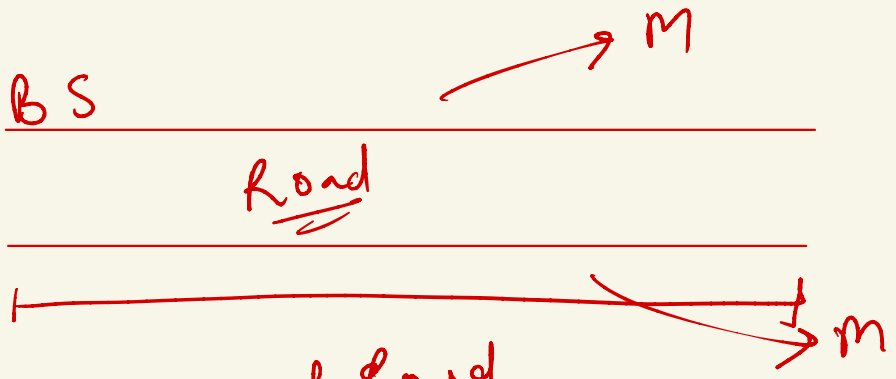
Dynamic Programming

FJP - 2

By:
Ritik Ramuka
DSA (Mentor)
8287829959
Pepcoding

Arrange Buildings

No Consecutive Buildings



n = size of Road

ans = m * m

S → B
B → S
S →

→ Side Arrangement

m → side 1 Arrangement
n → side 2 Arrangement

total = $\{m \times n\}$

Ques

No consecutive Buildings

BS ⇄ S, B → m

---

n → one Road

→ m

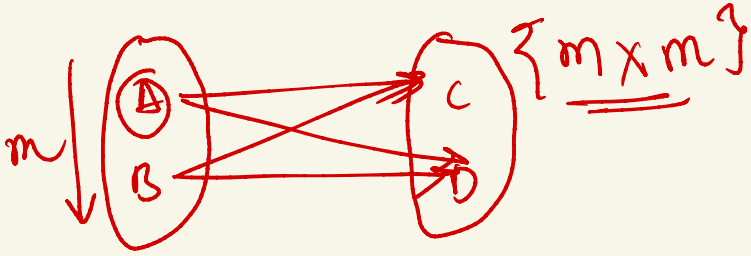
B → S

S → S

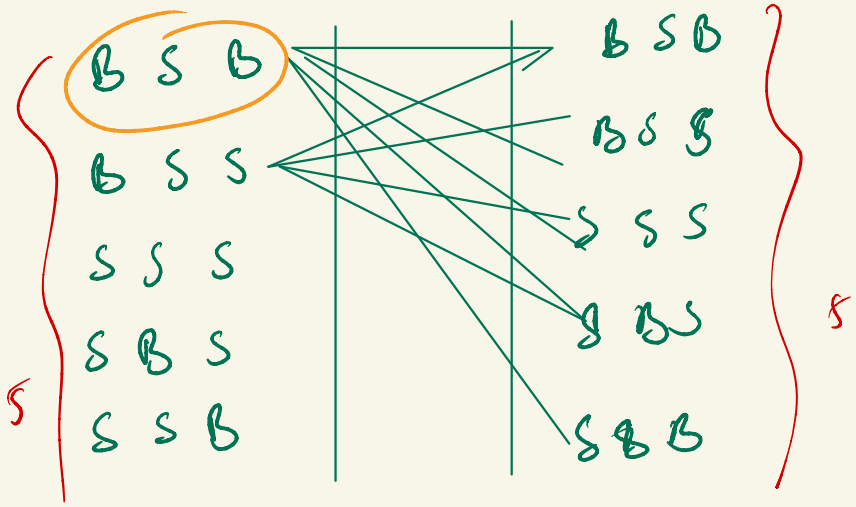
S → B

] → m

ans = $\{m * m\}$



Eg $n = 3$



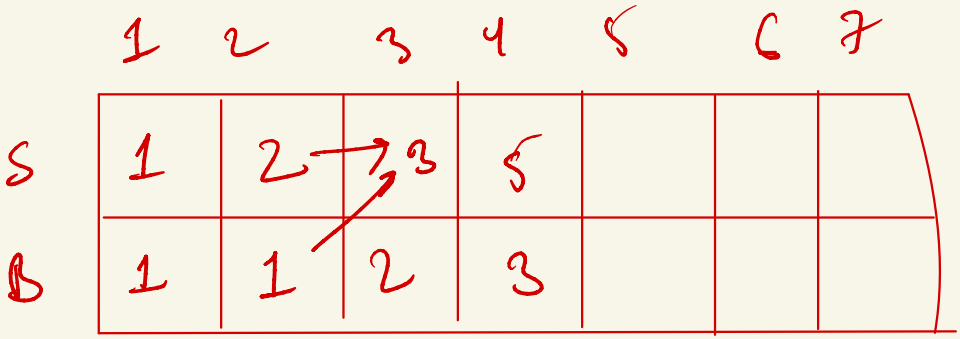
$5 * 5$

$= 5 + 5 + \text{---} \; m \text{ time}$ (5)

$= 5(1+1+\text{---}\, m)$

$= 5 * 5$

* Use long



$n = 4$

total way of side 1 = $5 \text{ C } 3 = 8$

total ways = $8 \times 8 = 64$

```java
import java.io.*;
import java.util.*;

public class Main{

public static void main(String[] args) throws Exception {
    // write your code here
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    // n == 1
    long space = 1;
    long building = 1;

    for(int i = 2; i <= n; i++) {
        // if adding space, BS', SS'
        long ns = space + building;
        // if adding building, SB'
        long nb = space;
        space = ns;
        building = nb;
    }

    long ans = (space + building) * (space + building);
    System.out.println(ans);
 }

}
```

$TC = O(N)$

$SC = O(1)$

Friends Pairing

$n \rightarrow$ friends

→ play solo

→ play in pair

↓ $(n-1)$

$\{f_1, f_2, f_3, f_4\}$

$(f_1, f_2) \rightarrow (f_3, f_4)$

$\rightarrow (f_3), (f_4)$

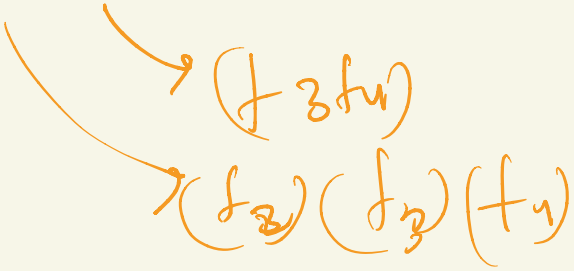
$(f_1, f_3) \rightarrow (f_2, f_4)$

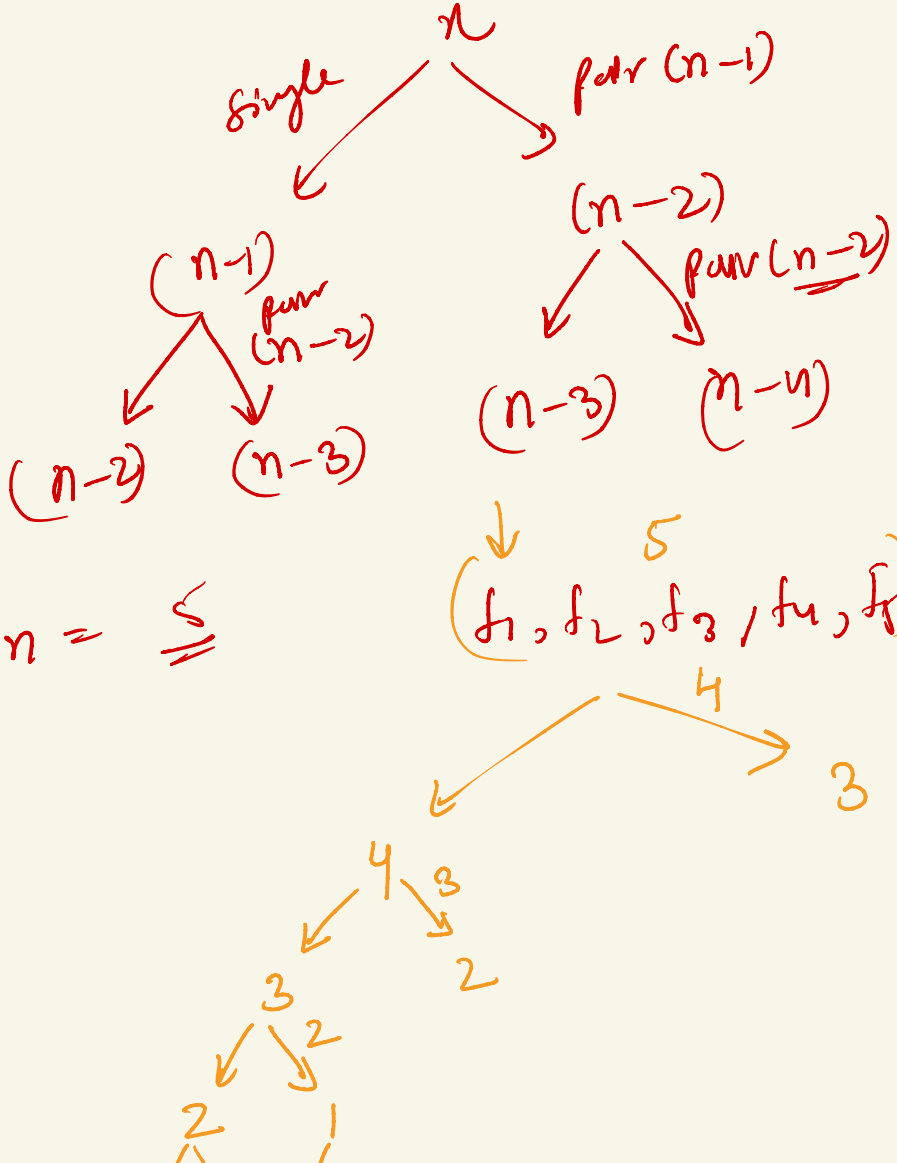
$\rightarrow (f_2), (f_4)$

(f_1, f_4)

$(f_1) \rightarrow (f_2 f_3)$

$\rightarrow (f_2 f_4)$


(f3 f4)
(f2)(f3)(f4)


n
single
pair (n-1)
(n-1)
pair (n-2)
(n-2)
(n-3)
(n-2)
pair (n-2)
(n-3)
(n-4)
n = 5
5
f1, f2, f3, f4, f5
4
3
4
3
2
3
2
2
1

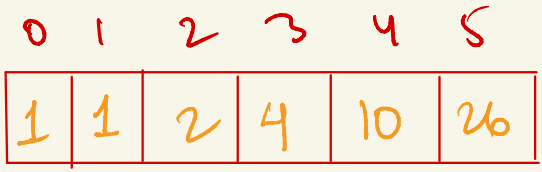
```
fn (n) {
    int single = paired;
    paired = (n-1) * f(n-2);
    single = 1 * f(n-1);
    return. pair + single;
}
```

```java
public static int rec(int n) {
    if(n == 0) {
        return 1;
    }

    int single = 0, pair = 0;
    if(n - 2 >= 0) {
        pair = (n - 1) * rec(n - 2);
    }

    single = 1 * rec(n - 1);

    return single + pair;
}
```

```java
public static int memo(int n, int[] dp) {
    if(n == 0) {
        return dp[n] = 1;
    }

    if(dp[n] != 0) {
        return dp[n];
    }

    int single = 0, pair = 0;
    if(n - 2 >= 0) {
        pair = (n - 1) * memo(n - 2, dp);
    }

    single = 1 * memo(n - 1, dp);

    return dp[n] = single + pair;
}
```

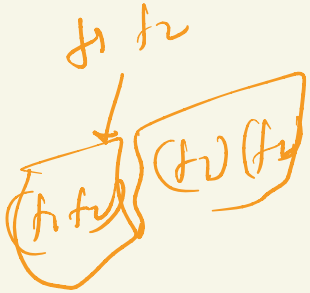
```java
// tabulation
int[] dp = new int[n + 1];
for(int i = 0; i <= n; i++) {
    if(i == 0) {
        dp[i] = 1;
        continue;
    }

    int single = 0, pair = 0;
    if(i - 2 >= 0) {
        pair = (i - 1) * dp[i - 2];
    }

    single = 1 * dp[i - 1];

    dp[i] = single + pair;
}
System.out.println(dp[n]);
```

n=5



p=1

s=1



TC = O(N)

SC = O(N)

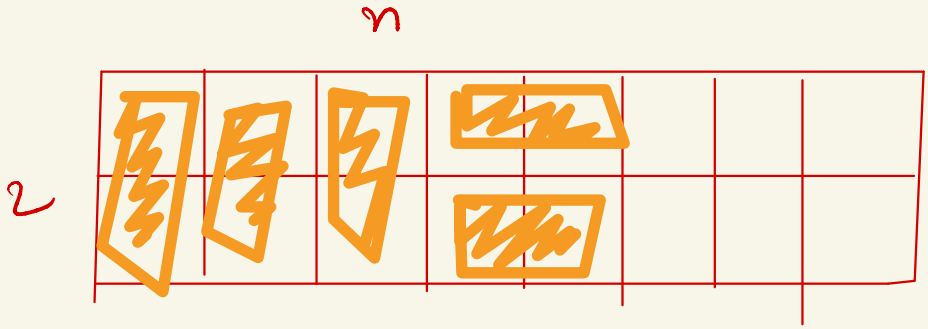
Space Optimization

11:10

TC: O(N)

SC: O(1)

Tiling with 2×1 tiles

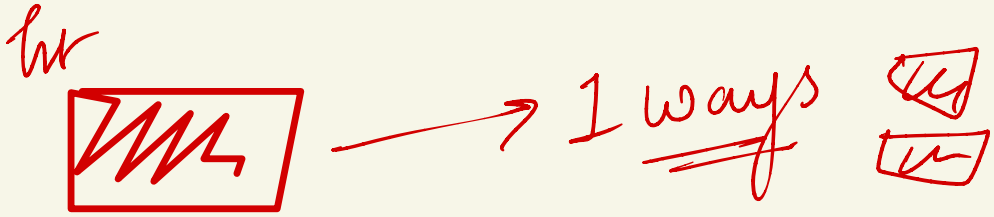


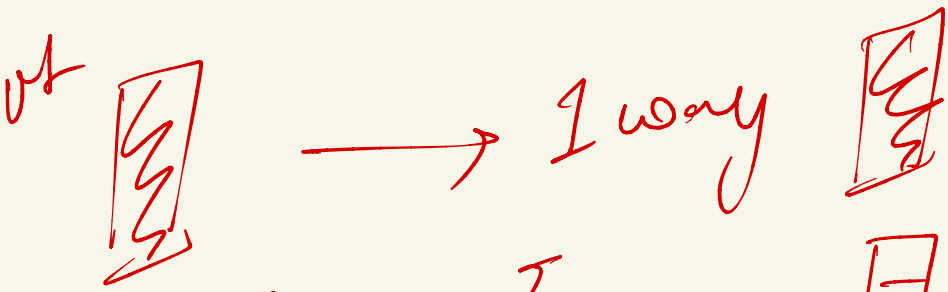
$2 \times n$ → path

2×1 → tile

ht → lateral space 2 unit

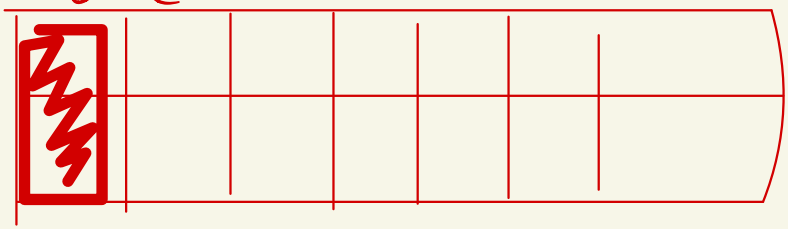
vt → lateral space 1 unit





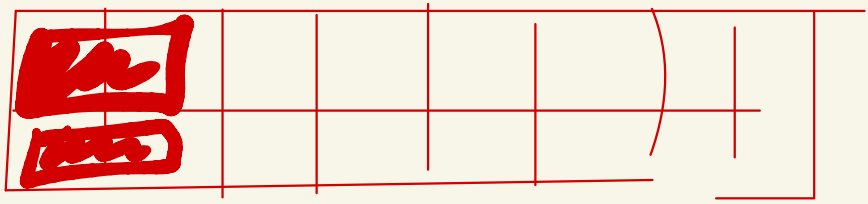

```
fun(n){
if(n==0) return 1;
if(n-2>=0){
int bt = fn(n-2);}
int ot = fn(n-1);
return bt + ot;
}
```

1 (n-1)



vt = 1 * fn(n-1)

(n-2)



ht = 1 * f(n-2)

n == 0
return 1

n == 1
return 1

any: ht + vt

```java
public static int rec(int n) {
    if(n == 0) {
        return 1;
    }

    int hz = 0, vt = 0;
    if(n - 2 >= 0) {
        hz = 1 * rec(n - 2);
    }
    vt = 1 * rec(n - 1);

    return vt + hz;
}

public static int memo(int n, int[] dp) {
    if(n == 0) {
        return dp[n] = 1;
    }

    if(dp[n] != 0) {
        return dp[n];
    }

    int hz = 0, vt = 0;
    if(n - 2 >= 0) {
        hz = 1 * memo(n - 2, dp);
    }
    vt = 1 * memo(n - 1, dp);

    return dp[n] = vt + hz;
}
```

```java
    // dp
    int[] dp = new int[n + 1];
    for(int i = 0; i <= n; i++) {
        if(i == 0) {
            dp[i] = 1;
            continue;
        }

        int hz = 0, vt = 0;
        if(i - 2 >= 0) {
            hz = 1 * dp[i - 2];
        }
        vt = 1 * dp[i - 1];

        dp[i] = vt + hz;
    }
    System.out.println(dp[n]);
```

```java
// space optimize
int prev1 = 1, prev2 = 1;
for(int i = 2; i <= n; i++) {
    int currAns = prev1 + prev2;
    prev2 = prev1;
    prev1 = currAns;
}
System.out.println(prev1);
```

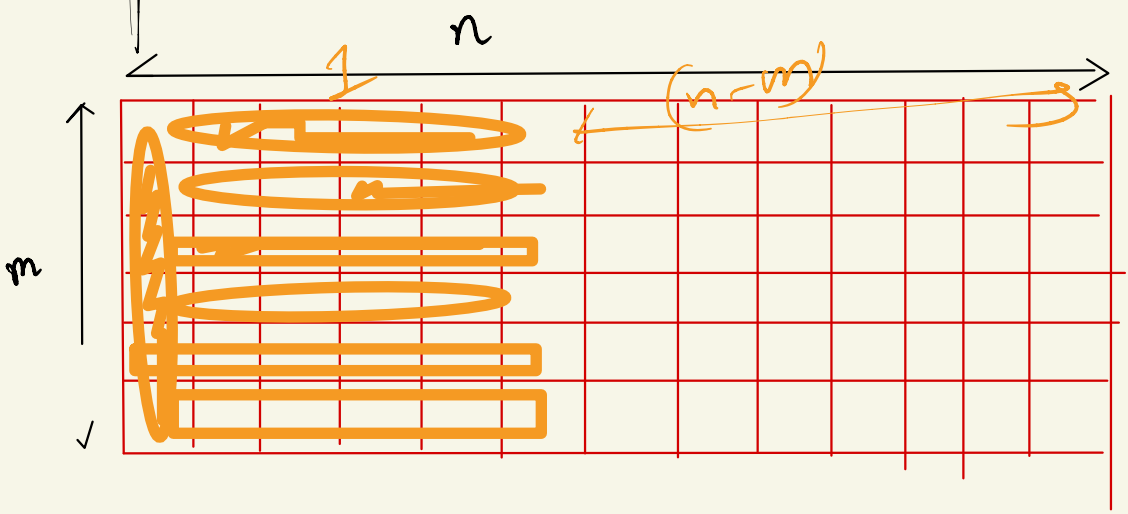
TC = O(N)

SC = O(1)

Tiling with 1*m tiles

tile dimension 1*m

path dimension m*n



```
fn (n , m) {
    if (n == 0) return 1;
    int lt = 0, ot = 0;
    if (n - m >= 0) {
        lt = fn(n - m) * 1
    }
    ot = fn(n - 1) * 1
    ans = ot + lt;
    return ans;
}
```

Space Opt. Not possible