Consecutive 1's not allowed

Medium Accuracy: 48.5% Submissions: 17061 Points: 4

Given a positive integer **N**, count all possible distinct binary strings of length **N** such that there are **no consecutive 1's**. Output your answer **modulo 10^9 + 7**.

Count Valid Binary Strings

↓

0, 1
R, B
{, }

} Binary can have any such variation.

① → "0", "1" Ans = 2

② → "00", "01", "10", "~~11~~" Ans = 3

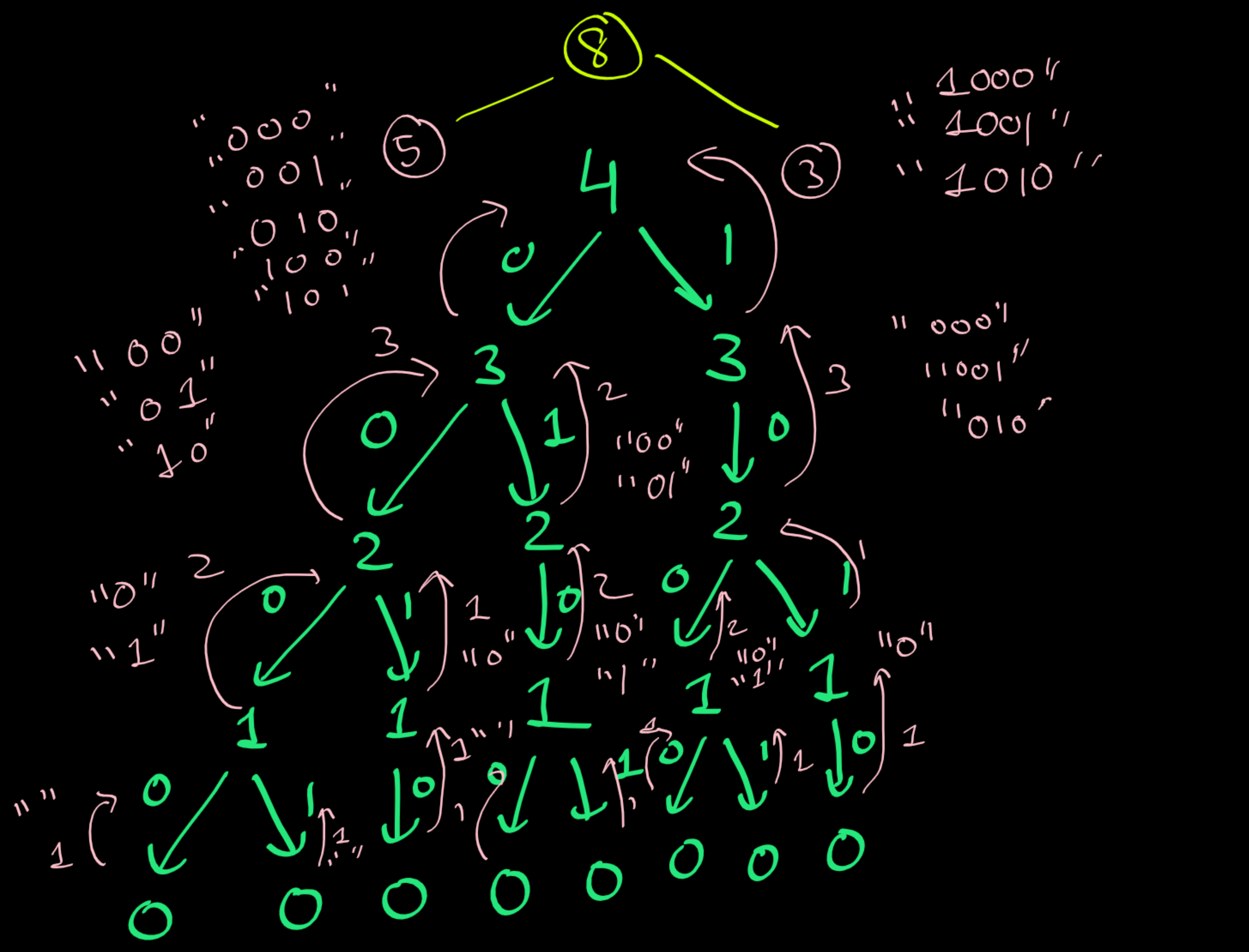
③ → "000", "001", "010", "~~011~~", "100", "101", "~~110~~", "~~111~~"

Ans = 5

Recursion

✓ ↙ Get (At return time)

↓ ✗ Print (when going into rec)



DP works only with get Method most of the times.

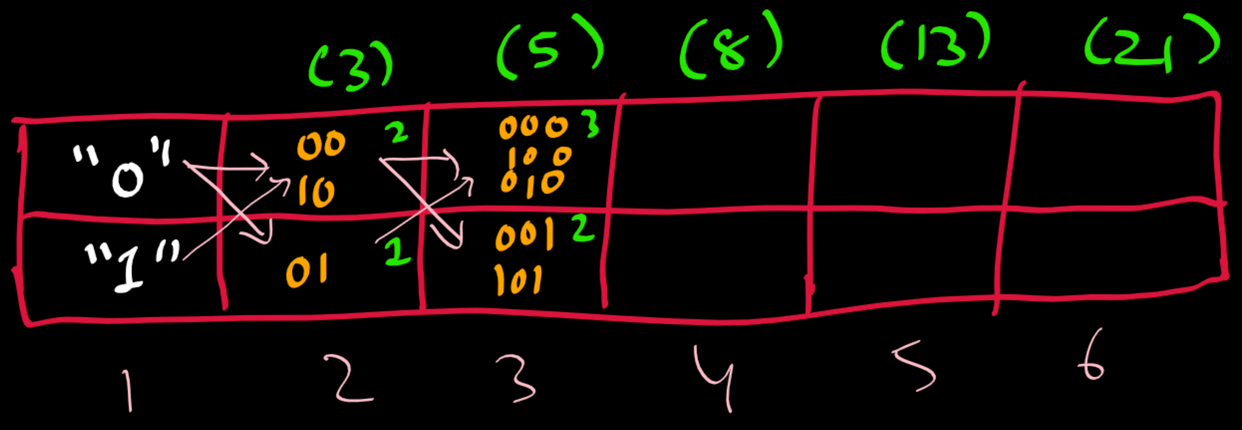
```
long memo(int noOfDigits,int prevDigit) {

    if(noOfDigits == 0) return 1; //empty string

    long appending0 = memo(noOfDigits -1,0);
    long appending1 = (prevDigit == 0) ? memo(noOfDigits - 1,1) : 0;

    return (appending0 + appending1) % mod;
}
```

} Recursive Code

```
class Solution {

    int mod = 1000000007;

    long memo(int noOfDigits,int prevDigit,long [][] dp) {

        if(noOfDigits == 0) return 1; //empty string
        if(dp[noOfDigits][prevDigit] != 0)
            return dp[noOfDigits][prevDigit];

        long appending0 = memo(noOfDigits -1,0,dp);
        long appending1 = (prevDigit == 0) ? memo(noOfDigits - 1,1,dp) : 0l;
        dp[noOfDigits][prevDigit] =  (appending0 + appending1) % mod;
        return (appending0 + appending1) % mod;
    }

    long countStrings(int n) {
        // code here
        long[][] dp = new long[n + 1][2];
        return memo(n,0,dp);
    }
```

} Memoization Code

Tabulation for No of Valid Strings



⟹ fibonacci Series

So, we can even directly write fibonacci code here.

Length: 1 2 3 4 5 6

2 3 5 8 13 21

0 1 1 2 3 5 8 13 21

0th 1st 2nd 3rd 4th 5th 6th 7th 8th

No of N length valid strings = $(N+2)^{th}$ fib No

Tabulation Code

```
class Solution {
    int m = 1000000007;
    long countStrings(int n) {
        // code here
        if(n == 0) return 1;
        int N = n + 2;

        long[] dp = new long[N+1];
        dp[0] = 0l;
        dp[1] = 1l;

        for(int i=2;i<dp.length;i++) {
            dp[i] = (dp[i-1] + dp[i-2]) % m;
        }

        return dp[N];
    }
}
```

91. Decode Ways

Medium 6734 3800 Add to List Share

A message containing letters from `A-Z` can be **encoded** into numbers using the following mapping:

```
'A' -> "1"
'B' -> "2"
...
'Z' -> "26"
```

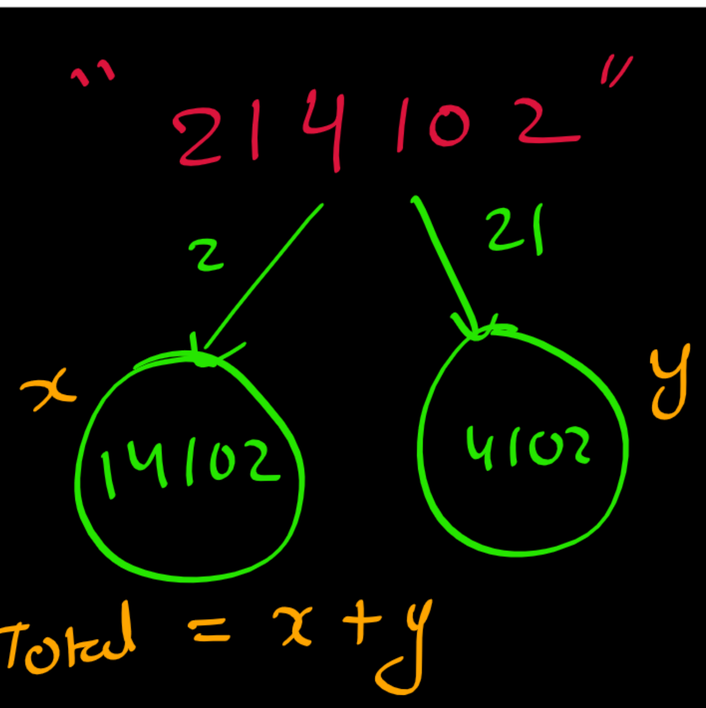
To **decode** an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, `"11106"` can be mapped into:

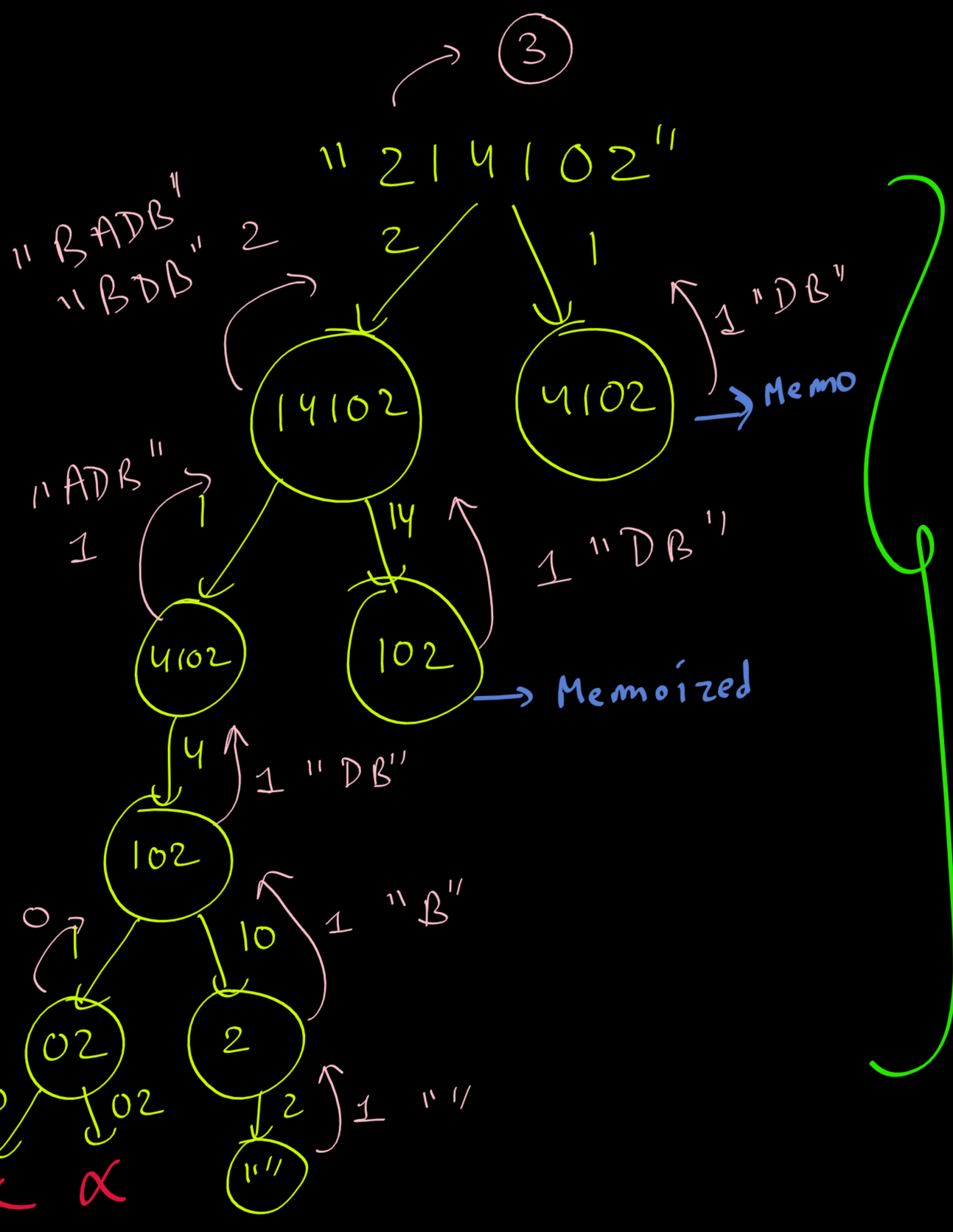
- `"AAJF"` with the grouping `(1 1 10 6)`
- `"KJF"` with the grouping `(11 10 6)`

Note that the grouping `(1 11 06)` is invalid because `"06"` cannot be mapped into `'F'` since `"6"` is different from `"06"`.

Given a string `s` containing only digits, return *the* ***number*** *of ways to* ***decode*** *it.*

The test cases are generated so that the answer fits in a **32-bit** integer.





Memoized Recursion Tree

DP table can be made by taking indexes as the length of the input string.

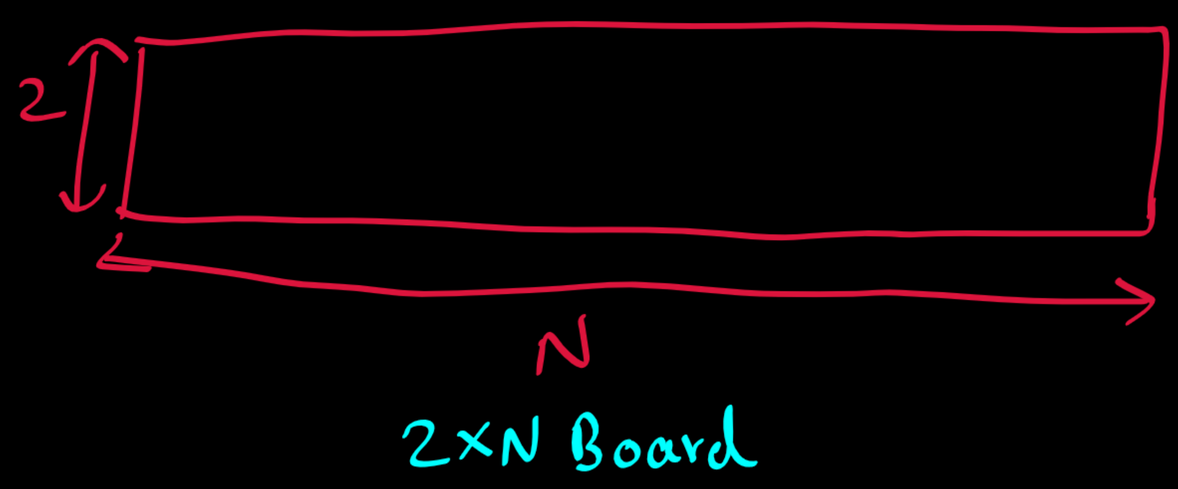
Count Encodings

A 1
B 2
C 3
D 4
E 5
F 6
G 7
H 8
I 9
J 10
K 11
L 12
M 13
N 14
O 15
P 16
Q 17
R 18
S 19
T 20
U 21
V 22
W 23
X 24
Y 25
Z 26

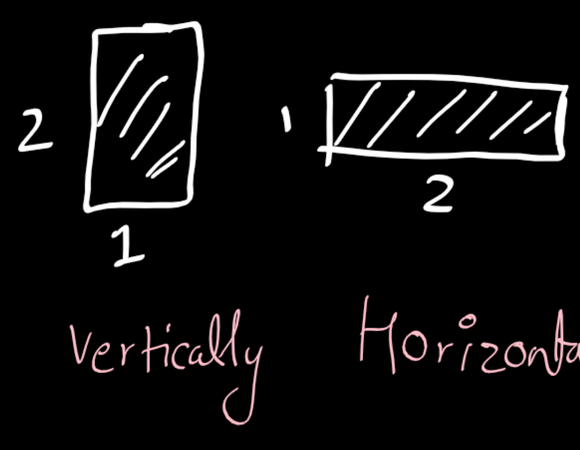
```java
public int memo(String s, int idx, int[] dp) {
    if(idx == s.length()) return 1;

    if(dp[idx] != -1) return dp[idx];

    int ans1 = 0, ans2 = 0;
    if(s.charAt(idx) != '0') {
        ans1 = memo(s,idx + 1,dp);
    }

    if(idx + 1 < s.length() && s.charAt(idx) != '0') { //length of s should be atleast 2
                                                       //& first char should not be 0
        int code = (s.charAt(idx) -'0') * 10 + (s.charAt(idx+1) -'0');
        if(code >= 10 && code <= 26) {
            ans2 = memo(s,idx + 2,dp);
        }
    }

    return dp[idx] = ans1 + ans2;
}

public int numDecodings(String s) {
    int[] dp = new int[s.length() + 1];
    Arrays.fill(dp,-1);
    return memo(s,0,dp);
}
```

Memoization Code for Count Encodings

Tiling Problem - 1

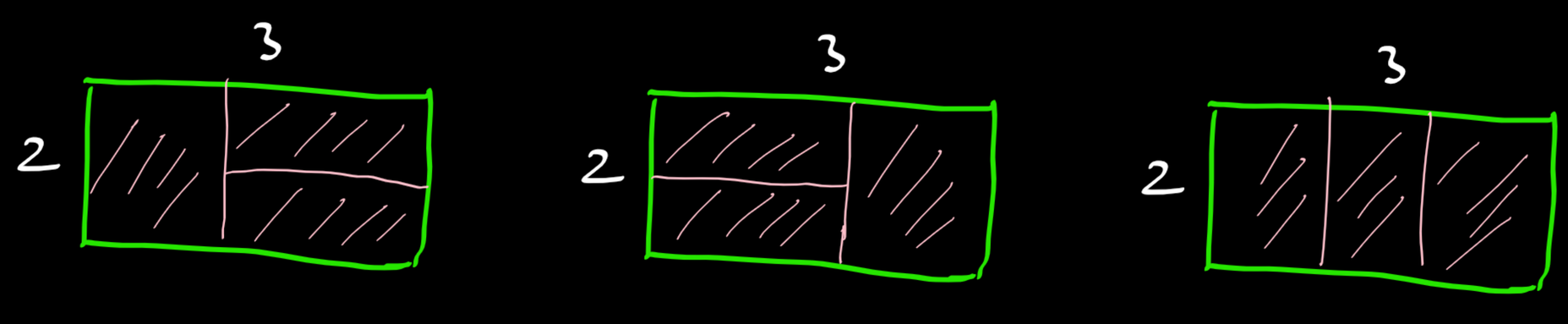


2×N Board

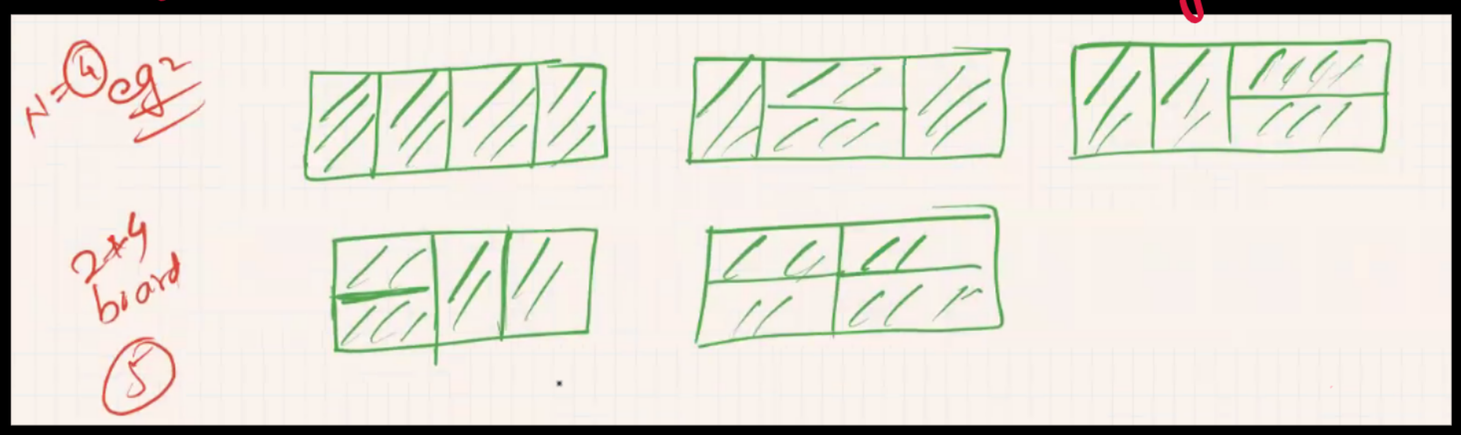


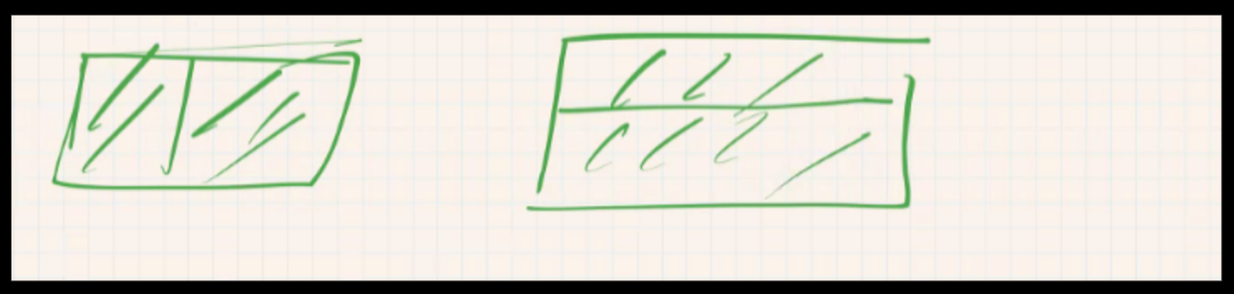
Vertically Horizontally

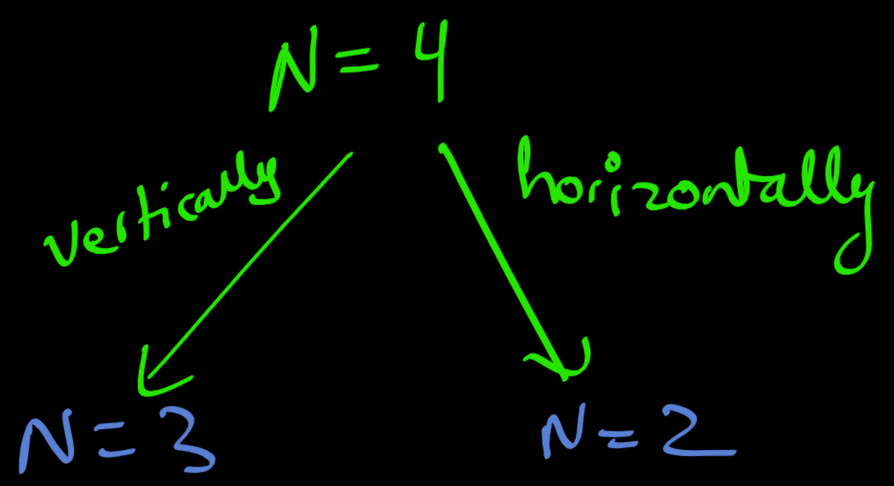
for N = 3



3 ways to file 2*N board using 2*1 fiks



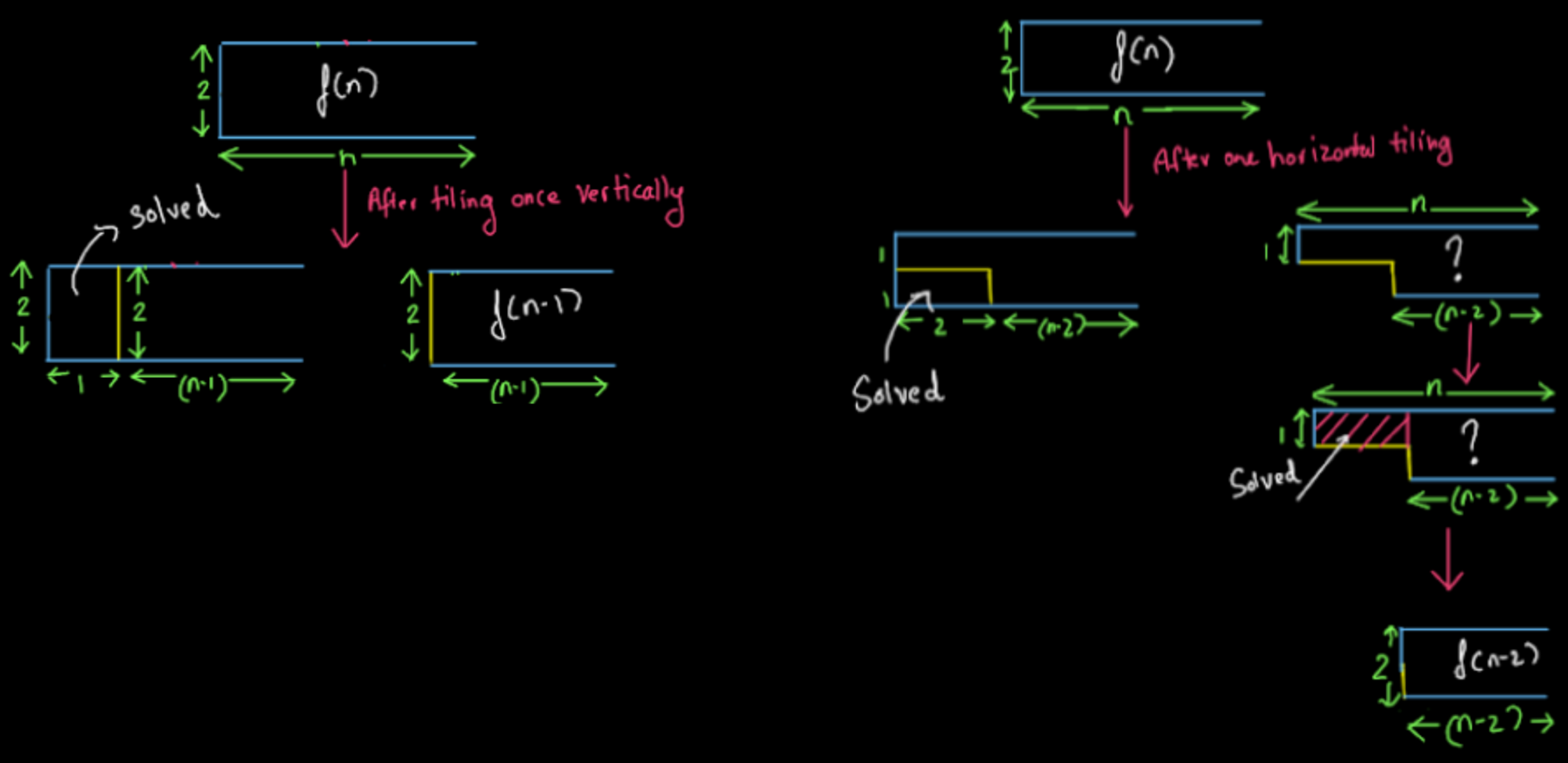




$$\Rightarrow \text{tiling}(2*N) = \text{tiling}(2*(N-1)) + \text{tiling}(2*(N-2))$$

$$\underbrace{dp[N] = dp[N-1] + dp[N-2]}$$

Fibonacci Exactly



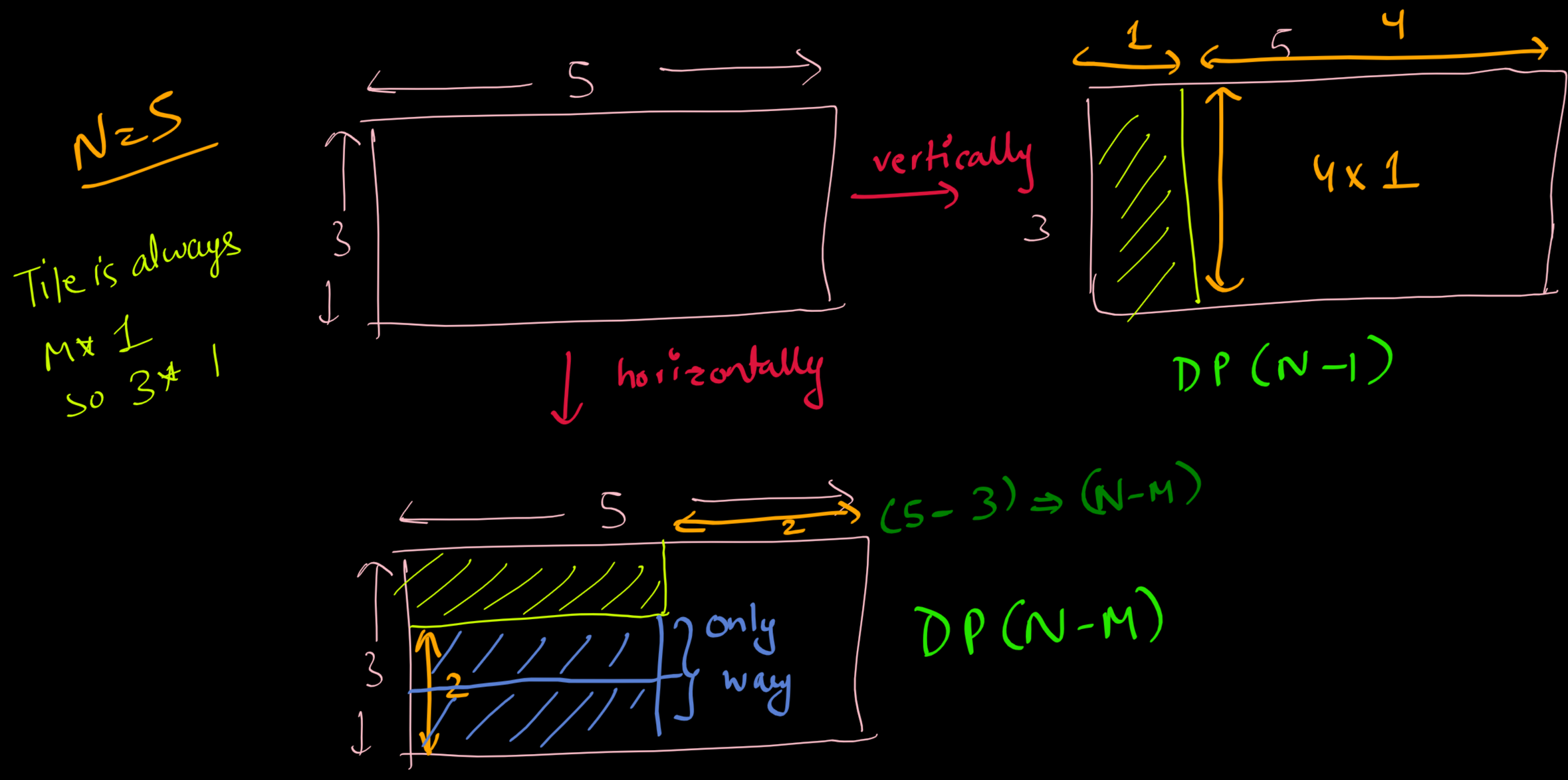

```java
class Solution {
    static Long numberOfWays(int N) {
        // code here
         long[] dp = new long[N + 1];
        if(N == 0) return 0l;
        if(N == 1) return 1l;
        if(N == 2) return 2l;

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

        for (int i = 3; i < dp.length; i++)
        {
          dp[i] = (dp[i - 1] + dp[i - 2]) % 1000000007;
        }

        return dp[N];
    }
};
```

Tabulation Code

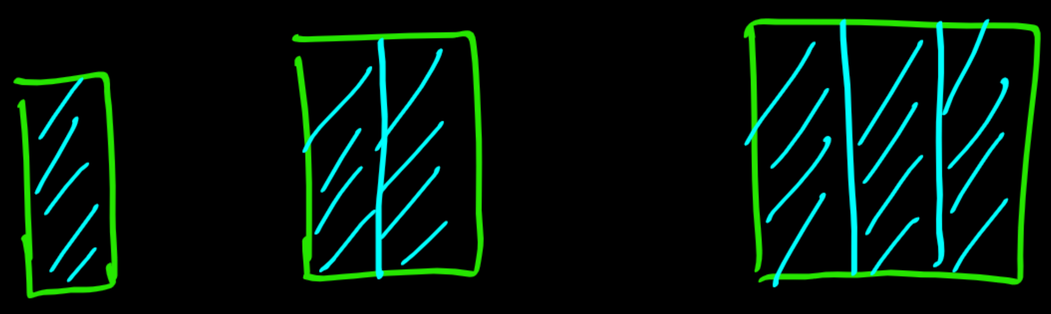
Tiling Problem - II

$$DP(N) = DP(N-1) + DP(N-M)$$

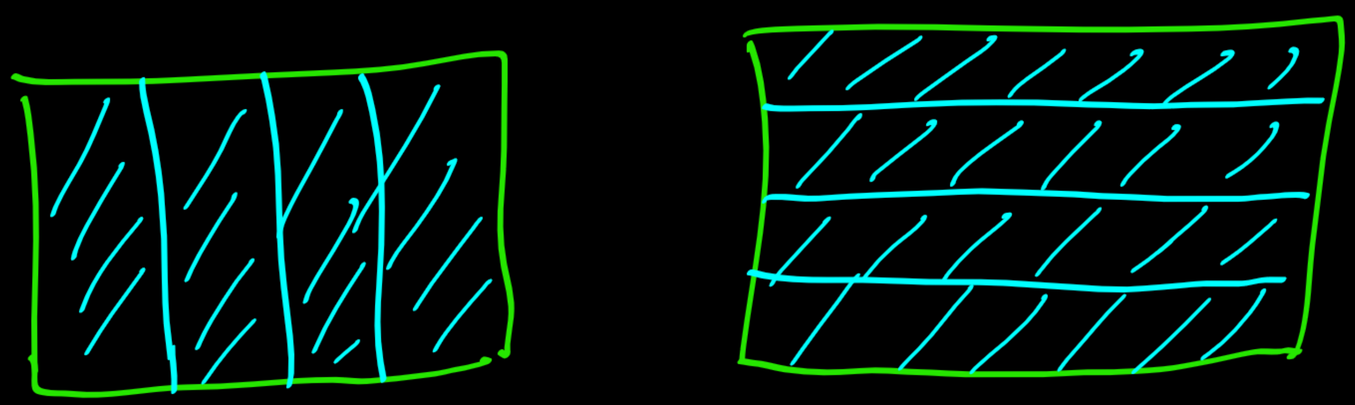
↑ vertically ↑ horizontally -



If $N < M \Rightarrow ans = 1$



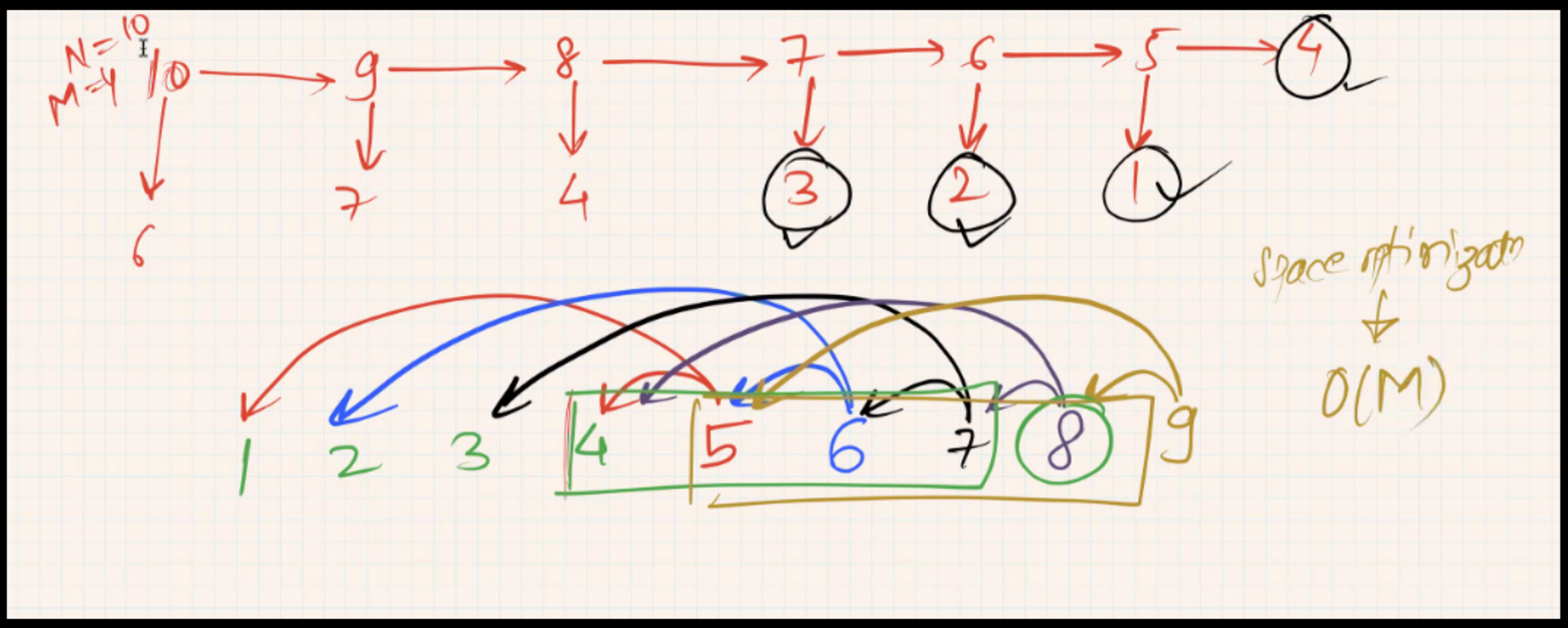
If $N = M \Rightarrow ans = 2$



```java
class Solution
{
    int mod = 1000000007;
    public int memo(int n, int m, int[] dp) {

        if(n < m) return 1;
        if(n == m) return 2;
        if(dp[n] != -1) return dp[n];

        int ans1 = memo(n-1,m,dp);
        int ans2 = memo(n-m,m,dp);

        return dp[n] = (ans1 + ans2) % mod;
    }

    public int countWays(int n, int m)
    {
        // Code here
        int[] dp = new int[n + 1];
        Arrays.fill(dp,-1);
        return memo(n,m,dp);
    }
}
```

Memoization Code

Space Optimization using Sliding Window



This will reduce the TC from O(N) to O(M)

```java
class Solution
{
    int mod = 1000000007;
    public int countWays(int n, int m)
    {
        // Code here
        if(n < m) return 1;
        if(n == m) return 2;
        Deque<Integer> dp = new ArrayDeque<>();

        for(int i=1;i<m;i++) {
            dp.add(1); //for states N < M ans is 1
        }

        dp.add(2); //for states N = M, ans is 2

        for(int i=m+1;i<=n;i++) {
            int ans = (dp.getFirst() + dp.getLast()) % mod;
            dp.removeFirst();
            dp.addLast(ans);
        }

        return dp.getLast();
    }
}
```

Space Optimization

TC: O(N)

SC: O(M)