Recursion

Recursive self ←

Smaller values - computed

↓

larger values - reuse

Ways to decode.
Rec | DP

$f(n) = g(f(n-1))$

Length of a linked list.

head → length(head) = { 0 ; if head is null
1 + length(head.next) ; if head is non-null.

f → length.

f(n) → length(head)
↓
n nodes.

f(n-1) = length(head.next)
↓
n-1 nodes

length(head)

$f(n) = \begin{cases} 1 + \text{length}(\text{head.next}) \\ 1 + f(n-1) \\ g(f(n-1)) \end{cases}$; $n > 0$

```
int length(LinkedListNode node) {
  if (node is null)
      return 0
  return 1 + length(node.next)
}
```

```
LinkedListNode {
  int val
  LLN next
}
```

For. eg. 1 → 2 → 3 → null

returned value ← 3 ← length(1 → non-null) → 1 2.

3

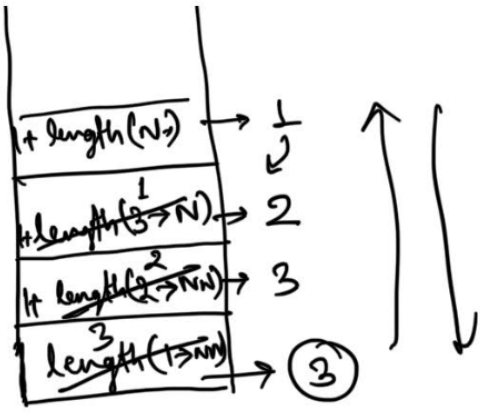
1 + length(2 → NN) 2

↓ 2

1 + length(3 → N) 1

↓ 1

1 + length(N) 0

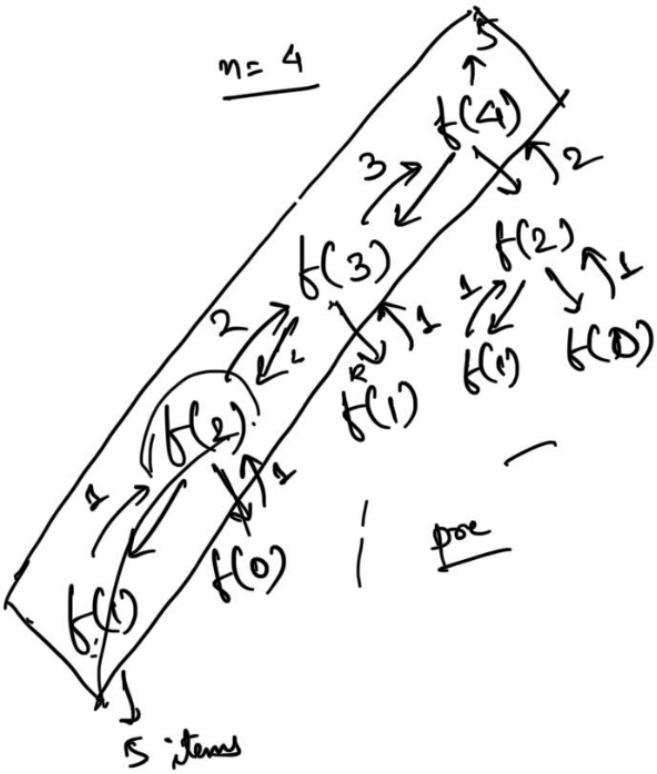


1+length(N) 0

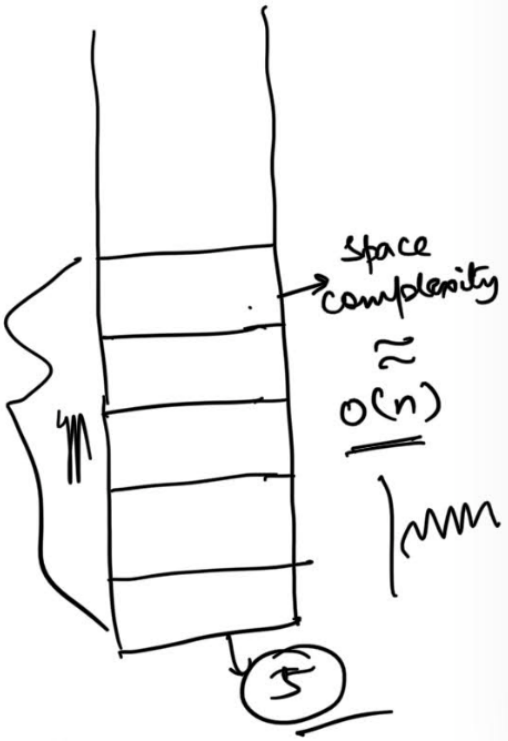
Fibonacci $f(n) = f(n-1) + f(n-2)$; if $n == 0, 1 \rightarrow f(n) = 1$

1 1 2 3 5 8 13 21 . . .

n=4



height of tree



Length of String.

LL vs String.

1 → 2 → 3 → null

node

"abcde"

index

lenOfString(S, 0)

```
int lengthOfString(String s) {
    if (s == "")
        return 0
    return 1+ s.substring(1);
}
```

beginIndex →

Very inefficient.

Sum of elements in an array.

1	2	3	1	1	4

sum of first x items of the array.

Sum(arr, a.length)

```
int Sum(int[] arr, int items) {
    if (items == 0)
        return 0;
    return arr[items-1] + Sum(arr, items-1);
}
```

Sum(6)
12
4 + Sum(5)
8
1 + Sum(4)
7
1 + Sum(3)
6
3 + Sum(2)
3
2 + Sum(1)
1
1 + Sum(0) 0

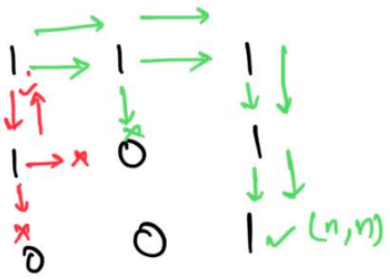
Tree

12

1D array → linear data structure.
calling recursive fn only once
↓
Simple tree.

2D array / tree / graph
↓
backtracking

Rat in a maze



1 — traversable
0 — non-trav.

(1,1) → (n,n)

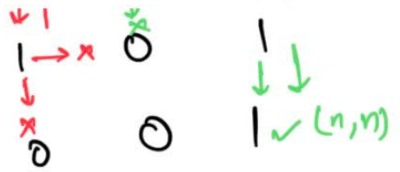
(x,y) → (x, y+1) (0,1)
2 dir^n
↓ (1,0)
(x+1, y)

```
dirs = [[1,0], [0,1]]
rec(x,y)
for(int[] dir : dirs) {
    nx = x + dir[0] ? 0 ;
    ny = y + dir[1] ? 0 ;
}
```

(x+1, y) (x, y+1)

4 dir^n
(-1,0)
(0,-1) ← → (0,1)
(1,0)

dirs = [[1,0], [0,1], [-1,0], [0,-1]]



(1,1) → (n,n)

2 dir^n

(x,y) →(0,1)→ (x,y+1)

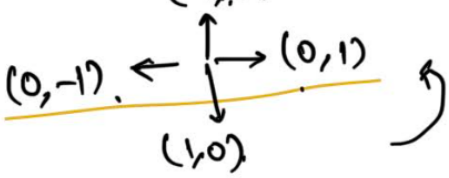
(x,y) ↓(1,0) (x+1,y)

dirs = [[1,0], [0,1]] ←

```
rec(nx,ny)
for(int[] dir : dirs) {
    nx = x + dir[0]     1  0
    ny = y + dir[1]     0  1
}
```

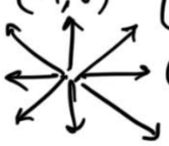
(x+1, y) (x, y+1)

4 dir^n



dirs = [[1,0],[0,1],[-1,0],[0,-1]]

8 dir^n.



dirs = [. . . .].

rec(grid, 1, 1)

int[][] dirs.

```
List<int[]> rec(int[][] grid, int x, int y)
invalid → if (x,y out of bounds or grid[x][y]==0)
              return []
base → if (x==n && y==n) ←
              return [[n,n]]
rec^n → for (int[] dir : dirs) {
            nx, ny = x+dir[0], y+dir[1]
            List<int[]> path = rec(grid, nx, ny)
            if (path is not empty) {
                path.add(0, [x,y])
                return path
            }
        }
        return [ ]
}
```

[(1,1),(1,2),(1,3),(2,3),(3,3)]

path to be considered only after reaching (n,n)

[(1,1) (1,2), (1,3), (2,3), (3,3)]

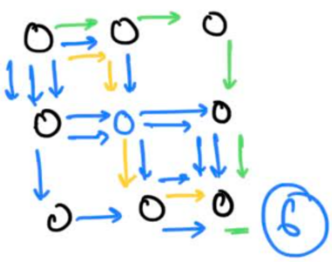
int[][] emptygrid

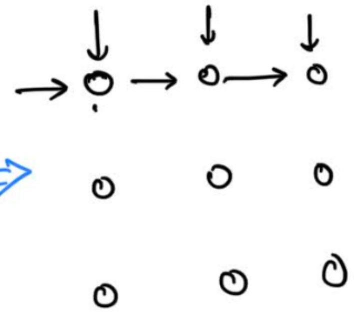
2 loops (i,j) EG[i][j] = 1

2 limits (i, j) G[i][j] = 1

}

Unique Paths in a Grid.





curr → new locⁿˢ   prev locⁿˢ → curr locⁿ.

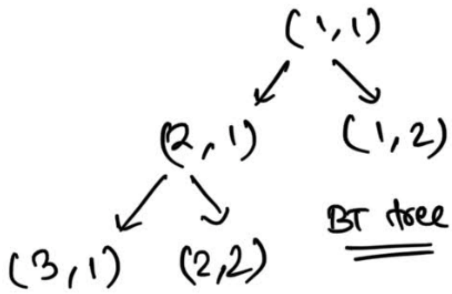
```
int rec(     x,     y )
  → if (x == m && y == n)
        return 1

  → return rec(x+1, y) + rec(x, y+1)
}
```

rec(1,1)

```
int[][] dp (-1)

⇒ int rec ( x, y)
    if(x == m && y == n)
        return 1
    if(dp[x][y] == -1) {
        dp[x][y] = rec(x+1, y)
                   +
                   rec(x, y+1)
    }
    return dp[x][y]
}
```



result from recursive steps.
↓
directly returned.

results are memoized before returning so that they can be reused later avoiding resolve of overlapping subproblems.

Ways to decode.

11232

A = 1
B = 2
⋮
Z = 26

int[] dp (size+1)

avoiding recomputation of overlapping subproblems.

Ways to decode.

11232

A = 1
B = 2
⋮
Z = 26

int [] dp (size+1).

int w2d(int characters)

dp[0] = w2d(0).

dp[1] = w2d(1). → "1" → A → w2d(0).

dp[0] = 1 ← Base case.

dp[size] = w2d(size).

w2d(2) → "11" → AA, K; 2 → w2d(0)

220

22 + "0" ✗

2 + "20" → T n-2

31 → w2d(2)

w2d(2) = (w2d(1) "0", w2d(0) "31")
= w2d(1) + w2d(0)

1-9, 10-26

validate if this string can be mapped to an alphabet.

w2d(n) = w2d(n-1) + (*) w2d(n-2)

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

Max Sum w/o adjacent elements.

0 | 1 2 3 4 | 2 3 4 5 | Max: 4

1 2 5 7
2 3 6 8
2 3 8

dp(i)[j] = max(dp(i-2)[0], dp[i-2][1]) + grid[i][j]

all options → max + curr val.