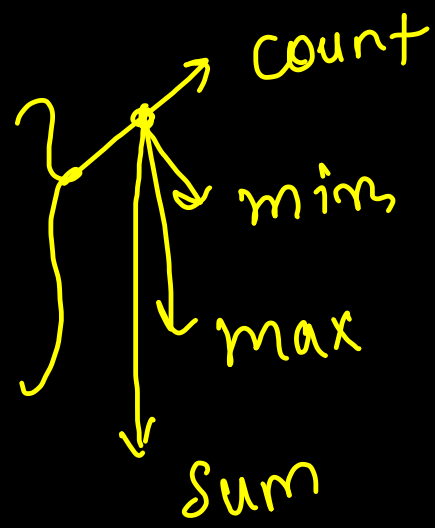
DP + Game Theory

- overlapping subproblems
- optimal substructure



{ 1, 100, -2, 3, 200 }

① Sum of picked value = maxm

② No two adjacent nodes

even 1 + (-2) + 200

vs

odd 100 + (3)

optimal 100 + 200

(1) Expectation / State

(idx, next) → 0 (no), 1 (yes)

dp[idx][next]

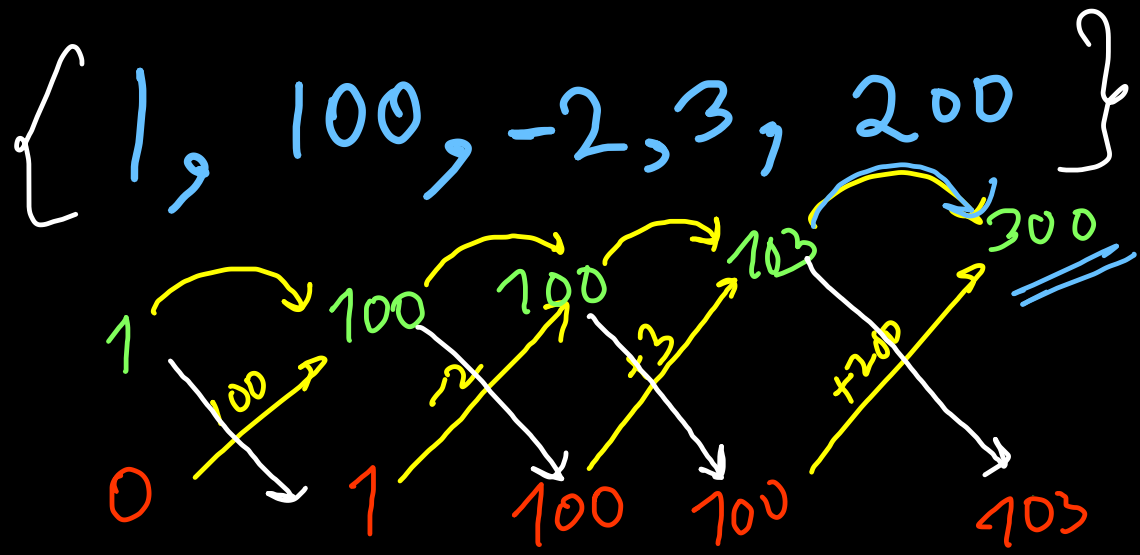
↳ maxm sum where choice is on idx & next item is having yes/no choice

optional yes

yes

no

Compulsory no



int helper(int idx, boolean prev)

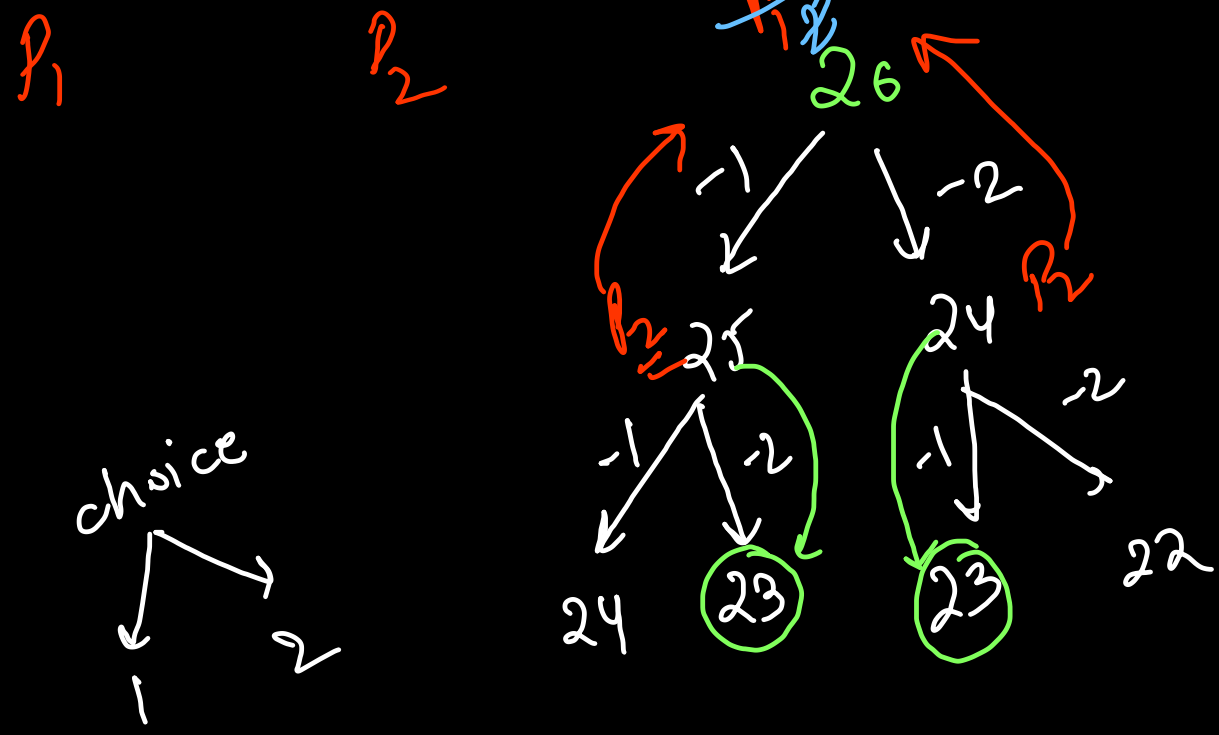
```java
public int rob(int[] nums) {
    int n = nums.length;
    int[] yes = new int[n];
    int[] no = new int[n];

    yes[0] = nums[0];
    for(int i = 1; i < n; i++) {
        yes[i] = Math.max(yes[i - 1], no[i - 1] + nums[i]);
        no[i] = yes[i - 1];
    }
    return yes[n - 1];
}
```

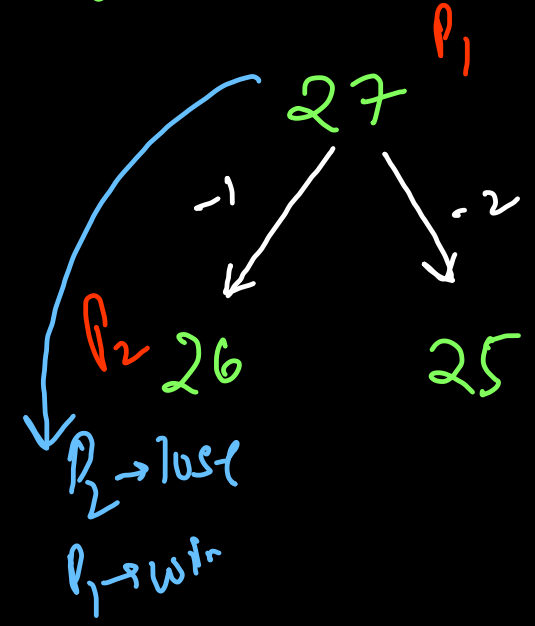
Time = O(n)

Space = O(n)

Game Theory

"You may assume → Both players play optimally"





Predict Winner

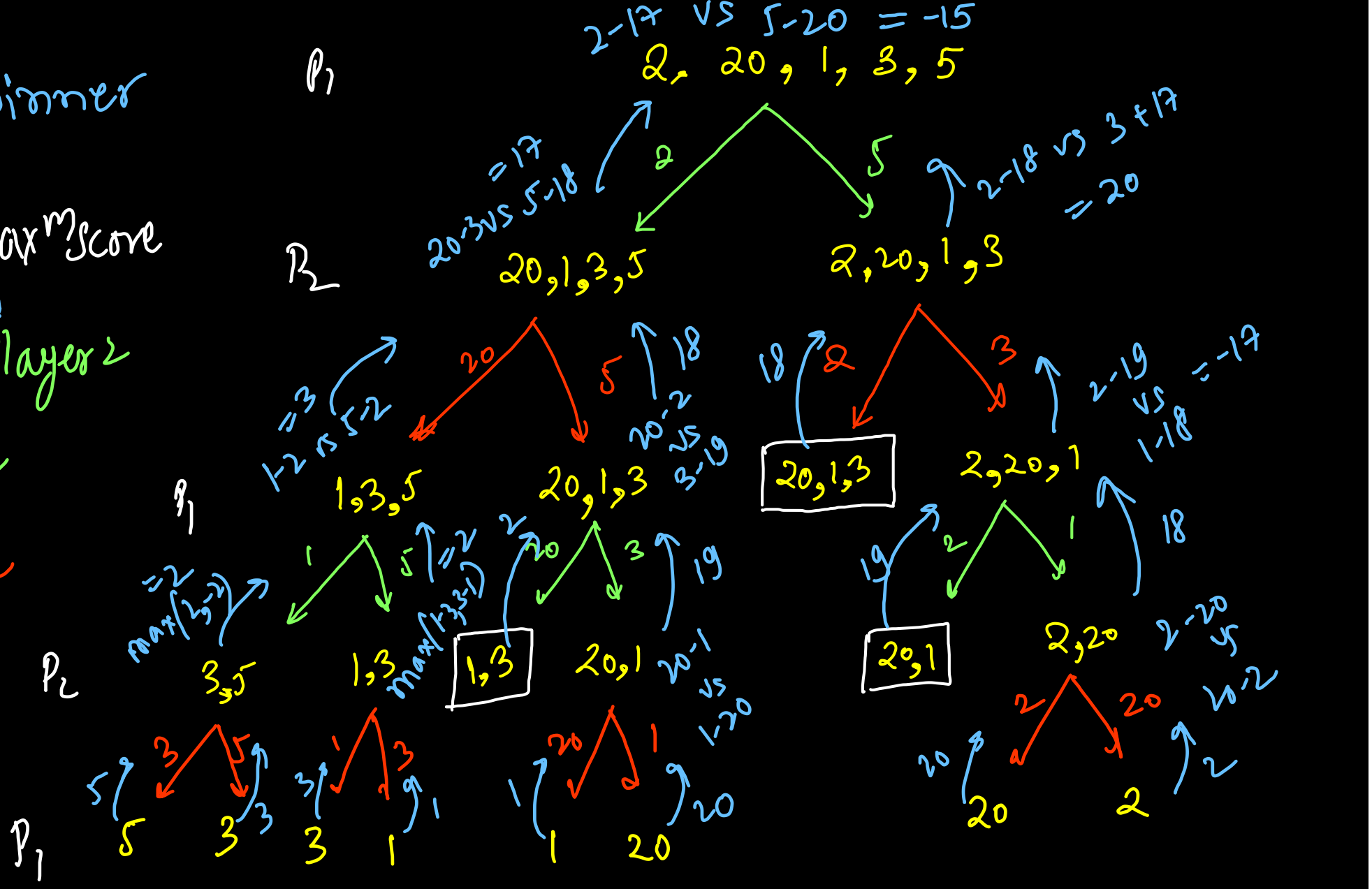
Player → Max Score

diff

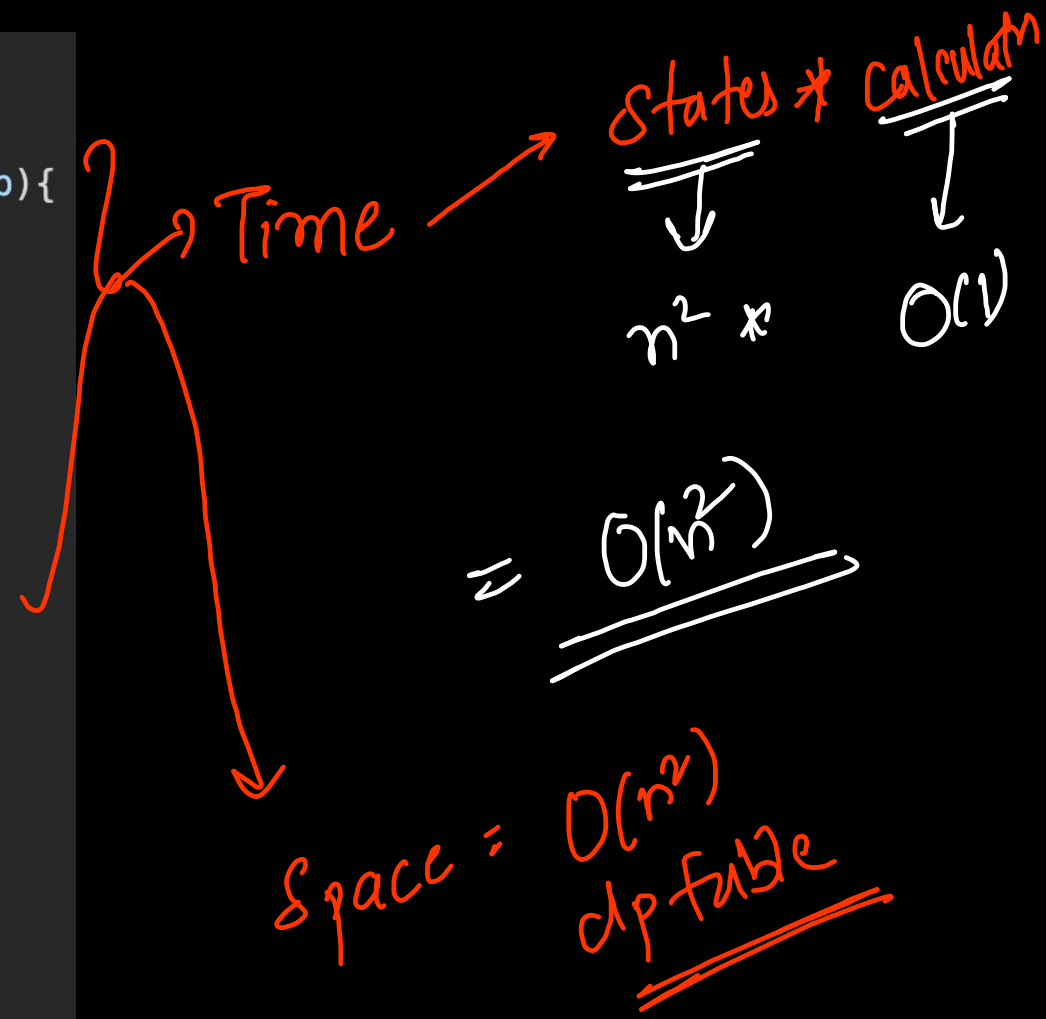
Player1 ≥ Player2

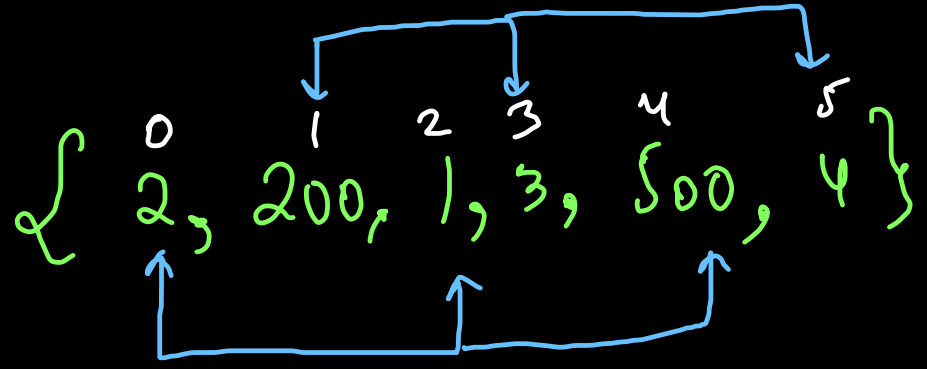
⇒ true

else false

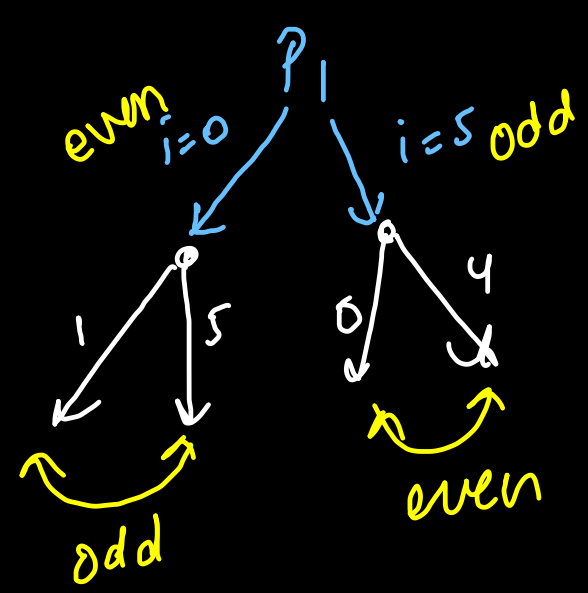


```java
int inf = Integer.MIN_VALUE;

public int helper(int l, int r, int[] nums, int[][] dp){
    if(l > r) return 0;
    if(l == r) return nums[l];
    if(dp[l][r] != inf) return dp[l][r];

    int first = nums[l] - helper(l + 1, r, nums, dp);
    int last = nums[r] - helper(l, r - 1, nums, dp);
    return dp[l][r] = Math.max(first, last);
}

public boolean PredictTheWinner(int[] nums) {
    int n = nums.length;
    int[][] dp = new int[n][n];
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            dp[i][j] = inf;

    int profit = helper(0, n - 1, nums, dp);
    return (profit >= 0);
}
```





LC 877) Stone Game



Player 1

Player 2

```
class Solution {
    public boolean stoneGame(int[] piles) {
        return true;
    }
}
```

↳ Greedy algorithm

2+1+500 (even) > 200+3+4 (odd)

① even length

② no tie (either P1/P2 will win)

0-1 Knapsack → electronics → dp (2d)

vs

Unbounded Knapsack → unlimited frequency → dp (1d)

vs

Fractional Knapsack → grocery store → greedy choice → Maximize → profit/weight

profits	100	200	300	400	500
	↓ 10	↓ 100	↓ 5	↓ 20	↓ 500
weight	10	2	60	20	1

total capacity = 22 kg → capacity exceeding

maximize (profit) →

CSES Dice Probability

independent → add
dependent → multiply

You throw a dice n times, and every throw produces an outcome between 1 and 6. What is the probability that the sum of outcomes is between a and b?

$n = 2$

$[a, b]$

$[9, 10]$

dp[2][9] + dp[2][10]

dp[2][9]

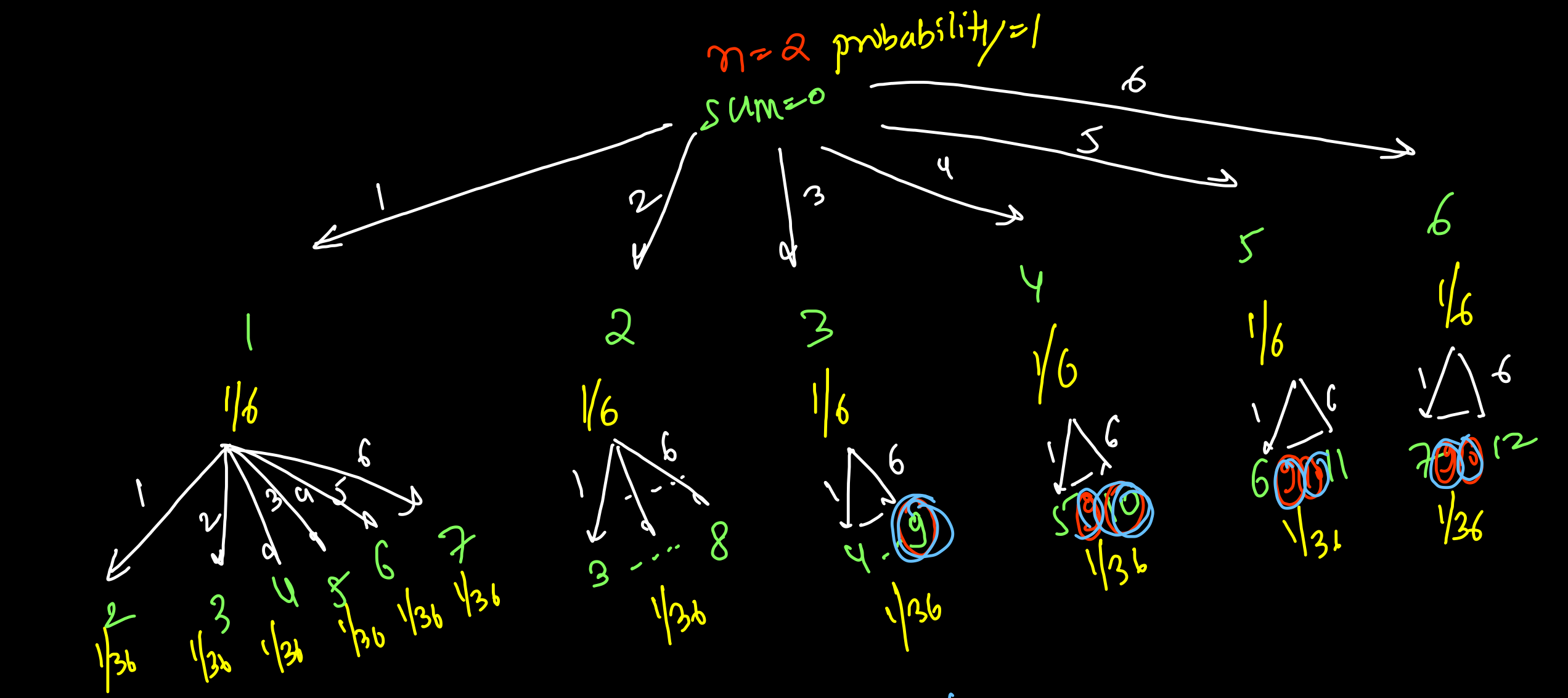
3+6
4+5
5+4
6+3

4/36

dp[2][10]

4+6
5+5
6+4

3/36


n=2 probability=1
sum=0
1
2
3
4
5
6
1
1/6
2
1/6
3
1/6
4
1/6
5
1/6
6
1/6
1/36
1/36
1/36
1/36
1/36
1/36
1/36
1/36
1/36
1/36
total answers= 7 * 1/36 = 7/36 = 0.194444

State → double dp[n][sum] (a ↔ b)

n → numbers of remaining dice rolls

transition

$$dp[n][sum] = \sum_{d=1}^{6} dp[n-1][sum-d]$$

base case

```
if (sum < 0 || n < 0) return 0.0;
if (sum == 0 && n == 0) return 1.0;
```

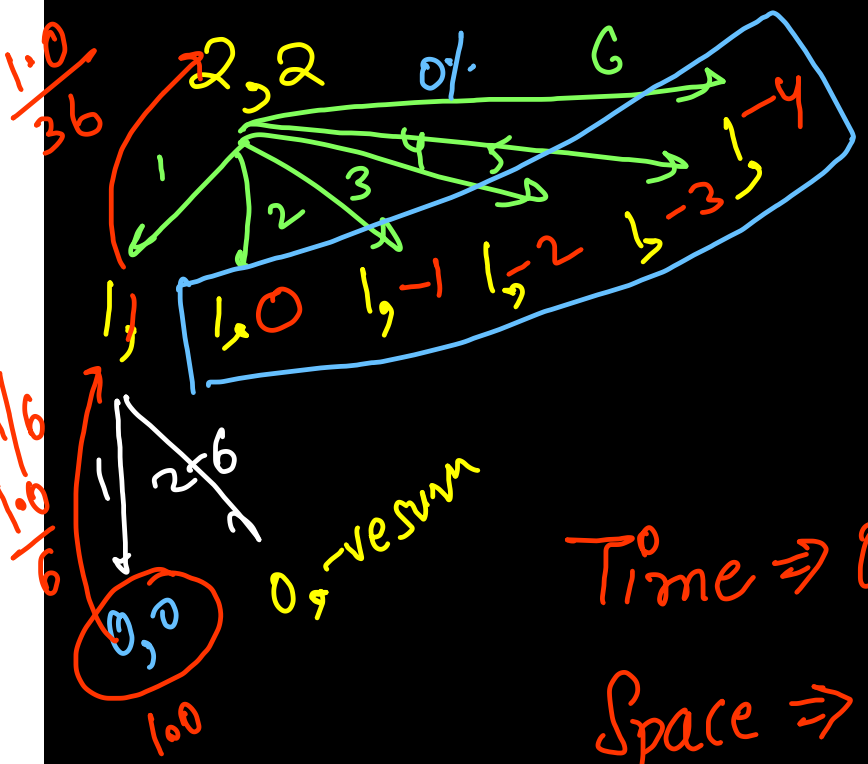
```java
public static double helper(int n, int sum, double[][] dp){
    if(n == 0 && sum == 0) return 1.0;
    if(n <= 0 || sum <= 0) return 0.0;
    if(dp[n][sum] != -1.0) return dp[n][sum]; // memoization

    double ans = 0.0;
    for(int d = 1; d <= 6; d++){
        ans += helper(n - 1, sum - d, dp) * (1.0 / 6.0);
    }
    return dp[n][sum] = ans;
}

public static void solve() throws Exception {
    int n = scn.nextInt();
    int a = scn.nextInt();
    int b = scn.nextInt();


    double[][] dp = new double[n + 1][b + 1];
    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= b; j++){
            dp[i][j] = -1.0;
        }
    }

    double ans = 0;
    for(int sum = a; sum <= b; sum++){
        ans += helper(n, sum, dp);
    }
    out.println(String.format("%.6f", ans));
}
```

Unvisited

fixed 6 decimal places

$n = 2$, $sum = [2, 5]$

$dp[2][2]$, $dp[2][3]$, $dp[2][4]$, $dp[2][5]$

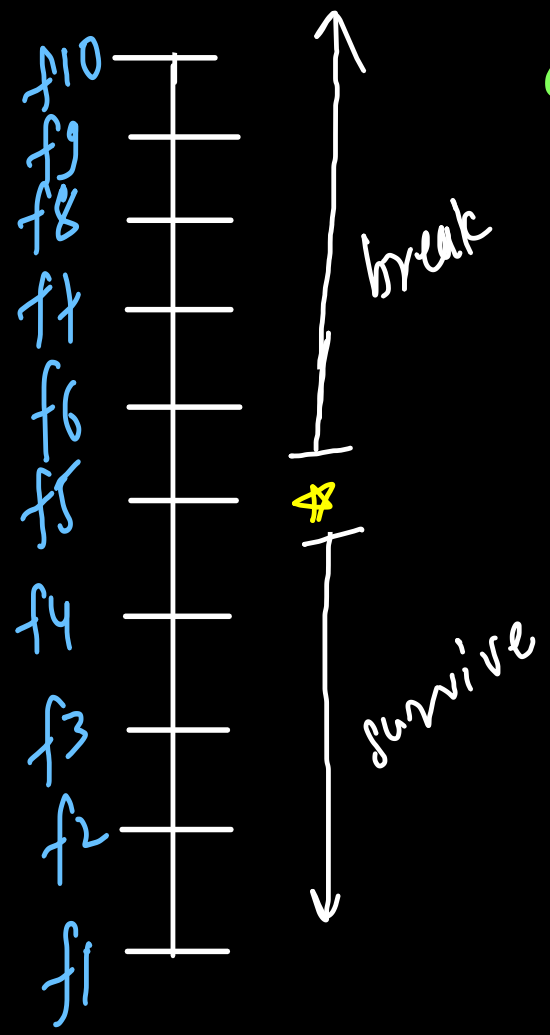


Time $\Rightarrow O(n \times b \times 6)$ — $100 \times 600 \times 6$

Space $\Rightarrow O(n \times b)$

Egg dropping puzzle

floors = 10



eggs = 1

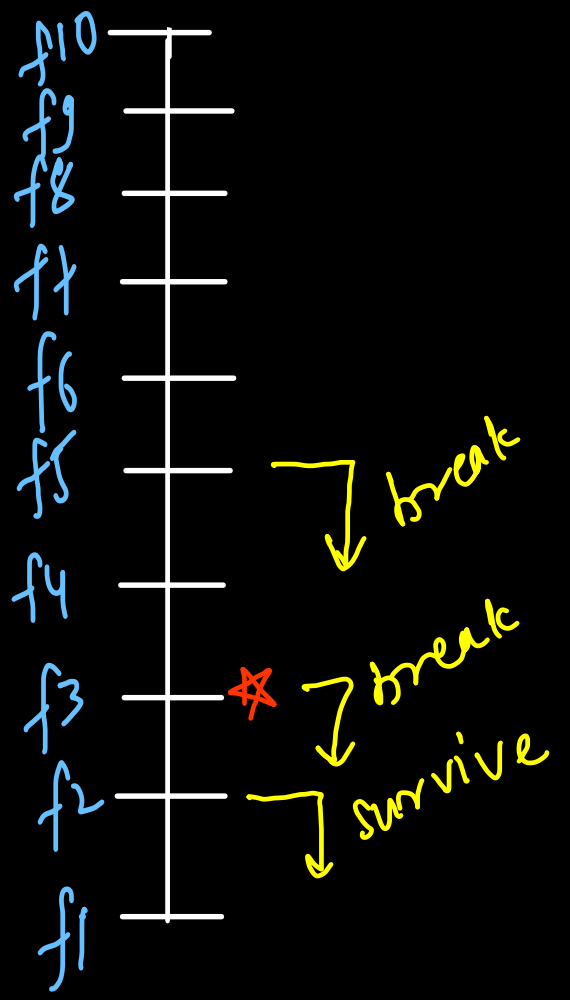
guaranteed min^m moves to find out the critical floor!

"critical floor"

eggs == 1 ⇒ floors moves (linear search)

eggs = ∞ (e ≥ f), floors = f

Binary search



low = 1 high = 10

mid = (1+10)/2 = 5

low = 1 high = 4

mid = (1+4)/2 = 2

low = 3, high = 4

mid = (3+4)/2 = 3

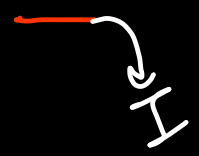
$\log_2 n$

eggs = 2 , floors = f = 100

{ puzzle / google } weighted search

f = 1

1 step

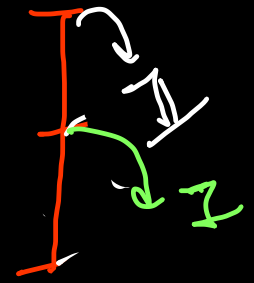


f = 2

1 step

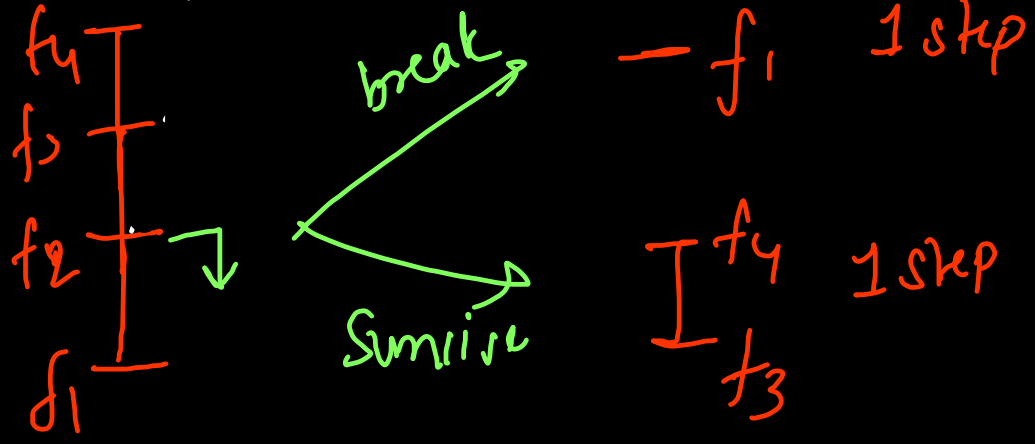
f = 3

2 step



f = 4

2 step



eggs = e, floors = f

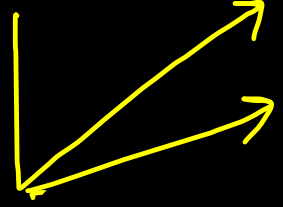
$$dp[e][f] = 1 + \text{choices } c=f \text{ to } c=1$$

survive → $dp[e][f-c]$

max(guaranteed)

break → $dp[e-1][c-1]$

base case



```
if (e==1) return floors;
if (f==1) return 1;
```

```java
int eggDrop(int floors, int eggs, int[][] dp){
    if(floors == 0) return 0;
    if(eggs == 1) return floors; // linear search
    if(dp[floors][eggs] != -1) return dp[floors][eggs];

    int ans = Integer.MAX_VALUE;
    for(int c = 1; c <= floors; c++){
        int breaks = eggDrop(c - 1, eggs - 1, dp);
        int survive = eggDrop(floors - c, eggs, dp);
        ans = Math.min(ans, 1 + Math.max(breaks, survive));
    }

    return dp[floors][eggs] = ans;
}
```

TLE

$O(f^2 \times e)$

```java
public int superEggDrop(int eggs, int floors) {
    int[][] dp = new int[floors + 1][eggs + 1];
    for(int i=0; i<=floors; i++){
        for(int j=0; j<=eggs; j++){
            dp[i][j] = -1;
        }
    }
    return eggDrop(floors, eggs, dp);
}
```

```
int eggDrop(int floors, int eggs, int[][] dp){
    if(floors == 0) return 0;
    if(eggs == 1) return floors;
    if(dp[floors][eggs] != -1) return dp[floors][eggs];

    int low = 1, high = floors, ans = floors;
    while(low <= high){
        int mid = low + (high - low) / 2;

        int breaks = eggDrop(mid - 1, eggs - 1, dp);
        int survive = eggDrop(floors - mid, eggs, dp);

        ans = Math.min(ans, 1 + Math.max(breaks, survive));
        if(breaks < survive) low = mid + 1;
        else high = mid - 1;
    }

    return dp[floors][eggs] = ans;
}
```

f x e

log f

O(f x e x log f)

Accepted