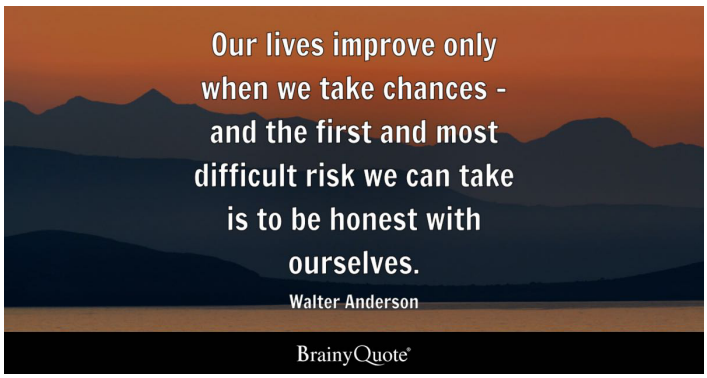
DP 2



Good Evening

Content

(a) Maximum subsequence sum (House robber 1)

(b) Count unique paths

(c) N digit no.s

(d) Catalan no.

(e) No. of unique BSTs

Max subsequence sum

01. Given A[N], calculate max subsequence sum

Note 1 → In a subseq, 2 adj ele can't be picked

Note 2 → Empty subseq is also valid

A[] = {9 14 3} →

Ans = 14

{- - -} = 0

{- - 3} = 3

{- 14 -} = 14

{- 14 3} ✗

{9 - -} = 9

{9 - 3} = 12

{9 14 -} ✗

{9 14 3} ✗

A[]: {9 4 13 24}

Ans = {9 24} ⇒ 33

A[] = {10 20 30 40}

Ans = {20 40} = 60

A[] = {-4 -3 -2}

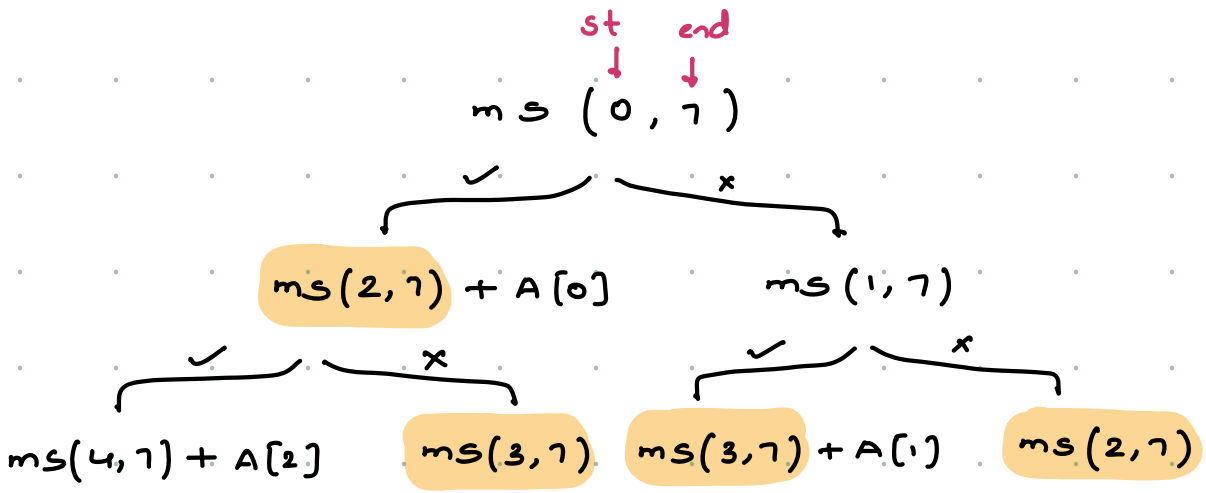
Ans = { } = 0

Brute force → Generate all the valid subsequences

for every element, provide a choice to be picked or not picked.

A[] = {2 -1 -4 5 3 -1 4 2}

0 1 2 3 4 5 6 7



* Top Down Approach

```
main( ){
    int [] dp = new int [N];
    Arrays.fill(dp, -∞);
    return maxSub(A, 0, n-1, dp)
}

int maxSub(int[] A, int st, int end, int[] dp){

    if (st > end) return 0;

    if (dp[st] != -∞) return dp[st];

    int pick = maxSub(A, st+2, end, dp) + A[st]

    int npick = maxSub(A, st+1, end, dp)

    return dp[st] = max(pick, npick);

}
```

TC : O(N)

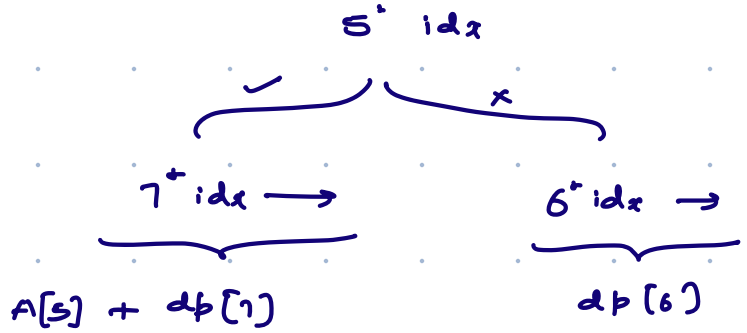
SC : O(N)

* Bottom Up Approach

A[] = {2 -1 -4 5 3 -1 4 2}
0 1 2 3 4 5 6 7

dp[] = {11 9 9 9 7 4 4 2}
0 1 2 3 4 5 6 7

dp[i] = max subsequence sum from i to n-1 such that there are no two adjacent elements present.




```
int [] dp = new int [n+2];
for (i=n-1; i≥0; i--){
    dp[i] = Max ( A[i] + dp[i+2] , dp[i+1]);
}
return dp[0]
```

i+2<n ? A[i]+dp[i+2] : A[i]

i+1<n ? dp[i+1] : 0

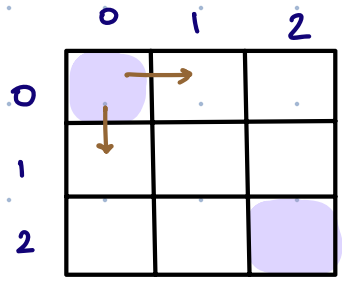
TC : O(N)
SC : O(N)

Q Number of ways to go from (0,0) → (n-1, m-1)

mat[N][M]

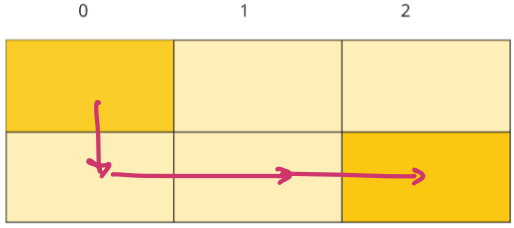
Note :- From cell we can go to right or down

Eg :-



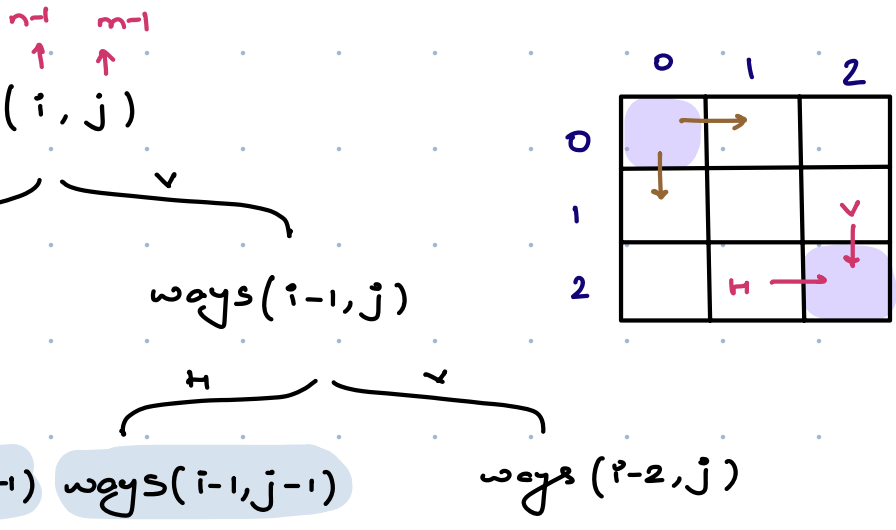
HHVV
HVHV
HVVH
VVHH
VHHV
VHVH

Ans = 6



HHV
HVH
VHH

Ans = 3




```
int [ ] [ ] dp = new int [n] [m];
```

```
int ways(i, j, dp)
{
    if (i==0 && j==0) return dp[i][j] = 1;
    if (i<0 || j<0) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    ans = ways(i, j-1) + ways(i-1, j);
    return dp[i][j] = ans;
}
```

n-1, m-1

TC: O(n*m)
SC: O(n*m)

	0	1	2
0	1	1	1
1	1	2	3
2	1	3	6

```
int [][] dp = new int [n][m];
// fill 0th row & 0th col with value 1

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

TC: O(n*m)
SC: O(n*m)

* N Digit number

10:07 pm → 10:17pm

Find out the number of A digit positive numbers, whose digits on being added equals to a given number B.

Note that a valid number starts from digits 1-9 except the number 0 itself. i.e. leading zeroes are not allowed.

A = 2

B = 4

{ 2 2
1 3
3 1
4 0 }

Ans = 4

A = 1

B = 3

{ 3 }

Ans = 1

nod = 2

sum = 3

{ 1 2
2 1
3 0 }

Ans = 3

nod = 3

sum = 4

{ 2 2 0
1 2 1
2 1 1
2 0 2
1 1 2
4 0 0
1 0 3
⋮ }

Brute force Idea → Iterate on all the A digits no. & check the sum of the digits.

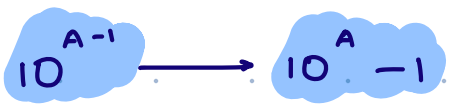
3 digit No.

100 → 999

4 digit no

1000 → 9999

A digit No.

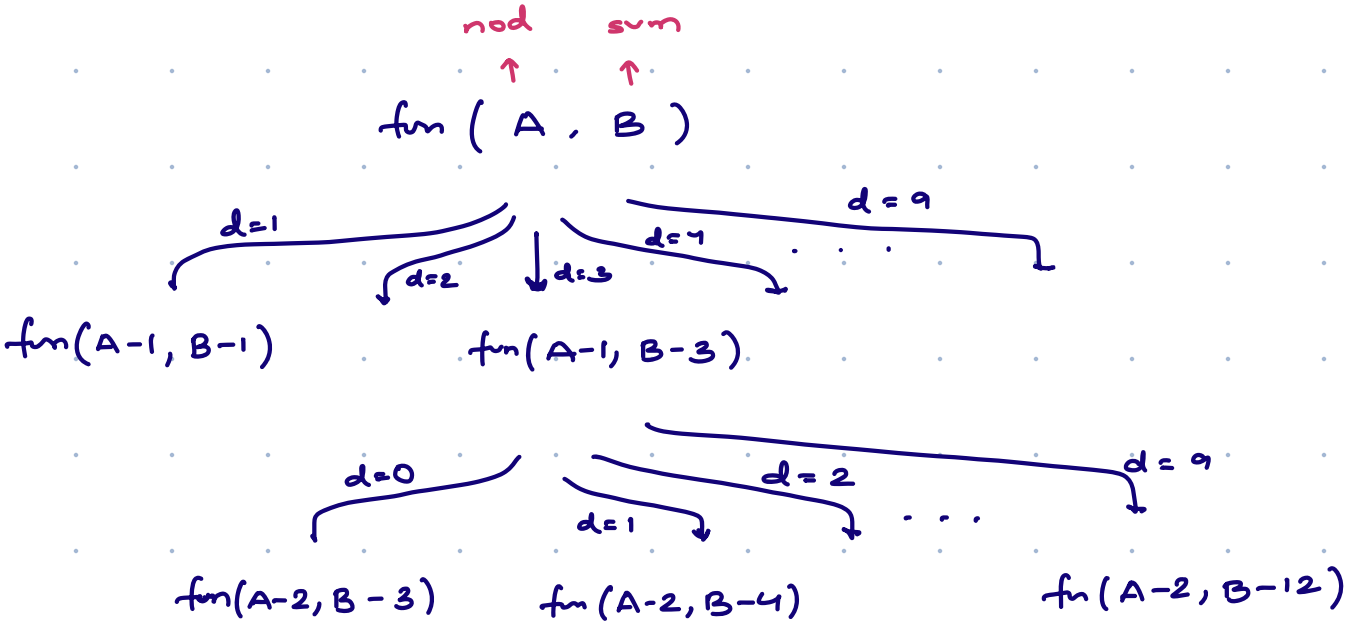


TC : $O(10^A - 10^{A-1}) * A$

SC : $O(1)$

```
for ( int num = 10^(A-1) ; num < 10^A ; num++ ) {
    int digsum = 0
    int temp = num;
    while (temp > 0) {
        digsum += temp % 10;
        temp = temp / 10;
    }
    if (digsum == sum) ans ++;
}
```

* Recursion



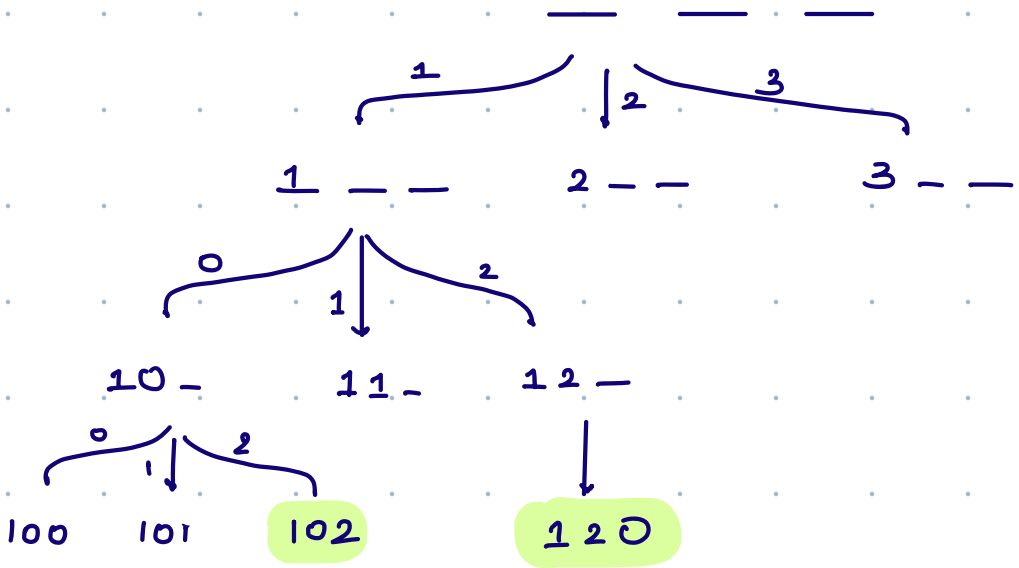

```
main ( ) {
    ans = 0
    for ( i=1 ; i<10 ; i++) {
        if (B-i ≥ 0) {
            ans += fun(A-1, B-i);
        }
    }
}

int fun ( int A , int B ) {
    if ( B < 0 || A < 0 ) return 0;
    if ( A == 0 && B == 0) return 1;
    int ans = 0
    for ( i=0 ; i<10; i++) {
        if (B-i ≥ 0) {
            ans += fun(A-1, B-i);
        }
    }
    return ans ;
}
```

TC : $O(10^A)$

SC : $O(A)$

N = 3

S = 3




```
main (){
    int [][] dp = new int [A+1][B+1];
    solve ( A, B, True , dp )
}

int solve ( nod, sum, boolean isFirstdig , dp ){

    if ( nod == 0 && sum == 0) return 1;

    if (nod == 0 || sum < 0) return 0;

    int ways = 0

    int st = isFirstdig == true ? 1 : 0;

    for ( int d = st ; d ≤ 9; d++){
        if ( dp[nod][sum] != -1) return dp[nod][sum];
        ways += solve (nod-1, sum-d, false, dp )
    }

    return  dp[nod][sum] = ways;
}
```

* Bottom Up Approach

dp[i][j] = count of i digit no. with digsum = j

for i=1 { if (1 ≤ j ≤ 9) dp[i][j] = 1 ; else dp[i][j] = 0 }

```
for (j=1 ; j ≤ B ; j++){
    dp[1][j] = 1
}
```

i = nod
j = sum

```
for ( i=2 ; i ≤ N ; i++){
    for ( j=0 ; j ≤ S ; j++){
        for (d=0 ; d ≤ 9 ; d++){
            if (d ≤ j){
                dp[i][j] += dp[i-1][j-d];
            }
        }
    }
}
```

TC : O(A*B)
SC : O(A*B)

i \ j	0	1	2	3
0	0	0	0	0
1	0	1 [1]	1 [2]	1 [3]
2	0	1	2	3
3	0	1	3	6

N = 3
S = 3
} Ans = 6

i , j
nod , sum

2 , 2

d = 0 → 2-1 , 2-0 = dp[1][2]

d = 1 → 2-1 , 2-1 = dp[1][1]

d = 2 → 2-1 , 2-2 = dp[1][0]

* Catalan number

$C_0 = 1$

$C_1 = 1$

$C_2 = C_0 * C_1 + C_1 * C_0 = 2$

$C_3 = C_0 * C_2 + C_1 * C_1 + C_2 * C_0 = 5$

$C_4 = C_0 * C_3 + C_1 * C_2 + C_2 * C_1 + C_3 * C_0 = 14$

$$C_N = C_0 * C_{N-1} + C_1 * C_{N-2} + C_2 * C_{N-3} + \cdots + C_{N-1} * C_0$$

$$C_N = \sum_{i=0}^{n-1} C_i * C_{n-1-i}$$

```
C[0] = 1
C[1] = 1

for (i = 2; i <= n; i++) {
    for (j = 0; j < i; j++) {
        c[i] += c[j] * c[i-j-1];
    }
}
```

$C_3 =$

0	1	2	3
1	1	2	5

$i = 3$

$j = 0, 1, 2$

$$c[3] = c[0] * c[3-0-1] + c[1] * c[3-1-1] + c[2] * c[3-2-1]$$
$$= c[0] * c[2] + c[1] * c[1] + c[2] * c[0]$$

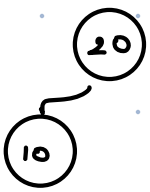
* Total no. of Unique BSTs

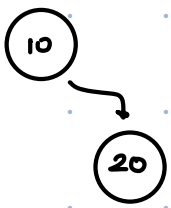
Q Given N, count total no. of unique BSTs that can be formed.

N = 0 ⇒ • ⇒ 1 = C_0

N = 1 {10} ⇒ 

⇒ 1 = C_1

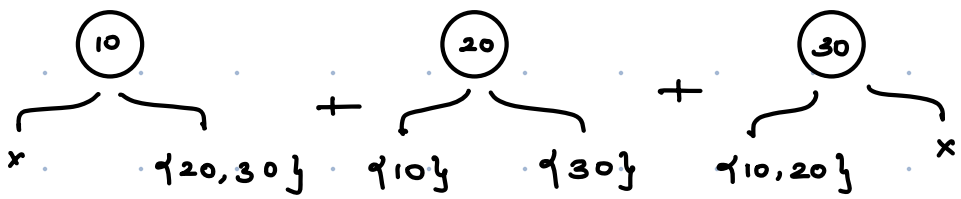
N = 2 {10, 20} ⇒ 

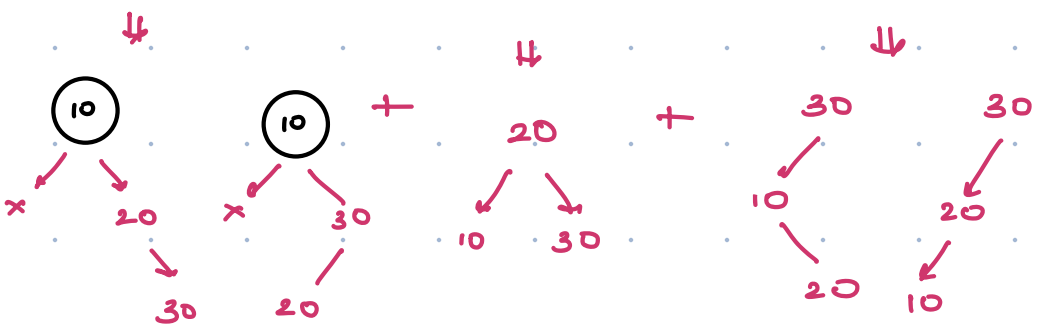


⇒ 2 = C_2

N = 3

{10, 20, 30}





Ans = 5 C_3

```
C[0] = 1

C[1] = 1

for (i = 2; i <= n; i++) {
    for (j = 0; j < i; j++) {
        C[i] += C[j] * C[i-j-1];
    }
}
```

TC : $O(N^2)$

SC : $O(N)$

* **Similar question**

→ Count no. of valleys

→ Count chords in a circle

→ No. of unique BSTs

→ Count no. of valid parenthesis.