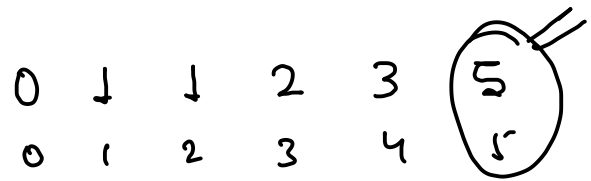
Dynamic Programming -

Sacrifice with space
and
Save time } massive time

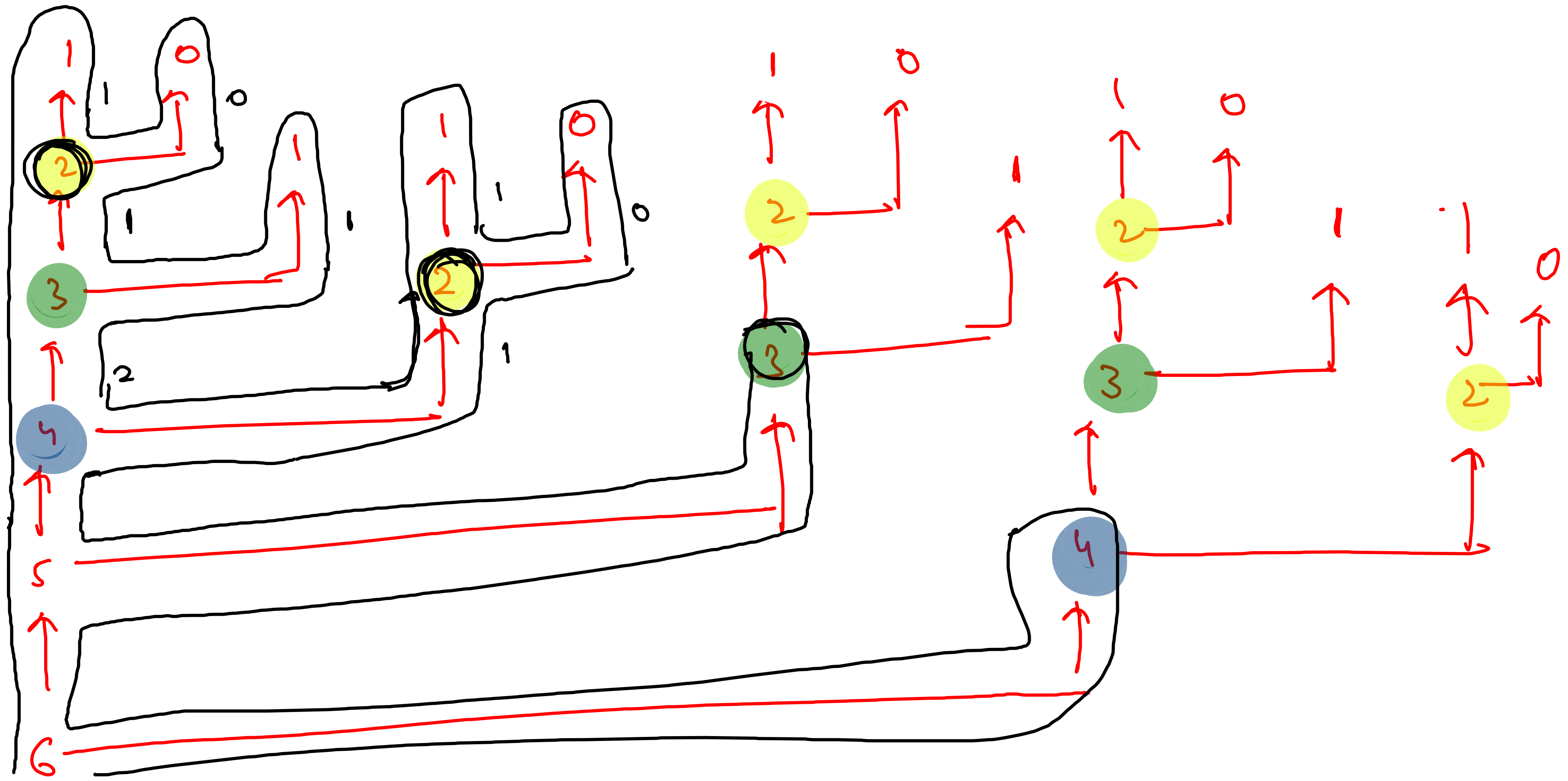
↳ Repitited problems.




```
public static int fib_rec(int n) {
    if(n == 0 || n == 1) return n;

    int fnm1 = fib_rec(n - 1);
    int fnm2 = fib_rec(n - 2);
    int fibn = fnm1 + fnm2;

    return fibn;
}
```


fib(5) = 5

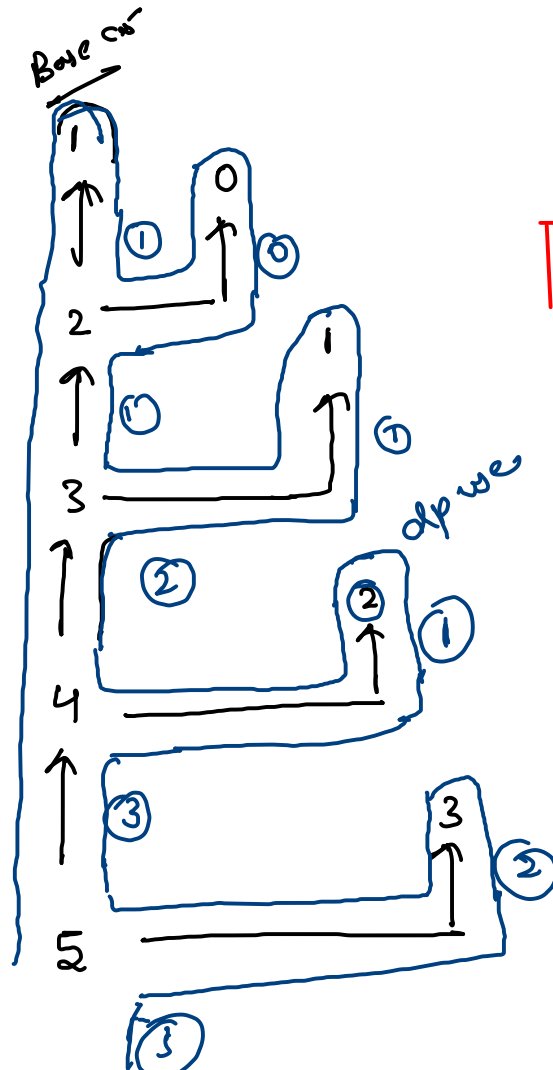
n = 5

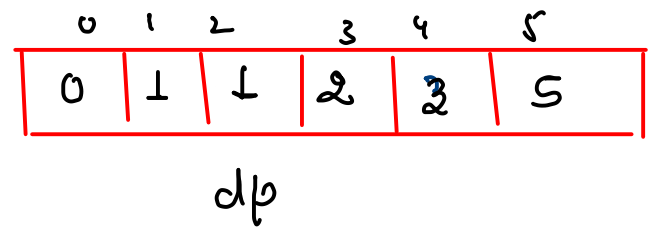

6
5
4
3
2
1
0

```java
public static int fib_memo(int n, int[] dp) {
    if(n == 0 || n == 1) {
        dp[n] = n;
        return dp[n];
    }
    // if problem is already solved, then return the solution
    if(dp[n] != 0) {
        return dp[n];
    }

    int fnm1 = fib_memo(n - 1, dp);
    int fnm2 = fib_memo(n - 2, dp);
    int fibn = fnm1 + fnm2;
    // solve and store the problem in dp
    dp[n] = fibn;
    return fibn;
}
```

n = 5

dp size = n+1





Tabulation:

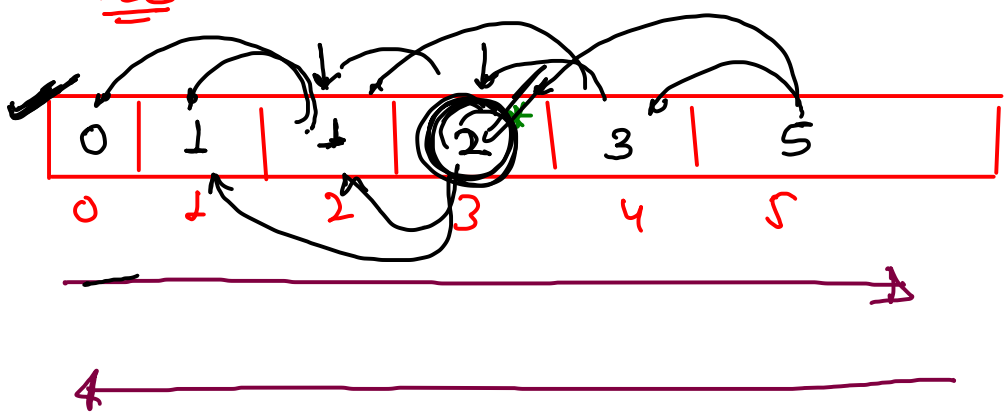
1. Make Storage
2. Assign meaning to cell
3. Identify traversal of problem.
4. set prerequisite in 'dp'. → Smallest to largest problem

dp[8] = answer of fib(n).

dp[8] = it have sum of dp[2] + dp[1]

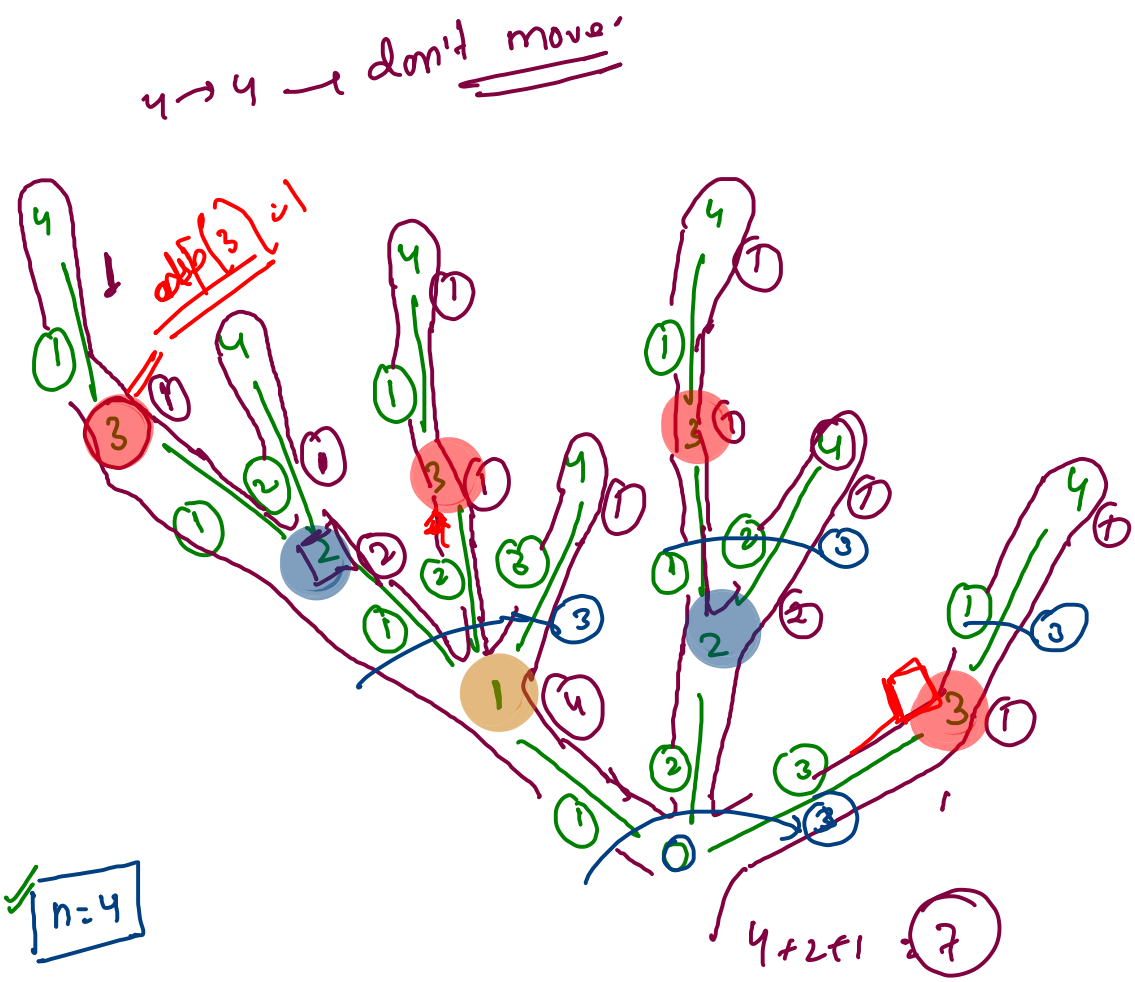
fib tabulation:-

n = 5

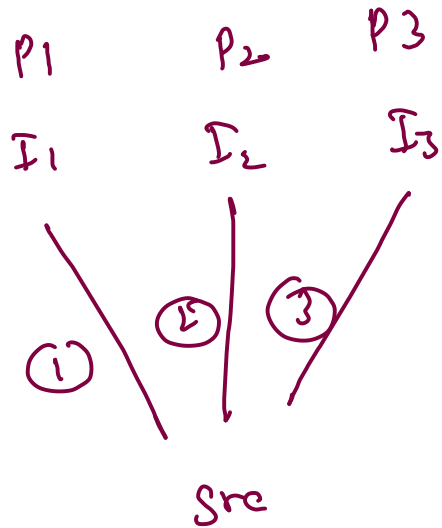


dp[n]

Climb Stair paths

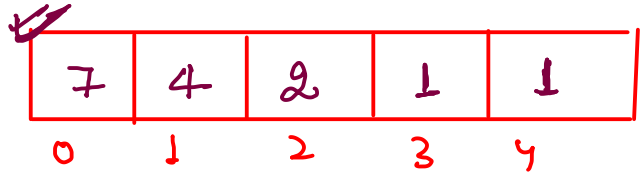


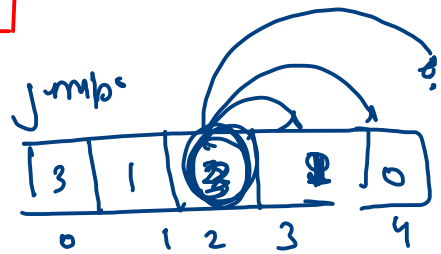
as.

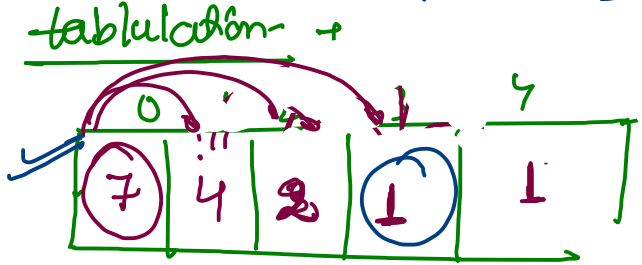


Src to dst = p1 + p2 + p3

```java
public static int climbStair_memo(int i, int n, int[] dp) {
    if(i == n) return dp[i] = 1;

    if(dp[i] != 0) return dp[i];

    int count = 0;
    for(int jump = 1; jump <= 3 && i + jump <= n; jump++) {
        count += climbStair_memo(i + jump, n, dp);
    }

    return dp[i] = count;
}
```

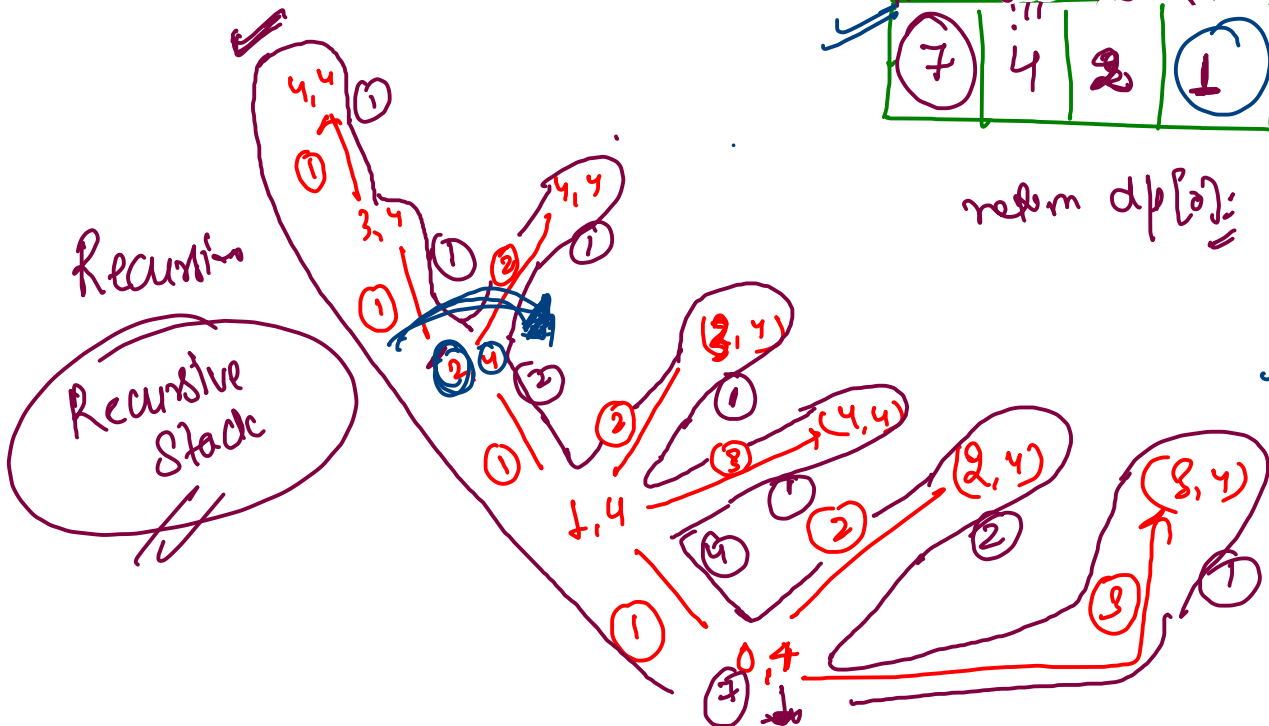






return dp[0]; Smallest problem.

dp[n] = 1;



n=7

dp - size = n+1 = 8

Meaning dp[i] = No. of path from i to n

44	24	13	7	4	2	1	1
0	1	2	3	4	5	6	7

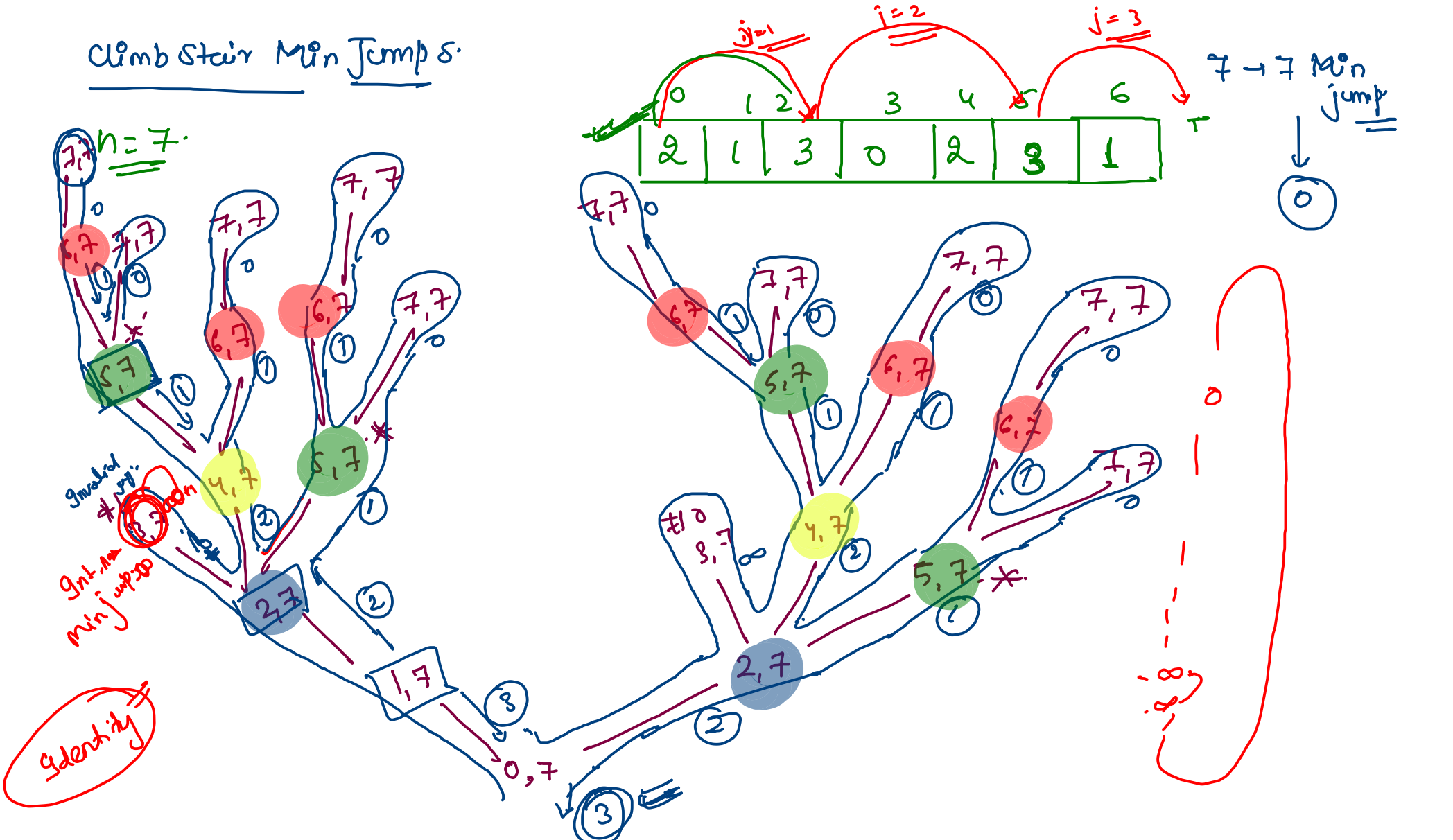
Smallest proble dp[n]

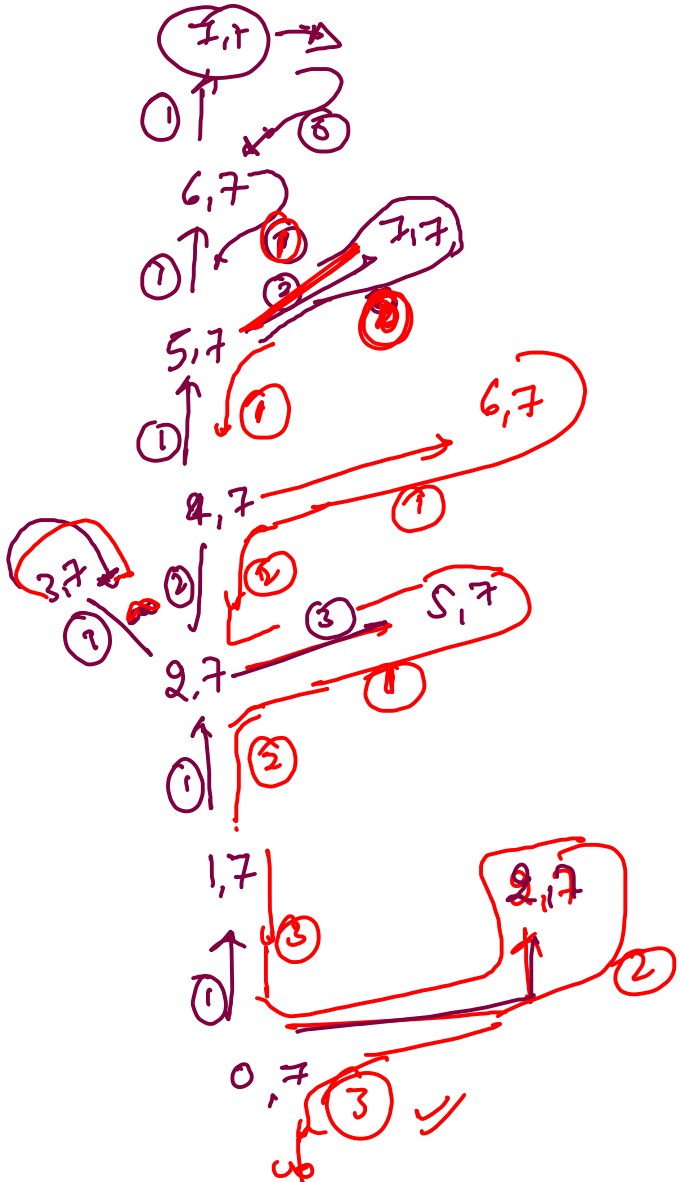
dp[n] = 1;

2	4	2	0	0
0	1	2	3	4

3	2	1	0	1
0	1	2	3	4

Climb Stair Min Jumps

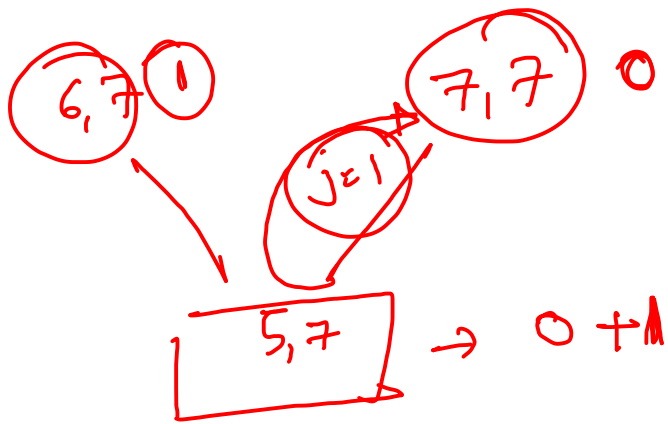




n = 7

2	1	3	0	2	3	1
0	1	2	3	4	5	6

3	3	2	∞	2	1	1	0
•	1	2	3	4	5	6	7

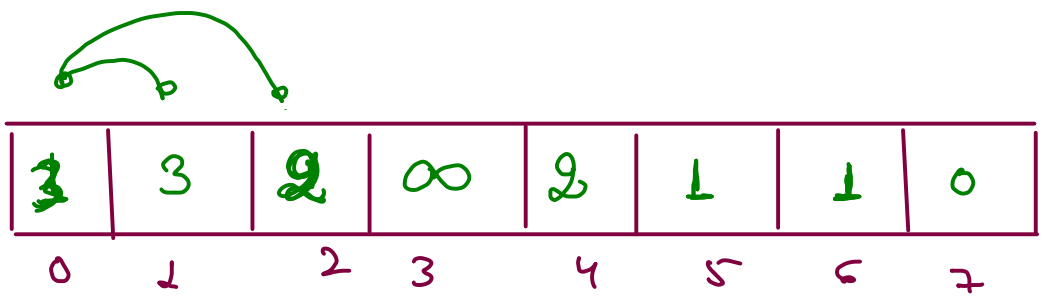


Tabulation-

2	1	3	0	2	3	1
0	1	2	3	4	5	6

↑ ↑

* Meaning dp[i]:- dp[i] have min jumps, in which we can reach at destination from i.



dp[0] = 3