DP-1

Question : Fibonacci series

	1st	2	3	4	5					
Fib =	0	1	1	2	3	5	8	13	21	34	55

Fib(n) = Fib(n-1) + Fib(n-2)

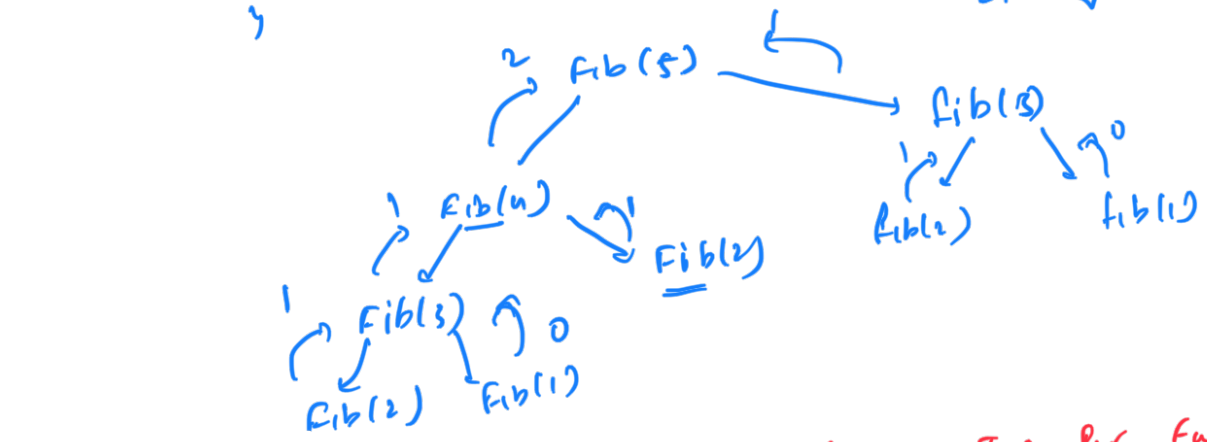
```
int fib(n) {
    if(n==1) return 0;
    if(n==2) return 1;
    return fib(n-1) + fib(n-2);
}
```

O(1) { ... }

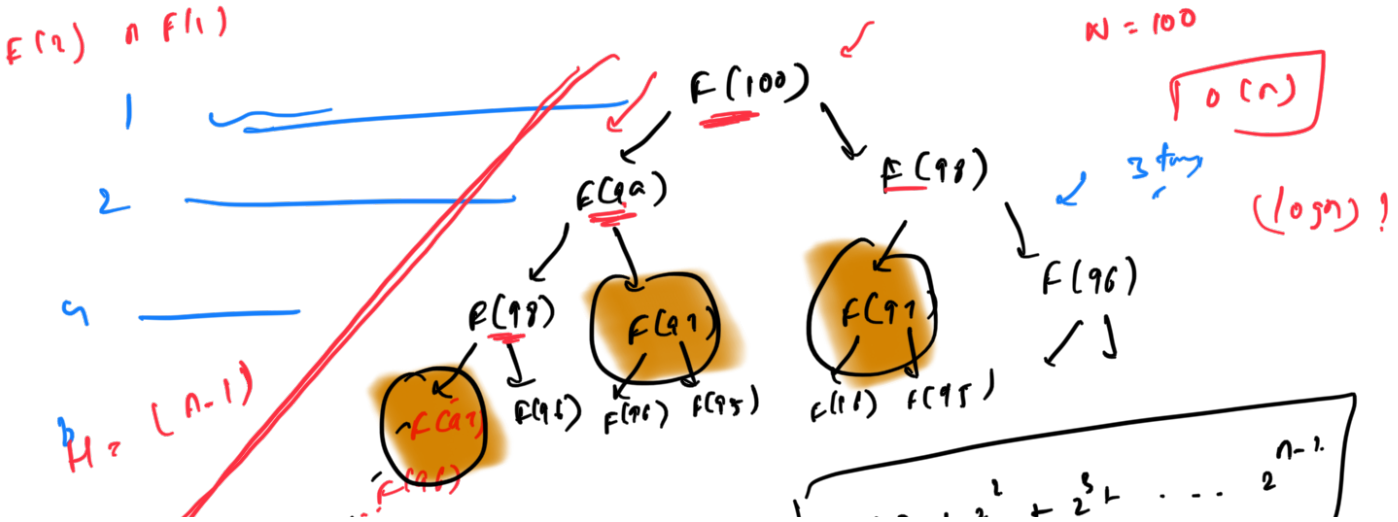
F(n) = F(n-1) + F(n-2)

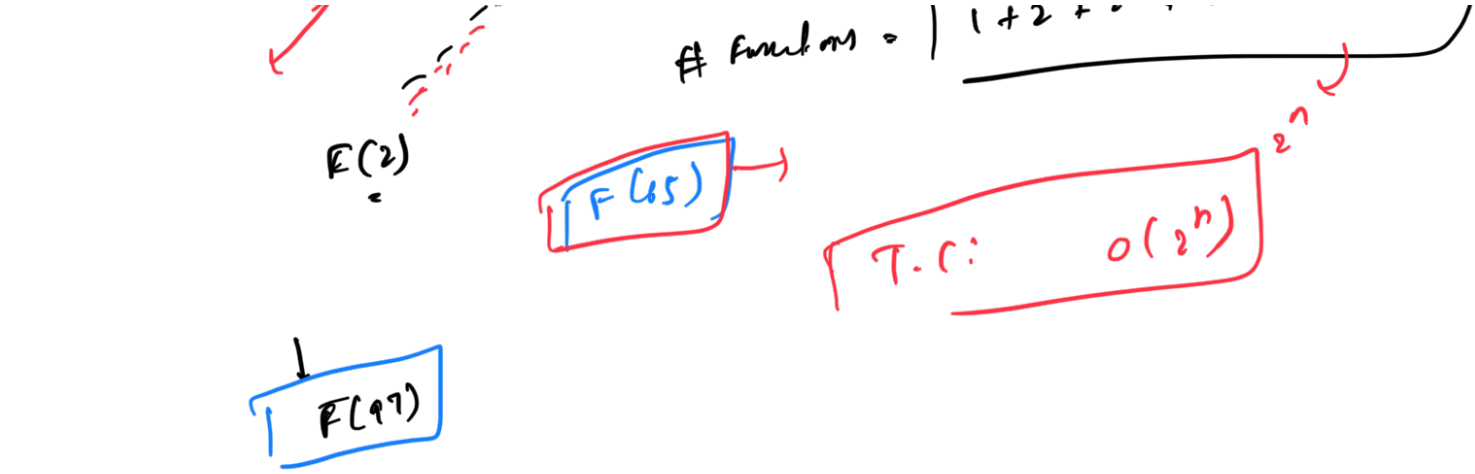
aT(n/b) + f(n)

2+1 = 3



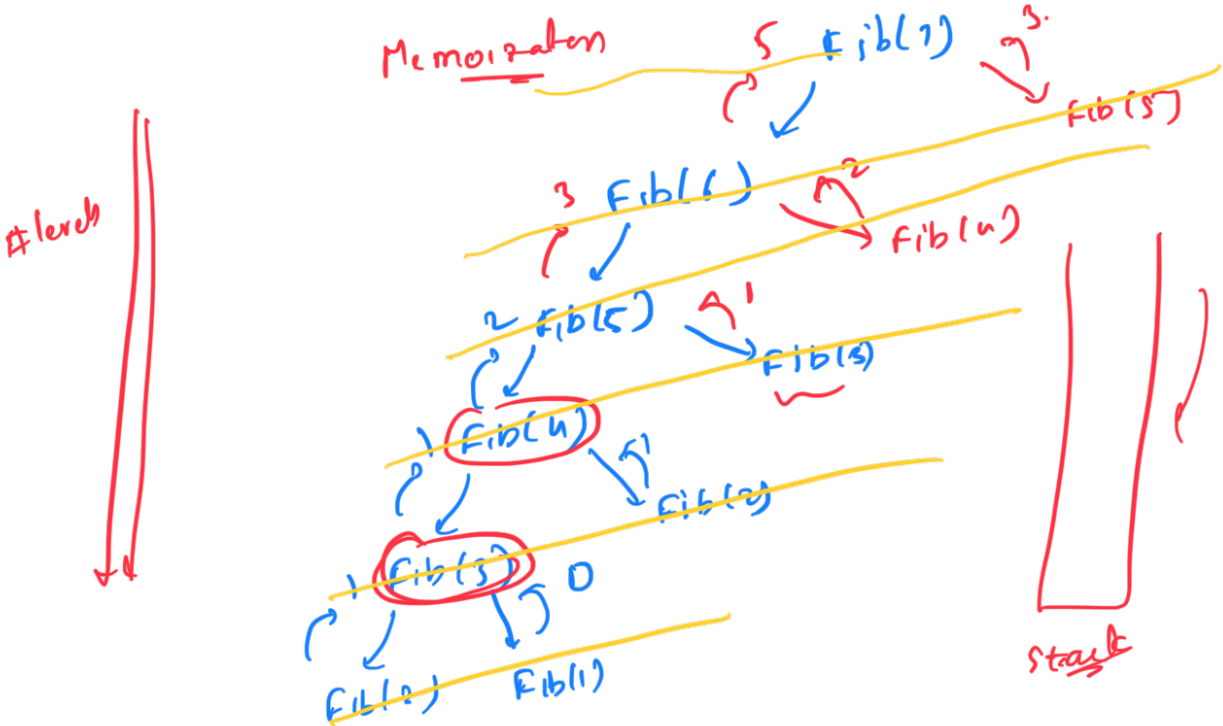
T.C : O(# Functions calls × T.C Per Function)





Memoization:

T.C from Exponential to Polynomial T.C



T-C: $O(n \times 2 \times 1) = O(n)$

T.C: O(n)

n → O(n)

N = 105

1s

HashMap / Array

dp[3] dp[2]

dp[N+1] = -1

```
fib(n) {
  if(n == 1) return 0;
  if(n == 2) return 1;
  if(dp[n] != -1) return dp[n];
  ans = fib(n-1) + fib(n-2);
  dp[n] = ans;
  return ans;
}
```

Same

dp[1] = 0; dp[2] = 1;

dp: | -1 | -1 | -1 | -1 | -1 | -1 |

Array (N) + Stack Space.

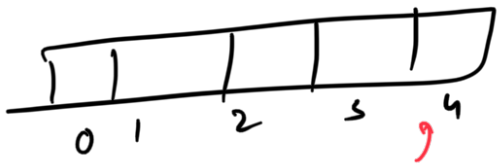
S.C: O(N) + O(N)

→ O(N)

```
fib(n) {
  if(dp[N] != -1) return dp[N];
  if(N == 1) {
    dp[1] = 0;
    return 0;
  }
  if(N == 2) {
    dp[2] = 1;
    return;
  }
}
```

N = 5

A[5]:



dp[5]

dp[N+1]

2 Types of DP:

1) Top - Down DP

Recursion + Memoization

2) Bottom - up DP

Tabulation

Cases

→ Build answers from base

Fib[1] = 0

Fib[2] = 1

Fib[3] = 1

Fib[4] = 2

Fib(1) → Fib(2) → Fib(3) . . . . Fib(n)

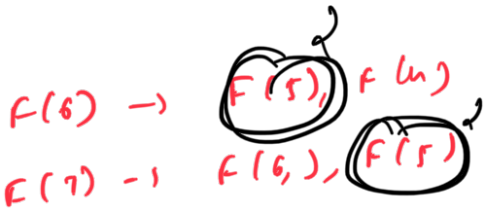
Steps

```
int Fib(n) {
    int Fib[N+1];
    Fib[1] = 0;
    Fib[2] = 1;
    for(i = 3; i <= n; i++) {
        Fib[i] = Fib[i-1] + Fib[i-2];
    }
    return Fib[n];
}
```

Fib(i)

T.C: O(n)

S.C: O(n)



F(7) → F(6), F(5)

F(8) → F(7), F(6)

F(9) → F(8), F(7)

F(n) => F(n-1), F(n-2)

N = 4

```
n = 0, y = 1
for(i = 3; i <= n; i++) {
    z = n + y
    n = y
```

T.C: O(n)

y = z.

}
return z; (y)

S.C : O(1)

i = 3
z = 1, x = 1, y = 1
i = 4
z = 2, x = 1, y = 2.

Differences :

Top Down	Bottom Up
Memoization	Tabulation
Easy to implement	Little trickier implementation as order to compute states should be known
Stack space	No stack space
Less control	More control [We optimised S.C to O(1)]

2 Conditions

1. **Optimal Substructure:** A given problems has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems.
   A bigger problem can be solved by solving smaller sub problems. (Recursion)
2. **Overlapping subproblems** : Repeating subproblems

```
int fact(N) {
  if(N==1) return 1;
  return N* fact(N-1);
}
```

Factorial(N) → N * Factorial(N). Optimal Substruct

F(8) → F(7) → F(6) → F(5) → F(4) → F(3) → F(2) Overlapping subproblems

Steps to solve a DP problem

1) Can we solve it using Recursion?
   Can we break a bigger problem into smaller sub problems?

2) Recursion Relation

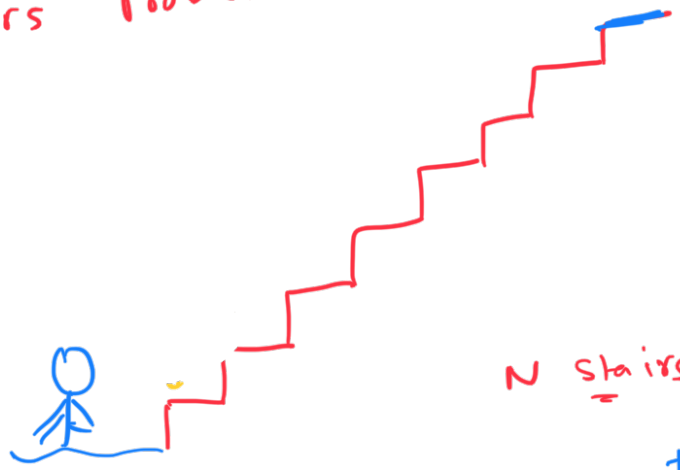
3) Base Case.

4) Overlapping Subproblems?
   a) No. of states (Size of DP array) (1D DP)
   b) dp[i] denotes what?
   c) DP expression
      dp[i] = dp[i-1] + dp[i-2];
   d) what is the final answer state?
      dp[N]

(1,5)

Stairs problem

Question:



N stairs

Find No. of ways to reach the N^{th} stair

Constraint: You can either take 1 or 2 steps

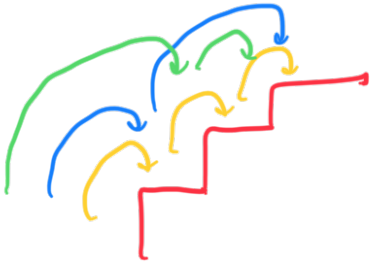
N = 1

=> 1

N = 2

{1,1} {2} => 2 ways

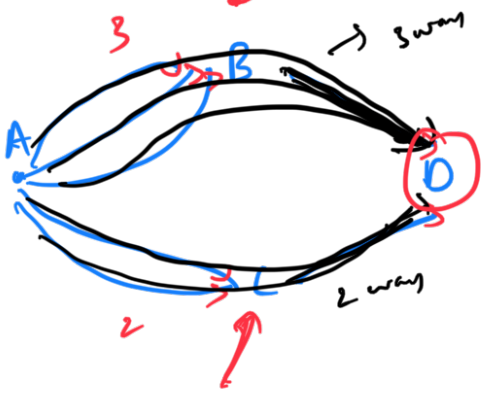
N = 3



{1,1,1}
{1,2}
{2,1}
} 3 ways

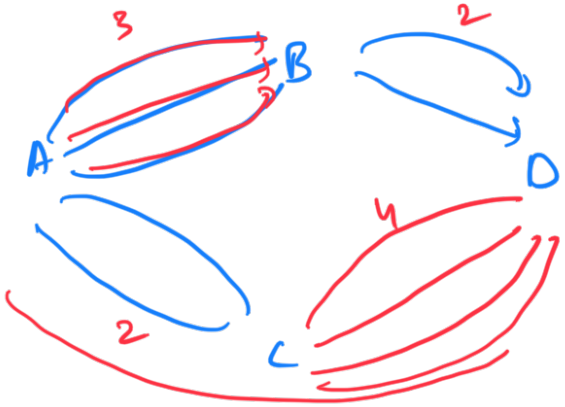
N = 5

8 ways



ways (A - D) = 3
ways (A - C) = 2

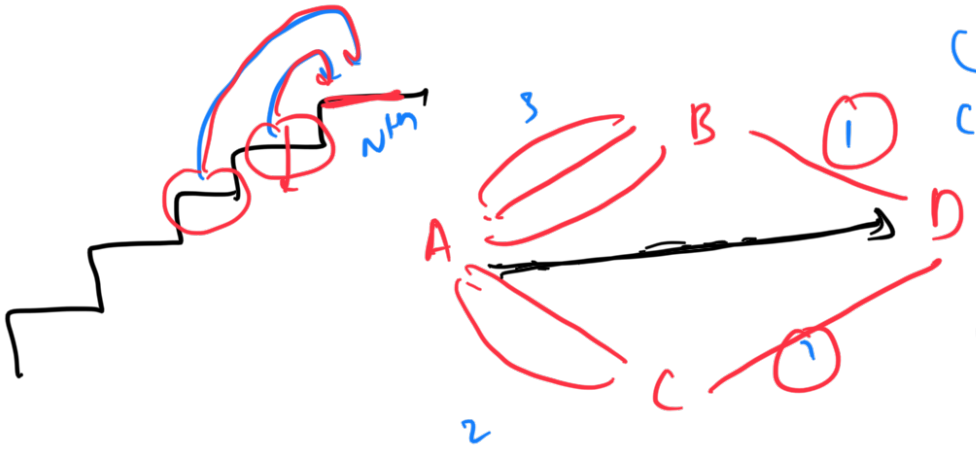
ways (A − D) = 3+2 way



n(A-B) x n(B-D) = 6
n(A-C) x n(C-D) = 8

Total n(A-D) 6+8

ways (N-1 + ways (N-2) x 2.



(A-D, B) : 3x1
(A-D, C) : 2x1

(A-D) = 3+2 = 5

3+2+1

ways (N) = ways (N-1) x 1 + ways (N-2) x 1

N = 1 => 1
N = 2 => 2.

Time Complexity

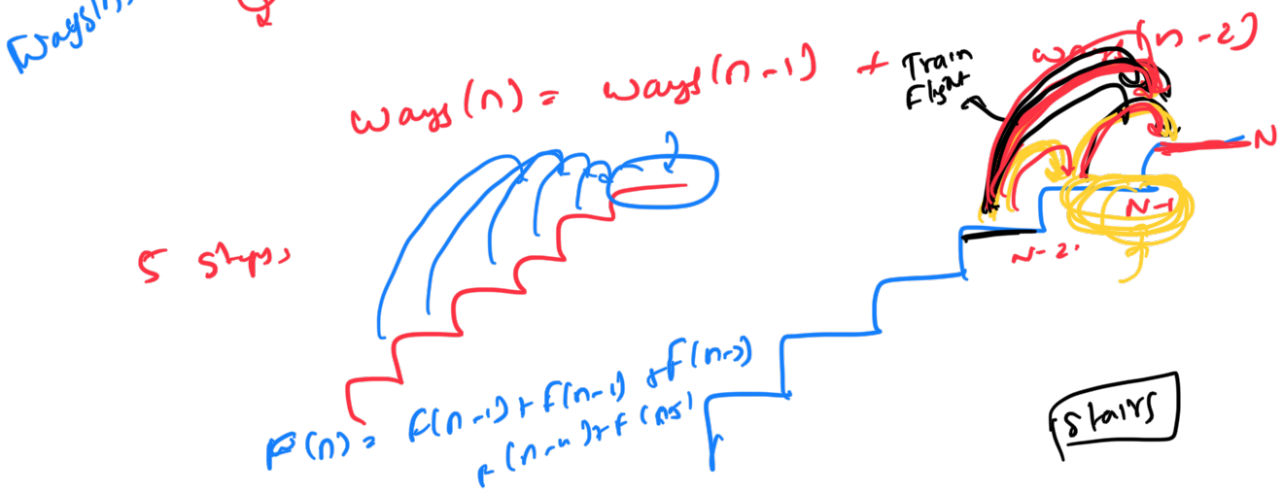
T.C : O(# States × T.C Per State)

n × 1

= O(n)

= O(n * 1)

Ways(n) = ways(n-1) + (2) ways(n-2)



Question: Lets Party

→ There are N people

→ Either party alone or party in Pair

→ Find no. of ways to party.

(A) N = 1 → (A) → 1 Way

(A,B) N = 2 → (A, B), (AB) → 2 ways

(A,B,C) N = 3 → (A, B, C), (AB, C), (AC, B), (BC, A) → 4 ways

N = 4 → 9 ways

25

5 + 6

N = 4 A B C D

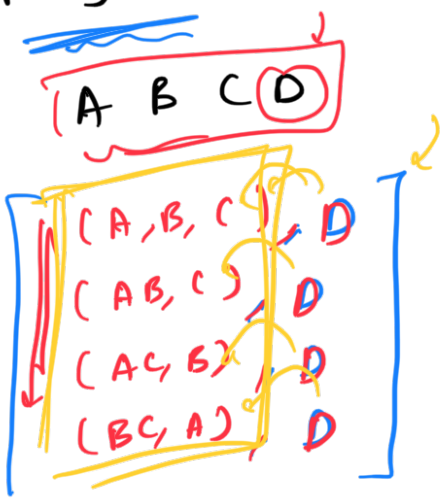
will ways(N-1) help?

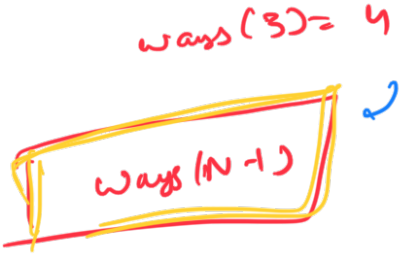
will ways(N-2) help?

Choices

Case 1) Party Alone.

N = 4

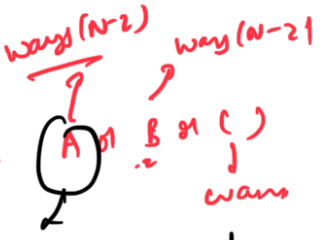


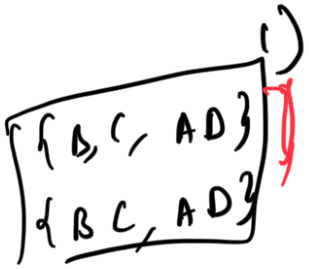


Case 2: Pair up

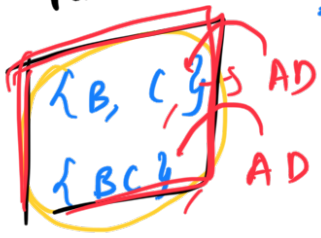
A B C D

He can pair up with (N-1) people





Pair with A



{AD}

2 ways

ways (N-2)

(N-1) × ways (N-2)

2) Pair with B

{A, C} BD

2 ways

{AC, BD}

{AC} B2

3) {A, B, CD}
{AB, CD}

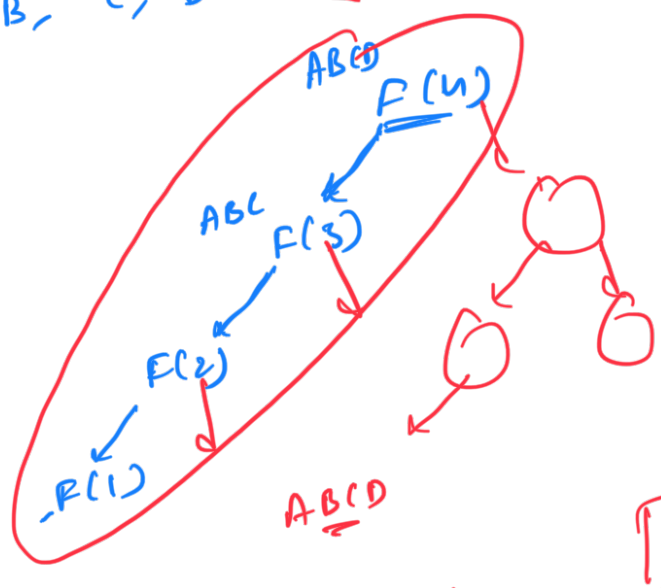
Pair with C
{A, B}, CD
{AB}, CD
} Ways

Case2: (n-1) ways (n-2)

ways(N) = ways(n-1) + (n-1) ways(n-2)

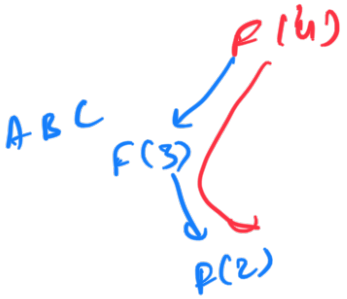
N == 1, ways(1) = 1
ways(2) = 2.

A, B, C, D



{A, B, C, D}
↓
Partitions Algo

{A, BC, D}



T.C: #States * T.C per state

N * O(1) ⇒ O(N)

S.C: O(N) → O(1) ! 2 variables

Question: Coin change

Input

Currencies = [5 10 25]

T = 37 → -1 → 25 → 25 + 5 = 30 → 2 coins

Target = 30

(5, 5, 5, 5, 5, 5)

(5, 5, 5, 5, 10)

(5, 5, 10, 10)

(10, 10, 10)

(5, 25)

2 coins

Ex2 3 4 5 Target = 2.

Return -1

Ex 1 10 18

Target = 20

{18, 1, 1} → 3 coins

{10, 10} → 2 coins

T = 6 {2+4} {3+3}

Approach 1:

A = 2 3 4

T = 6

Choices:

coin 2

Case 1 : Choose coin 2

Coins = 1 + minCoins(6-2)

Case 2 : Choose coin 3

= 1 + minCoins(6-3)

Case 3 : Choose coin 4

= 1 + minCoins(6-4)

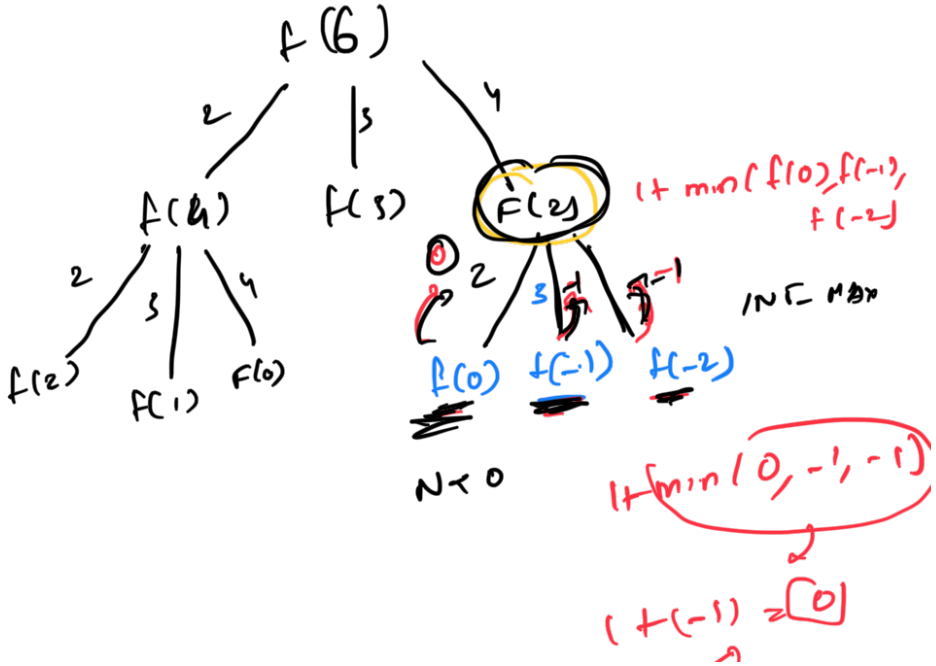
$$minCoins(N) = \min \begin{cases} 1 + minCoins(6-2) \\ 1 + minCoins(6-3) \\ 1 + minCoins(6-4) \end{cases}$$

minCoins(N) = 1 + min(minCoins(N - A[i])) ∀ i

Finds min no of coins to make a target N.

A: 

A = [2, 3, 4]



min(0, -1, -1) = 0

```
dp[N] = {-1};
int minCoins ( int N ) {
    if( N == 0) return 0;
    if( N < 0) return INT_MAX;
    if (dp[N] != -1) return dp[N];
    ans = INT_MAX;
    for(i = 0; i < A.size(); i++) {
        ans = min(ans, 1 + minCoins(N - A[i]))
    }
    dp[N] = ans;
    return ans;
}
```

O(size(A))

INF_MAX

return minCoins(N)

Target

A = 3 4 5

N = 2

dp[N] = INT_MAX


```
if( dp[N] == INT_MAX)
    return -1;
else dp[N]
```

T.C: (# states * T.C per state)

T.C: O(N x size(A)):

O(N)

S.C: [O(

minCoins(7, 3) = 2 {5+1+1}
minCoins(7, 2) = 3 {2+2+3}

Approach 2:

A = [2, 3, 4]
0 1 2

choices

Case 1) Not used at all →
Case 2) Used once or more times {1, 2, 5, 4, 5 ..... }

minCoins(T, N) = Finds Min No. of Coins to make a target T from currIdx of length N

A = [2, 3, 4]
0 1 2

T-

F(7, 3)

A = [2, 3, 4]
0 1 2

T = 7
N = 3

Case 1: Find Min Coins of Remaining array
minCoins(T, N-1)

Case 2:
1 + minCoins(T - A[n-1], N)

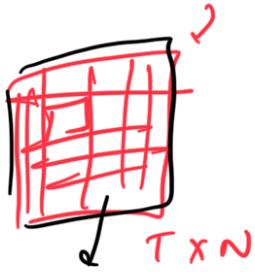
dp[target]

min(minCoins(T, N-1),

minCoins(T, N) = 1 + minCoins(T - A[n-1], N);

T.C: # states × T.C per state.

(2D Array)



T.C: O(T × N × 1)

T.C: O(T × N)

S.C: O(T × N)

Question: Return number of minimum squares needed to form N

N = 6 ⇒

$1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$ → 6

$1^2 + 1^2 + 2^2$ → 3

$(\sqrt{6})^2 = 6$

$y^2 =$

[1^2, 2^2]

T = 6

N = 17 ⇒

$1^2 + 4^2$

coins to make T = 17

Find min

using Coins = $[1^2, 2^2, 3^2, 4^2]$

$\sqrt{N}$

N = 4

A B C D

ways(4) = ways(3) + 3 ways(2)

A B C

ways(3) = ways(2) + 2 ways(1)

A B

ways(2) = ways(1) + 1 ways(0)

A

ways(1)