]

↓

[2,5,7,101] or [2,5,7,18]

ans = 4

→ Here we take small subset of it

[4 2 6 8] answer should be
0 1 2 3 [4,6,8] or [2,6,8]

prev_index = -1

we have option take current or not.

- not_take = 0 + solve(i+1, prev_index)

or take

if(prev_index == -1 || arr[i] > arr[prev_index]){

take = 1 + solve(i+1, i)

}

return max(not_take, take)

}