DP- lecture (14)

Buy & sell stocks

→ Infinite Transactions

→ Transaction fee

} Recursive code ← memoization

→ Cooldown

→ 2 Transaction Fees

→ K Transaction Fees

Buy & sell stock

{11, 6, 7, 19, 4, 1, 6, 18, 4}

19-6 = 13

18-1 = 17

infinite transactions

with cooldown

$$\begin{cases} dp[buy] = dp[buy-1],\ dp[sell-2]-p[i] \\ dp[sell] = dp[sell-1],\ dp[buy-1]+p[i] \end{cases}$$

BS BS

1st transaction

2nd transaction

SB

BBS

BSS

11 6 7 19 4 1 6 18 4

(extra stock) Buy	Sell (0 extra stock)
-11	0
-11 vs -6 → -6	0 vs -11+6 → 0
-6 vs 0-7 → -6	0 vs -11+7 → 1
0-19 vs -6 → -6	1 vs -6+19 → 13
-6 vs 1-4 → -3	13 vs -6+4 → 13
-3 vs 13-1 → 12	13 vs -3+1 → 13
13-6 vs 12 → 12	13 vs 12+6 → 18
13-18 vs 12 → 12	18 vs 12+18 → 30
18-4 vs 12 → 14	30 vs 12+4 → 30

```java
class Solution {
    public int maxProfit(int[] prices) {
         if(prices.length <= 1) return 0;

        int[] buy = new int[prices.length];
        int[] sell = new int[prices.length];

        buy[0] = -prices[0];
        sell[0] = 0;
        buy[1] = Math.max(-prices[0], -prices[1]); // Either buy 0th stock or 1st stock
        sell[1] = Math.max(0, prices[1] - prices[0]); // Either do nothing or Buy 0 Sell 1

        for(int i=2; i<prices.length; i++){
            buy[i] = Math.max(buy[i - 1], sell[i - 2] - prices[i]);
            sell[i] = Math.max(sell[i - 1], buy[i - 1] + prices[i]);
        }

        return sell[prices.length - 1];
    }
}
```

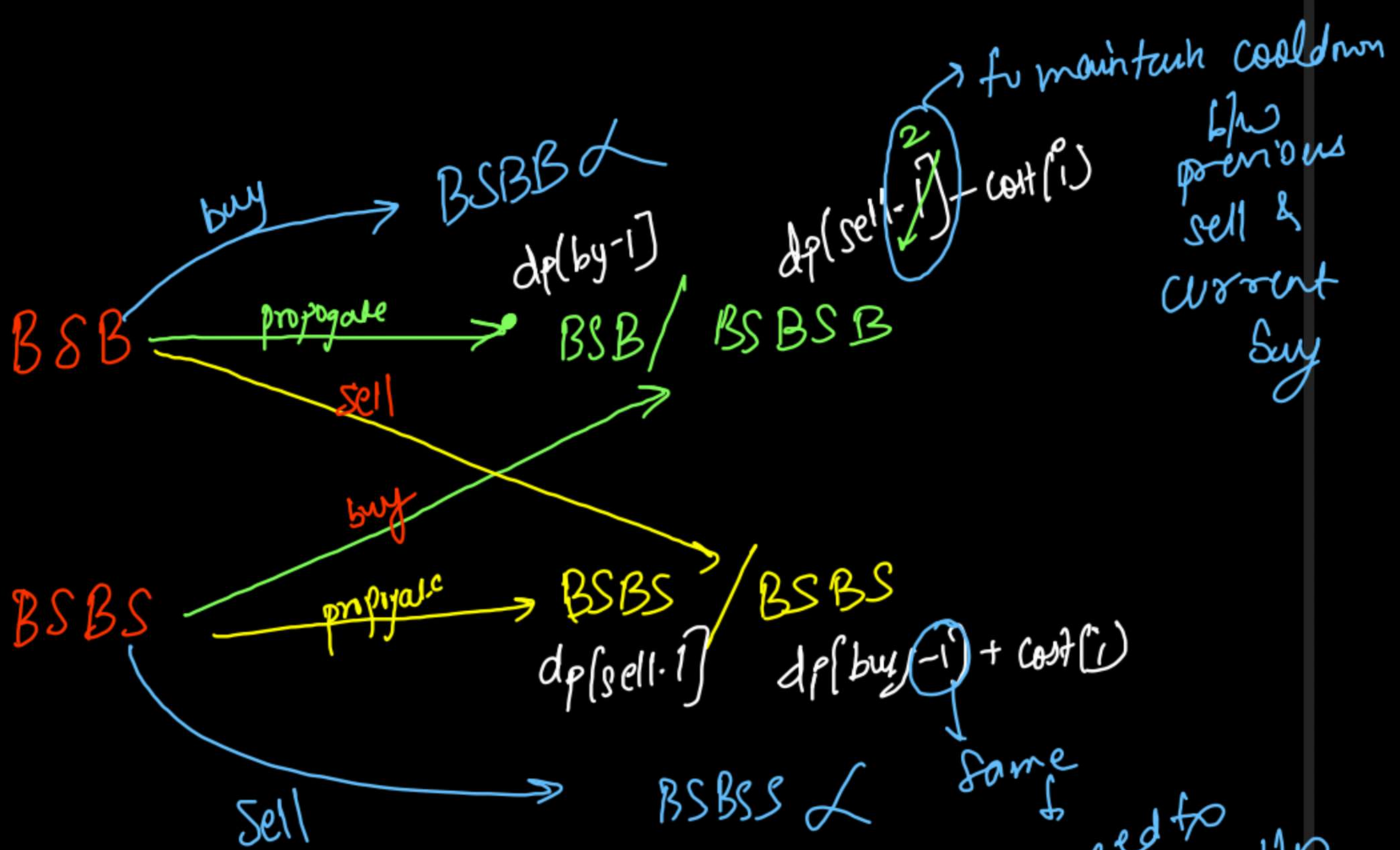
O(n) time

O(n) space

```java
class Solution {
    public int maxProfit(int[] prices) {
         if(prices.length <= 1) return 0;

        int buy0 = -prices[0];
        int sell0 = 0;
        int buy1 = Math.max(-prices[0], -prices[1]); // Either buy 0th stock or 1st stock
        int sell1 = Math.max(0, prices[1] - prices[0]); // Either do nothing or Buy 0 Sell 1

        for(int i=2; i<prices.length; i++){
            int newbuy = Math.max(buy1, sell0 - prices[i]);
            int newsell = Math.max(sell1, buy1 + prices[i]);

            buy0 = buy1; sell0 = sell1;
            buy1 = newbuy; sell1 = newsell;
        }

        return sell1;
    }
}
```

O(n) time

O(1) space


buy
BSBB
dp[by-1]
BSB
propogate
BSB
BSBSB
dp[sell-1]
cost[i]
2
to maintain cooldown b/w previous sell & current buy
sell
buy
BSBS
propogate
BSBS
dp[sell-1]
BSBS
dp[buy(-1)] + cost[i]
Sell
BSBSS
same
no need to cooldown b/w a transaction

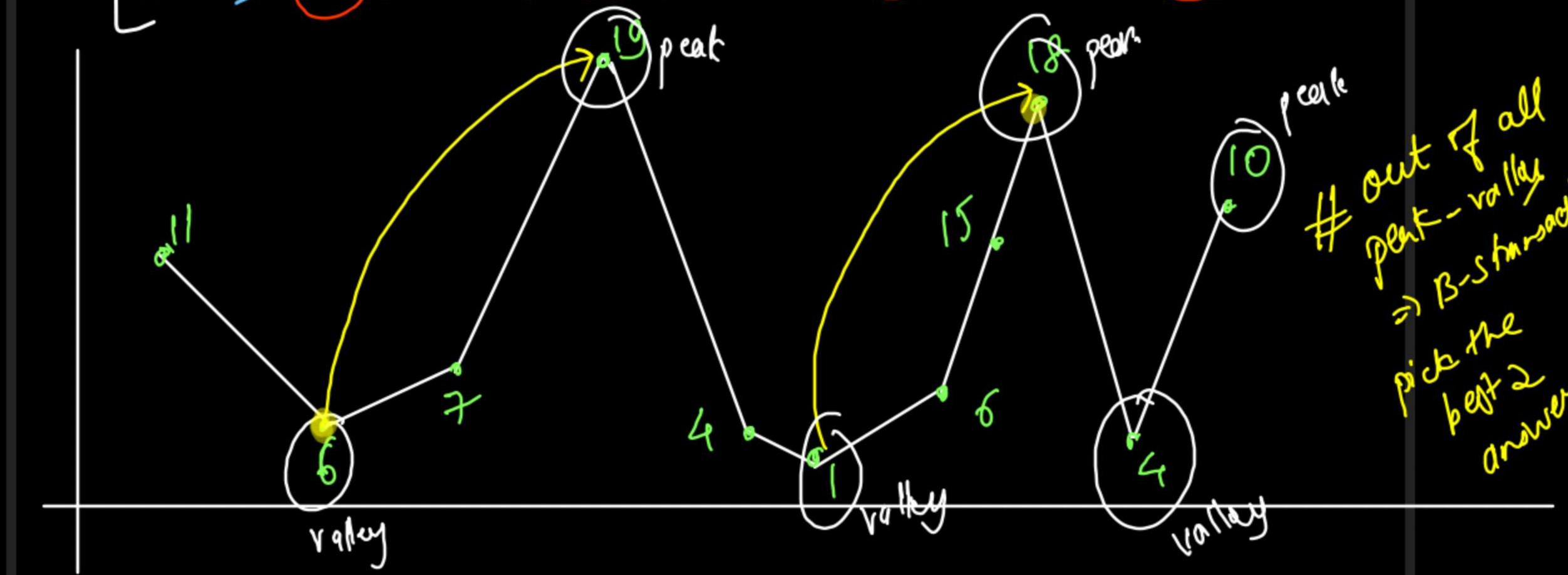
Infinite → Greedy
DP

Buy & sell Stocks → 2 Transactions
↕
Atmost

{11, 6, 7, 19, 4, 1, 6, 15, 18, 4, 10}

19-6

18-1



out of all peak-valley
⇒ B-S stock is
pick the best 2 answers

(122) #Infinite Transactions (Greedy)

```
class Solution {
    public int maxProfit(int[] prices) {
        int valley = 0, profit = 0;
        while(valley < prices.length){
            int peak = valley;
            while(peak + 1 < prices.length && prices[peak + 1] >= prices[peak]){
                peak++;
            }

            profit = profit + (prices[peak] - prices[valley]);
            valley = peak + 1;
        }
        return profit;
    }
}
```

buying day → R to L

selling day → L to R

17	17	17	17	17	7	6	6	6	0
+	+	+	+	+	+	+	+	+	+
0	0	1	13	13	13	14	17	17	17

{11, 6, 7, 19, 4, 1, 15, 18, 4, 10}

19-6

18-1

19 peak

18 peak

10 peak

11

7

4

15

6

valley

valley

valley

```
int[] selling = new int[prices.length];

int min = prices[0];
for(int i=0; i<prices.length; i++){
    min = Math.min(min, prices[i]);

    if(i - 1 >= 0)
        selling[i] = Math.max(selling[i - 1], prices[i] - min);
    else selling[i] = prices[i] - min;
}
```

Greedy

O(N) time

O(N) space

```
int[] buying = new int[prices.length];

int max = prices[prices.length - 1];
for(int i=prices.length-1; i>=0; i--){
    max = Math.max(prices[i], max);

    if(i != prices.length - 1)
        buying[i] = Math.max(buying[i+ 1], max - prices[i]);

    else buying[i] = max - prices[i];
}
```

```
int ans = 0;
for(int i=0; i<prices.length; i++){
    ans = Math.max(ans, buying[i] + selling[i]);
}
return ans;
```

GFG

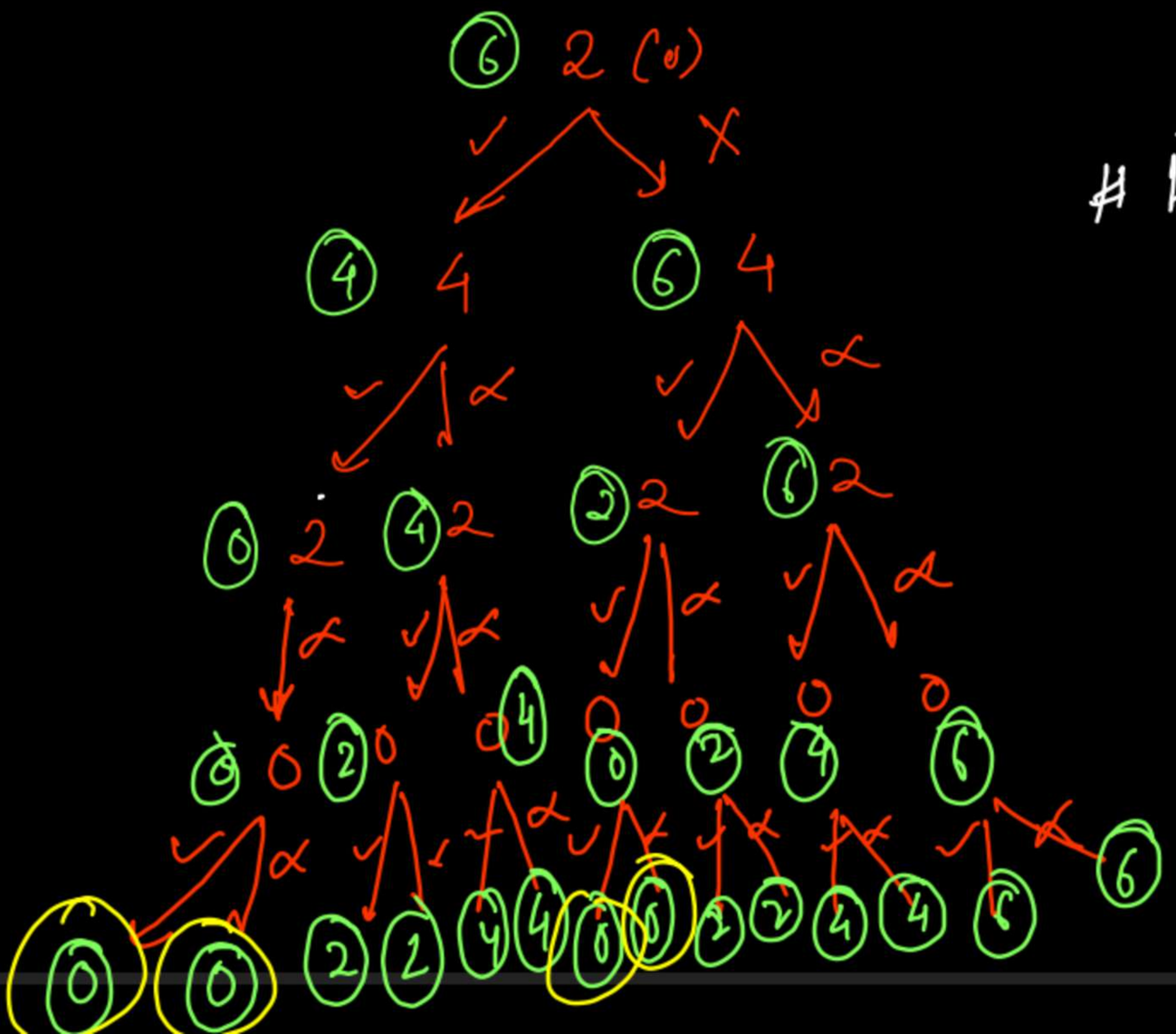
Count Target Sum Subset { Perfect Sum }

2D DP

DP state
- index
- rem Target

0	1	2	3
2	4	2	0

target = 6



Ans
- {2,4}
- {2,4,0}
- {4,2}
- {4,2,0}

14

If 0s are present & count the subsets, you will have to go to the last level

```
if (target == 0)
    return 1;
```

```java
public int memo(int index, int target, int[] arr, int[][] dp){
    if(index == arr.length){
        if(target == 0) return 1;
        return 0;
    }

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

    int no = memo(index + 1, target, arr, dp);
    int yes = (target >= arr[index])
            ? memo(index + 1, target - arr[index], arr, dp) : 0;
    return dp[index][target] = (no + yes) % 1000000007;
}
```

Memo

Time → O(N * Target)

Space → O(N * Target) (DP)

R-C-S → O(N)

```java
public int perfectSum(int arr[],int n, int sum)
{
    int[][] dp = new int[n + 1][sum + 1];
    for(int i=0; i<=n; i++){
        for(int j=0; j<=sum; j++){
            dp[i][j] = -1;
        }
    }
    return memo(0, sum, arr, dp);
}
```

{ 2, 4, 0, 0, 2 }
0 1 2 3 4

Tabulation { 0-1 knapsack (variation) } {∝ ∝} {∝, ŏ} {ŏ, ∝}, {0, 0}

10:45

	0	1	2	3	4	5	6 (Target)
0 ∝	1 {}	0	0	0	0	0	0
1 [2]	1 {2}	0	1 {2}	0	0	0	0
2 [4]	1 {4}	0	1 {2}	0	1 {4}	0	1 {2,4}
3 [0]	2 {0,4}	0	2 {2,0} {4,0}	0	2 {4,0} {4,0}	0	2 {2,4} {4,0}
4 [0]	2 {0,4} {4,0}	0	2 {2,0} {4,0} {2,0}	0	2 {4,0} {4,0} {4,0}	0	2 {2,4} {4,0} {4,0} {2,4,0}
5 [2]	4	0	8	0	8	0	8

Row by Row
Top to down

```java
public int perfectSum(int arr[],int n, int target)
{
    int[][] dp = new int[n + 1][target + 1];
    dp[0][0] = 1; // Empty Subset to form 0 Target

    for(int i=1; i<=n; i++){
        for(int j=0; j<=target; j++){
            int no = dp[i - 1][j]; // No Call
            int yes = (j >= arr[i - 1]) ? dp[i - 1][j - arr[i - 1]] : 0;

            dp[i][j] = (no + yes) % 1000000007;
        }
    }

    return dp[n][target];
}
```

Time → O(N * Target)

Space → O(N * Target)

2D DP

```java
public int perfectSum(int arr[],int n, int target)
{
    int[] dp = new int[target + 1];
    dp[0] = 1; // Empty Subset to form 0 Target

    for(int i=1; i<=n; i++){
        int[] newdp = new int[target + 1];

        for(int j=0; j<=target; j++){
            int no = dp[j]; // No Call
            int yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : 0;

            newdp[j] = (no + yes) % 1000000007;
        }

        dp = newdp;
    }

    return dp[target];
}
```

Time → O(N * Target)

Space → O(Target)

18

Check if Target Sum Subset is Present or Not

```java
public class Solution {
    public int solve(int[] arr, int target) {
        int n = arr.length;
        boolean[] dp = new boolean[target + 1];
        dp[0] = true; // Empty Subset to form 0 Target

        for(int i=1; i<=n; i++){
            boolean[] newdp = new boolean[target + 1];

            for(int j=0; j<=target; j++){
                boolean no = dp[j]; // No Call
                boolean yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : false;

                newdp[j] = no || yes;
            }

            dp = newdp;
        }

        return (dp[target] == true) ? 1 : 0;
    }
}
```

416 Partition Equal Subset Sum

$s_1 + s_2 = \text{total}$

$s_1 = s_2$

$2s_1 = \text{total}$

$s_1 = \frac{\text{total}}{2}$

eg 1 $\{2, 4, 1, 1, 2\}$ total = 10

$s_1 = s_2$

$\{2,1,2\}$ (5) $\{4,1\}$ (5)

true

mutually exclusive → Intersection = null

&

exhaustive → Union = N subsets

eg 2 $\{3, 4, 2, 5, 2, 1\}$ total = 17

$\{3,4,1\} \neq \{2,5,2\}$

odd ∴ false

Q2 $\{1, 2, 3, 8\}$ total = 14

$\{1,2,3\} \neq \{8\}$ false

S_1 S_2

Algorithm $S_1 + S_2 = total$; $S_1 = S_2$

$\Rightarrow$ total % 2 == 1 : false

$\Rightarrow$ checkTargetSumSubset(arr, total/2)

```java
public boolean checkTargetSumSubset(int[] arr, int target) {
    int n = arr.length;
    boolean[] dp = new boolean[target + 1];
    dp[0] = true; // Empty Subset to form 0 Target

    for(int i=1; i<=n; i++){
        boolean[] newdp = new boolean[target + 1];

        for(int j=0; j<=target; j++){
            boolean no = dp[j]; // No Call
            boolean yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : false;

            newdp[j] = no || yes;
        }

        dp = newdp;
    }

    return dp[target];
}
```

O(N * target) time
O(target) space

```java
public boolean canPartition(int[] nums) {
    int total = 0;
    for(int val: nums) total += val;

    if(total % 2 == 1) return false; // No Division Possible
    return checkTargetSumSubset(nums, total / 2);
}
```

main logic

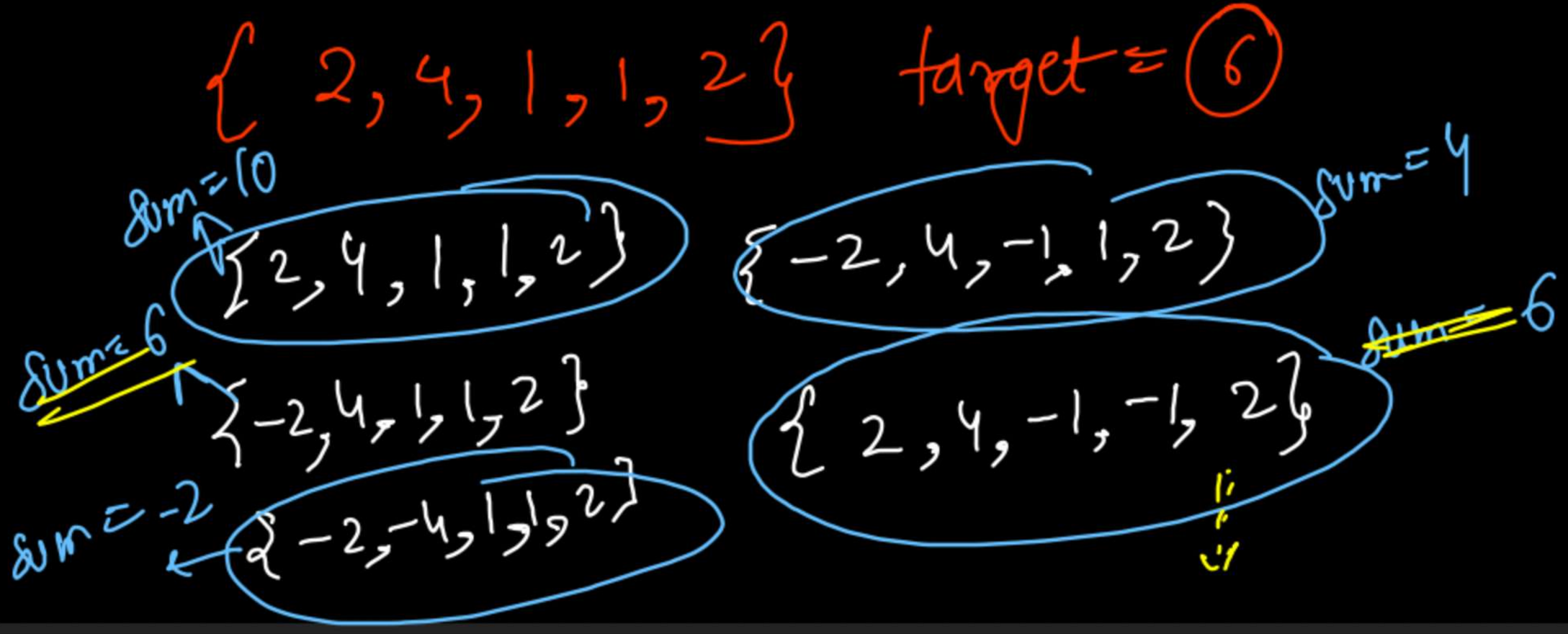
leetcode 494 Target Sum

You are given an integer array `nums` and an integer `target`.

You want to build an **expression** out of nums by adding one of the symbols `'+'` and `'-'` before each integer in nums and then concatenate all the integers.

- For example, if `nums = [2, 1]`, you can add a `'+'` before `2` and a `'-'` before `1` and concatenate them to build the expression `"+2-1"`.

Return the number of different **expressions** that you can build, which evaluates to `target`.



Integer
≥ 0

$\{2, 4, 1, 1, 2\}$ target = 6

Way 1

$\{4, 1, 1, 2\}$ $\{2\}$

Represents subset of elements having + sign

represents subset of elements having -ve sign

$S_1 - S_2$

$= (4+1+1+2) - (2)$

$= 6$

Way 2

$\{2, 4, 2\}$ $\{1, 1\}$

S_1 S_2

$S_1 - S_2$

$2+4+2-1-1$

$= 6$

Divide array into two mutually exclusive and exhaustive (S_1 & S_2) and check if

requirement

$$S_1 - S_2 = target \quad (I)$$

fact

$$S_1 + S_2 = total \quad (II)$$

$$\Rightarrow 2S_2 = total - target$$

$$\Rightarrow S_2 = \frac{total - target}{2}$$

constraints

(1) total − target → even

(2) total ≥ target

Check if there exists a subset with target as S_2

If we can also find $S_1 = \frac{total + target}{2}$

But it will have larger target, hence more time complexity

it will have lesser target, hence lesser TC

```java
public int findTargetSumWays(int[] nums, int target) {
    int total = 0;
    for(int val: nums) total += val;

    if(target > total)
        return 0; // Even if every element is +ve,  S1 - S2 = total

    if((total - target) % 2 == 1)
        return 0;

    return countTargetSumSubset(nums, (total - target) / 2);
}
```

O(N*target) time

O(target) space

eg: {10, 4, 3} target = 2

f − f − f −

{10, 4, 3} 17 {-10, -4, 3} -11

{10, 4, 3} sum = 17

{10, 4, -3} -3 {-10, 4, 3} 9

{10, -4, 3} 9 {-10, -4, -3} -17

{10, 4, -3} 11 {10, -4, -3} -3

{-10, -4, -3} sum = -17

-ve

Dynamic Programming → lecture 16

Target Sum Subset variations

Minimum Difference Subsets! { Tug of War - Different Size }

Print All Paths With Target Sum Subset DP

K-partitions

Partition Into Subsets

Tug Of War - Same Size

Minimize Difference Subsets.

Given an integer array **A** containing **N** integers.

You need to divide the array **A** into two subsets S1 and S2 such that the absolute difference between their sums is minimum.

Find and return this **minimum possible absolute difference**.

NOTE:

- Subsets can contain elements from A in any order (not necessary to be contiguous).
- Each element of A should belong to any one subset S1 or S2, not both.
- It may be possible that one subset remains empty.

Eg

{ 4, 7, 10, 13, 16, 20 }

$|S_1 - S_2|$ = minimum

{Subarray} {subset}

(mutually exclusive & exhaustive)

S_1 {4, 7, 10, 13} 34

S_2 {16, 20} 36

$|S_1 - S_2| = |34 - 36| = 2$

S_1 {4, 20, 10} 34

S_2 {7, 13, 16} 36

$|S_1 - S_2| = |34 - 36| = 2$

eg: {4, 7, 10, 13, 15, 20} total = 72

① $S_1 - S_2$ = min diff → S_1 is big, S_2 is smaller

② $S_1 + S_2$ = total

$\Rightarrow S_1 = S_2 = \frac{\text{total}}{2}$

We have the closest Target sum subset to (total/2)

{2, 4, 0, 0}

0 1 2 3

4-2 = ②

{4,0} — {2,0} = ②

S_1 S_2

(Sum)

	0	1	2	3	4	5	6 ← total sum
0 ∝	1 {}	0	0	0	0	0	0
1 [2]	1 {}	0	1 {2}	0	0	0	0
Heads 2 [4]	1 {}	0	1 {2}	0	1 {4}	0	1 {2,4}
3 [0]	2 {} {}	0	2 {2} {2,0}	0	2 {4} {4,0}	0	2 {2,4} {2,4} {4,0}
4 [0]	4 {} {} {}	0	4 {2} {2,0} {2,0}	0	4 {4} {4,0} {2,4,0}	0	4 {2,4} {2,4} {4,0} {2,4,0} ...

Row by Row Top to Down

```java
int total = 0;
for(int val: arr) total += val;

boolean[] dp = new boolean[total + 1];
dp[0] = true; // Empty Subset to form 0 total

for(int i=1; i<=n; i++){
    boolean[] newdp = new boolean[total + 1];

    for(int j=0; j<=total; j++){
        boolean no = dp[j]; // No Call
        boolean yes = (j >= arr[i - 1]) ? dp[j - arr[i - 1]] : false;

        newdp[j] = no || yes;
    }

    dp = newdp;
}
```

total = 6 (6+1)/2 = 3

$S_1 - S_2$

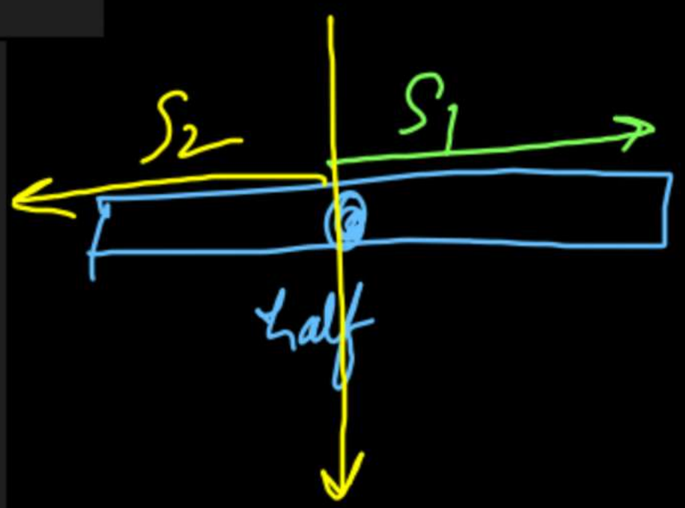
$3 - 3 = 0$

total = 7 (7+1)/2

$S_1 - S_2$

$4 - 3 =$

```java
int half = (total + 1) / 2;
for(int s1=half; s1<=total; s1++){
    if(dp[s1] == true){
        return (s1 - (total - s1));
    }
}
return total;
```



Print All Subsets with given target

Recursion → $O(2^N)$

Using DP (Tabulation) → BFS / DFS

→ Worst case → $O(2^N)$

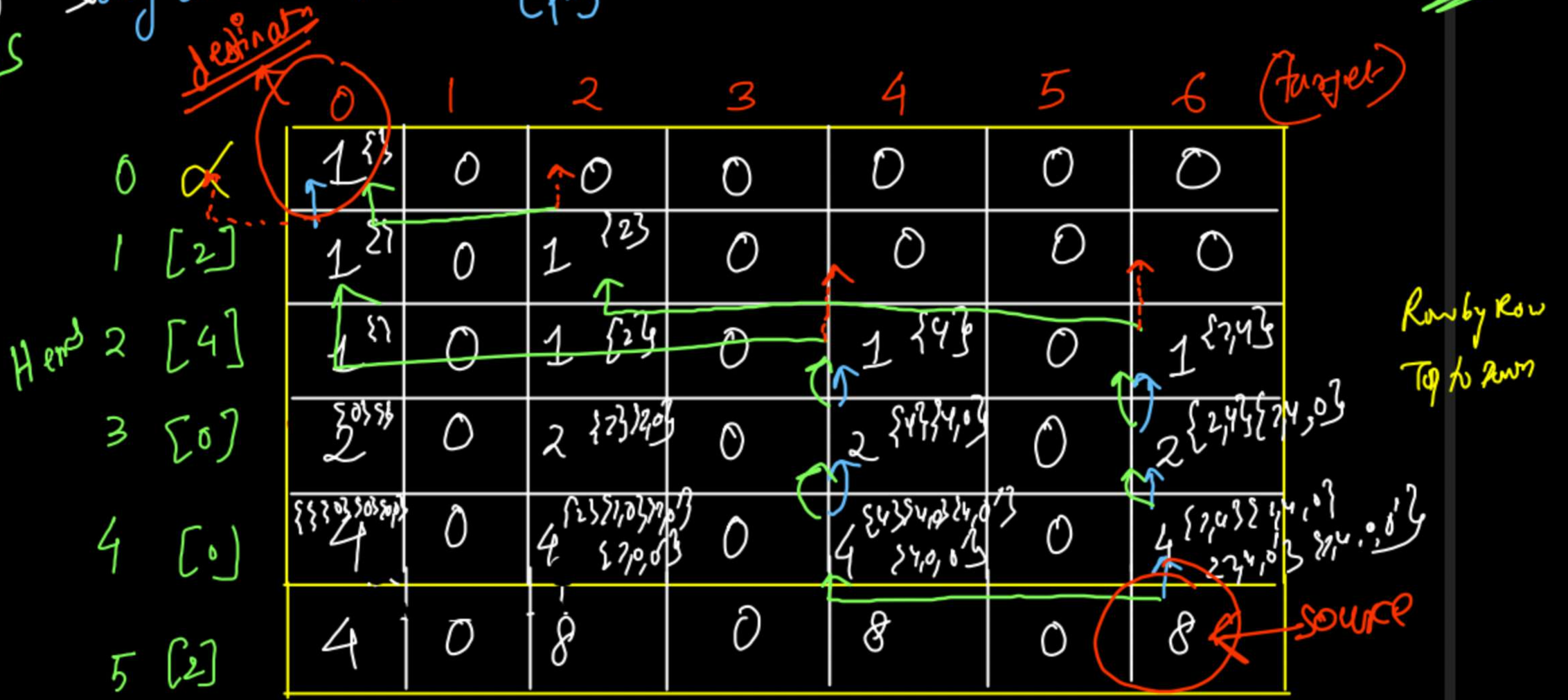
→ Avg case → less than exponential (polynomial)

(2,4) (2,4,0) (2,4,0,0) (2,4,0)

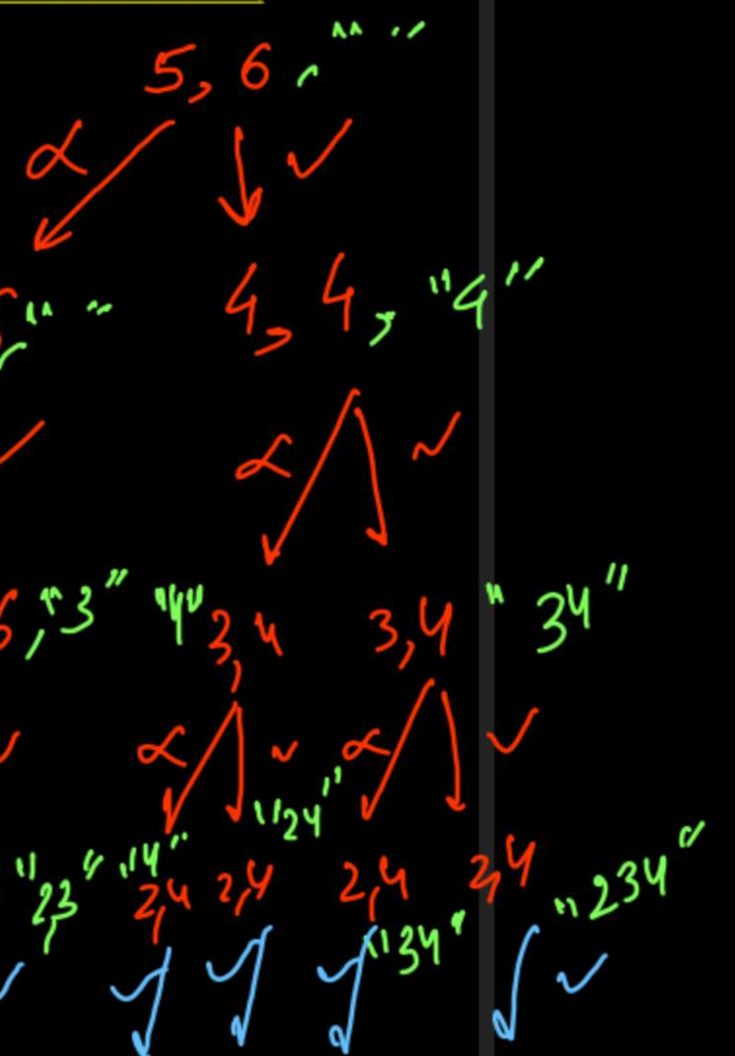
(4,2) (4,2,0) (4,2,0,0) (4,2,0)

BFS

queue



```java
Queue<Pair> q = new ArrayDeque<>();
q.add(new Pair(n, target, ""));

while(q.size() > 0){
    Pair curr = q.remove();
    if(curr.row == 0){
        System.out.println(curr.psf);
        continue;
    }


    int row = curr.row;
    int item = arr[row - 1];
    int col = curr.col;

    // NO
    if(dp[row - 1][col] > 0){
        q.add(new Pair(row - 1, col, curr.psf));
    }

    // YES
    if(col >= item && dp[row - 1][col - item] > 0){
        q.add(new Pair(row - 1, col - item, (row - 1) + " " + curr.psf));
    }

}
}
```



```java
public static void DFS(int row, int col, String psf, int[] arr, int[][] dp){
    if(row == 0){
        System.out.println(psf);
        return;
    }

    int item = arr[row - 1];

    if(dp[row - 1][col] > 0){
        DFS(row - 1, col, psf, arr, dp);
    }


    if(col >= item && dp[row - 1][col - item] > 0){
        DFS(row - 1, col - item, (row - 1) + " " + psf, arr, dp);
    }
}
```

```java
DFS(n, target, "", arr, dp);
```

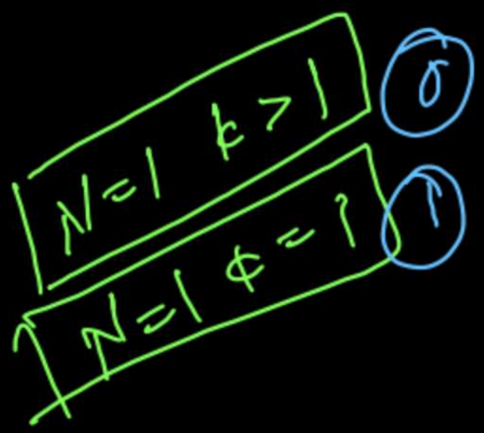
Print All ways to partition {Recursion} n elements into k non-empty subsets

No permutations should be printed

eg1 N=1, k=1

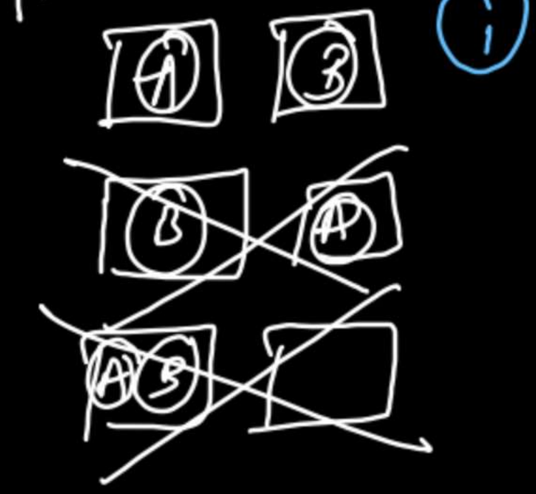
N=1, k=2

N=1 k=3



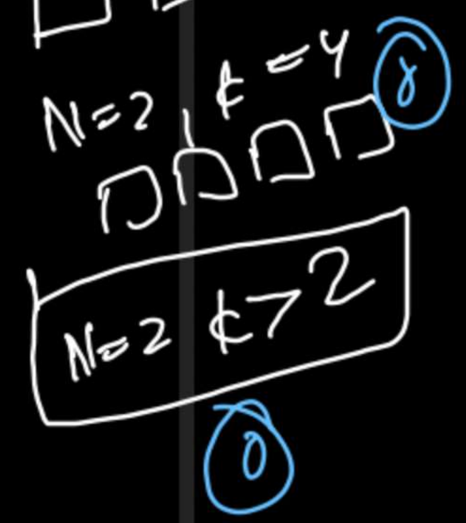
eg2 N=2 k=1

N=2 k=2



N=2, k=3

N=2, k=4



N=3

⇒ k=1

[A,B,C] (1)

⇒ k=2

[A] [B,C]

[A,B] [C] (3)

[A,C] [B]

[A,C]

[B] [AC] [BA] [C]

⇒ k=3 (1)

[A] [B] [C]

⇒ k>3 (4,5,6...)

(0)

N=4 k=1 [A,B,C] (1)

k=2 (7)

[A] [BCD]

[AB] [CD]

[AC] [BD]

[AD] [DC]

[ABC] [D]

[ABD] [C]

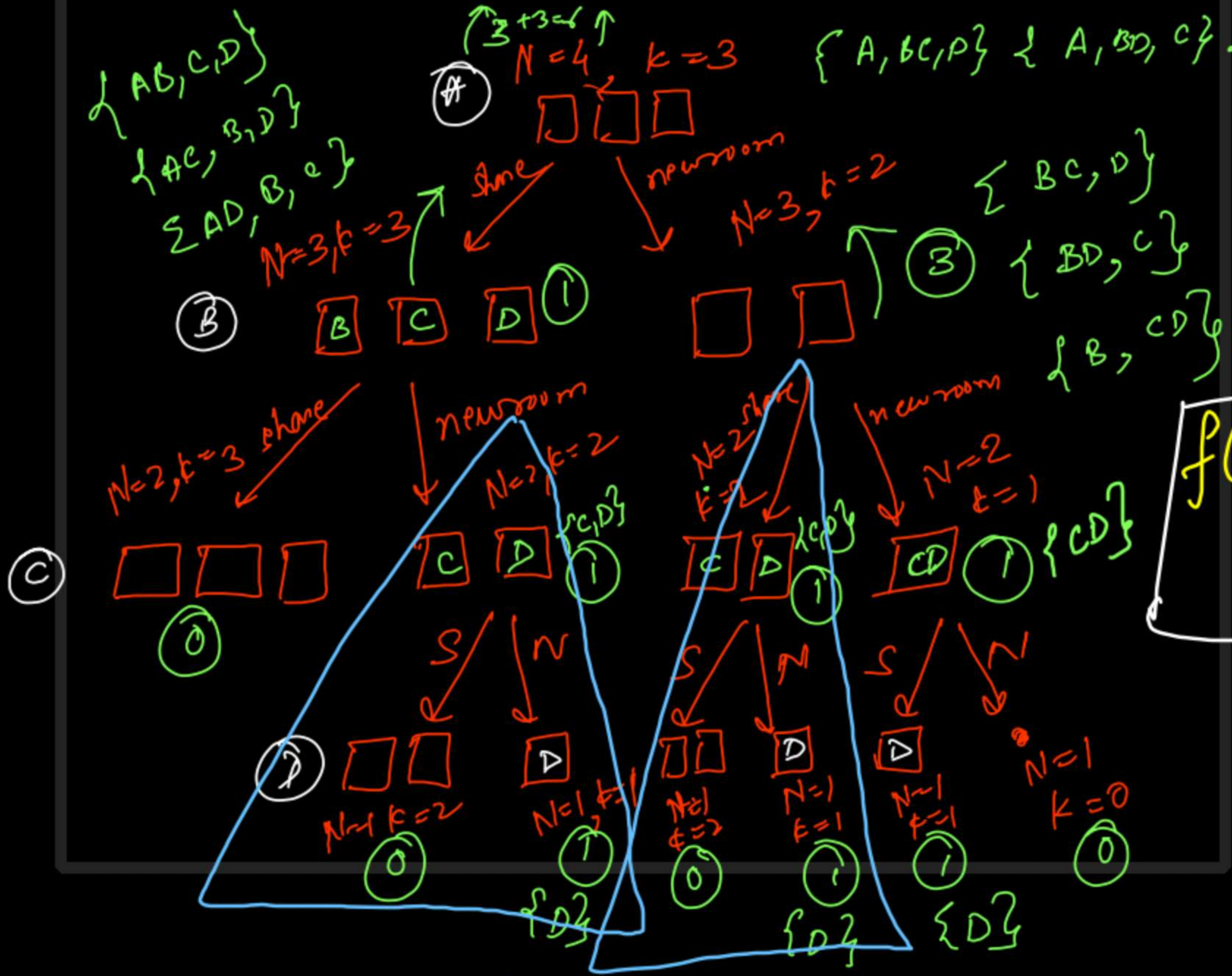
[ACD] [B]

k=4 (1)

[A] [B] [C] [D]

k>5

(0)



DP State
→ [N, K]

Recurrence Relation

$$f(N,K) = f(N-1,K) * K + f(N-1,K-1)$$

⇒ overlapping subproblems.

```java
// All Ways -> All Rooms -> All Persons
public static ArrayList<ArrayList<ArrayList<Integer>>> solution(int currPerson, int n, int k) {
    ArrayList<ArrayList<ArrayList<Integer>>> resWays = new ArrayList<>();

    if(n < k || k <= 0){
        return resWays;
    }

    if(currPerson == n){
        if(k == 1){
            resWays.add(new ArrayList<>());
            resWays.get(0).add(new ArrayList<>());
            resWays.get(0).get(0).add(currPerson);
        }
        return resWays;
    }
```

way → room → person

```java
// Share with Remaining People
ArrayList<ArrayList<ArrayList<Integer>>> ways1 = solution(currPerson + 1, n, k);

for(ArrayList<ArrayList<Integer>> way: ways1){
    for(int i=0; i<way.size(); i++){
        ArrayList<ArrayList<Integer>> newWay = deepCopy(way);
        newWay.get(i).add(0, currPerson);
        resWays.add(newWay);
    }
}
```

← way
← each room
← add person

```java
    // Private New Room
    ArrayList<ArrayList<ArrayList<Integer>>> ways2 = solution(currPerson + 1, n, k - 1);

    for(ArrayList<ArrayList<Integer>> way: ways2){
        ArrayList<ArrayList<Integer>> newWay = deepCopy(way);
        ArrayList<Integer> newRoom = new ArrayList<>();
        newRoom.add(currPerson);
        newWay.add(0, newRoom);
        resWays.add(newWay);
    }

    return resWays;

}
```

← new room
← add person
↑ add room in the way

Count ways to Partition N people in K non-empty rooms

```java
public static long partitionKSubset(int n, int k, long[][] dp) {
    if(n < k || k == 0) return 0l;
    if(n == 1){
        if(k == 1) return 1l;
        return 0l;
    }
    if(dp[n][k] != -1) return dp[n][k];

    long share = partitionKSubset(n - 1, k, dp);
    long newRoom = partitionKSubset(n - 1, k - 1, dp);

    return dp[n][k] = ((share * k) + newRoom);
}
```

Tabular (HW)

→ W/o space optimization
→ O(N*K) time
→ Space → 2D DP

→ With space optimization
→ O(N*K) time
→ Space → O(K) (1D DP)

Time → O(N*K)
Space → O(N) Recursion call stack
O(N*K) 2D DP

Remaining Questions

Tug of war → Same size

Buy & sell stocks → k Transactions

VV IMP

DP on Grids → Dynamic Programming Lecture 17

15th May Sunday, 9 AM to 12 PM

→ Minimum Path Sum → Leetcode 64
→ Print All Paths - NADOS

→ Triangle Path Sum → leetcode 120

→ Falling Path Sum (Goldmine) → leetcode 931

→ Unique Paths → Version I - leetcode 62
→ Version II - leetcode 63

Minimum Path Sum (LC 64)

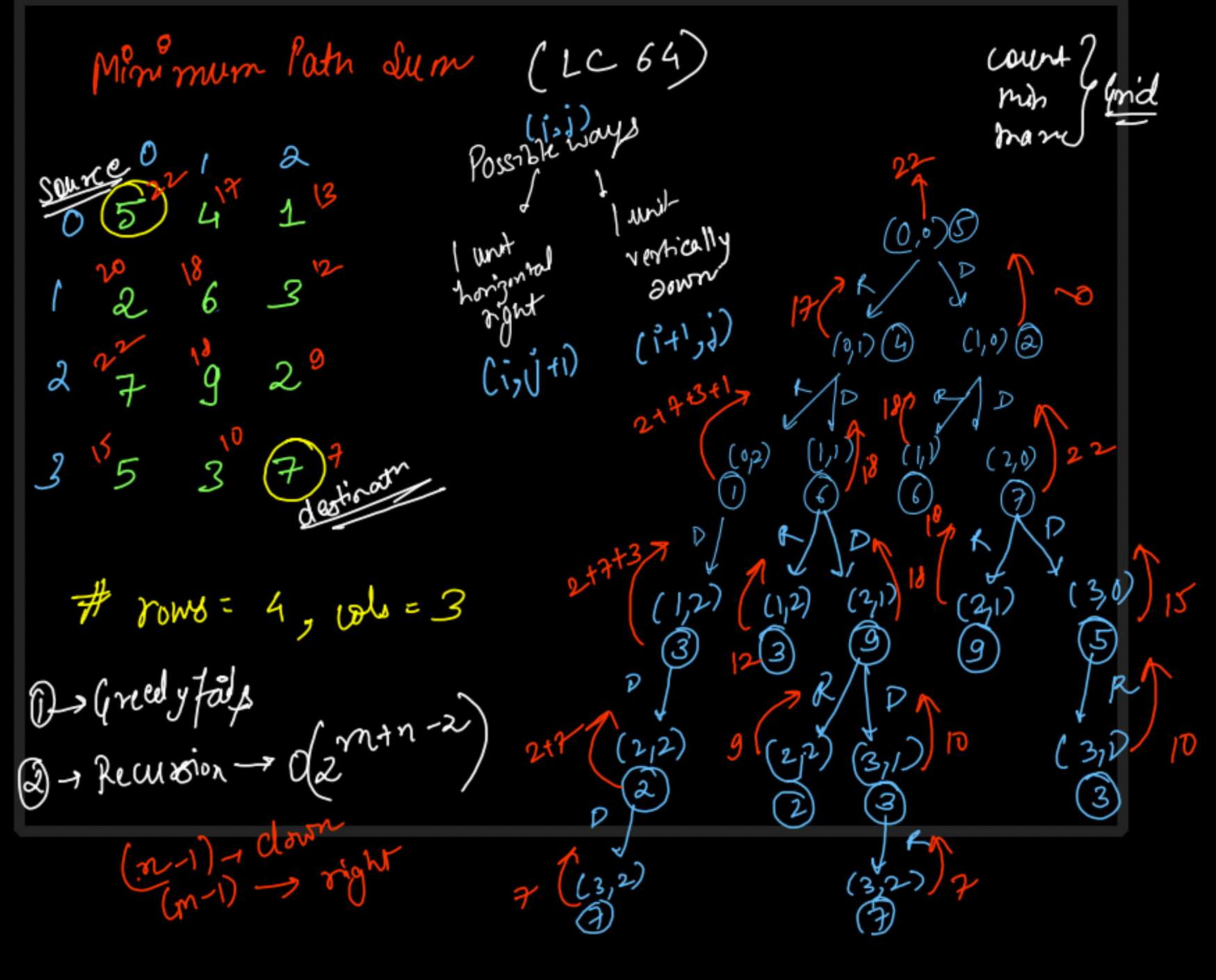
count min max } Grid

	0	1	2
0	5 (22)	4 (17)	1 (13)
1	2 (20)	6 (18)	3 (12)
2	7 (22)	9 (18)	2 (9)
3	5 (15)	3 (10)	7 (7) destination

source

Possible ways (i,j)

- 1 unit horizontal right → (i, j+1)
- 1 unit vertically down → (i+1, j)



rows = 4, cols = 3

① → Greedy fails

② → Recursion → $O(2^{m+n-2})$

(m-1) → down

(n-1) → right

```java
public int helper(int row, int col, int[][] grid, int[][] dp){
    // Negative Base Case: Out of the Grid
    if(row >= grid.length || col >= grid[0].length)
        return Integer.MAX_VALUE;

    // Positive Base Case: Destination is Reached
    if(row == grid.length - 1 && col == grid[0].length - 1)
        return grid[row][col];

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

    int horizontal = helper(row, col + 1, grid, dp);
    int vertical = helper(row + 1, col, grid, dp);

    return dp[row][col] = Math.min(horizontal, vertical) + grid[row][col];
}

public int minPathSum(int[][] grid) {
        int[][] dp = new int[grid.length + 1][grid[0].length + 1];
        for(int i=0; i<=grid.length; i++){
            for(int j=0; j<=grid[0].length; j++){
                dp[i][j] = -1;
            }
        }

        return helper(0, 0, grid, dp);
}
```

Memoization

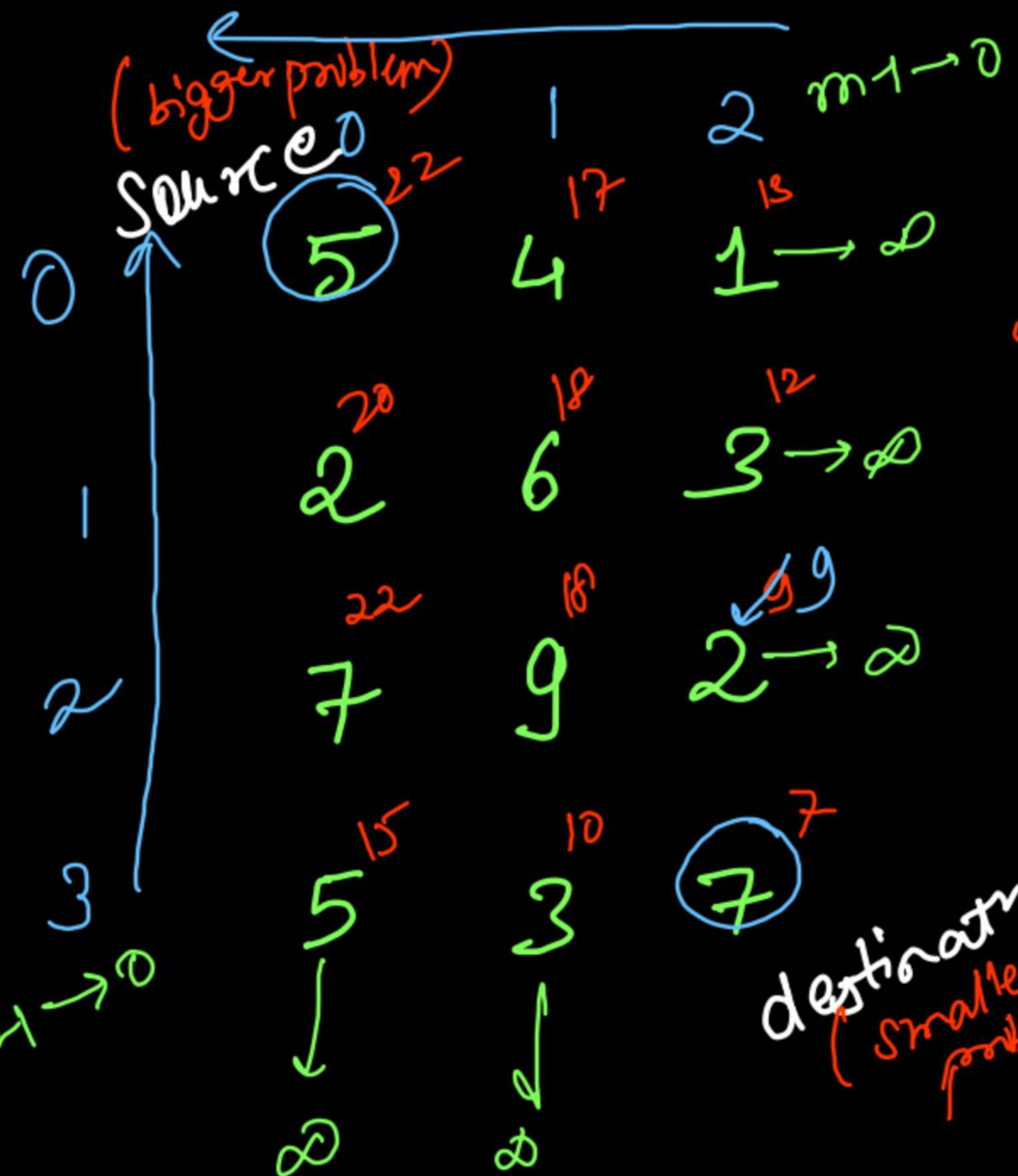
Time → O(N*M)

Space → O(N+M-2) R.C.S.

O(N*M) 2D DP

Tabulatn

Button up



Recurrence Relatn

$$dp[i][j] = \min(dp[i+1][j], dp[i][j+1]) + grid[i][j]$$

```java
public int minPathSum(int[][] grid) {
        int[][] dp = new int[grid.length + 1][grid[0].length + 1];
        for(int i=0; i<=grid.length; i++){
            for(int j=0; j<=grid[0].length; j++){
                dp[i][j] = Integer.MAX_VALUE;
            }
        }

        for(int i=grid.length-1; i>=0; i--){
            for(int j=grid[0].length-1; j>=0; j--){
                if(i == grid.length - 1 && j == grid[0].length - 1){
                    dp[i][j] = grid[i][j];
                    continue;
                }

                dp[i][j] = Math.min(dp[i + 1][j], dp[i][j + 1]) + grid[i][j];
            }
        }

        return dp[0][0];
}
```

Tabulation

O(N*M) time

O(N*M) Space

2D DP

```java
public int minPathSum(int[][] grid) {
        int[] dp = new int[grid[0].length + 1];
        for(int j=0; j<=grid[0].length; j++){
            dp[j] = Integer.MAX_VALUE;
        }

        for(int i=grid.length-1; i>=0; i--){
            for(int j=grid[0].length-1; j>=0; j--){
                if(i == grid.length - 1 && j == grid[0].length - 1){
                    dp[j] = grid[i][j];
                    continue;
                }

                dp[j] = Math.min(dp[j], dp[j + 1]) + grid[i][j];
            }
        }

        return dp[0];
}
```

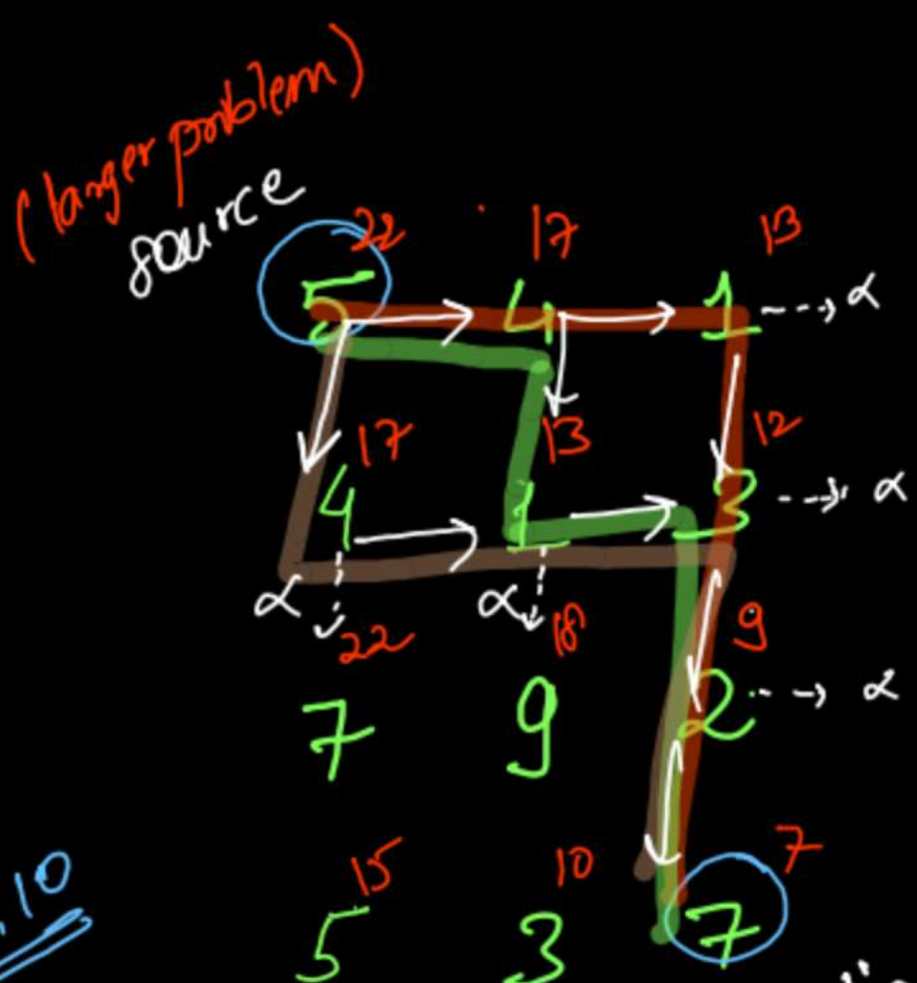
Space Optimization

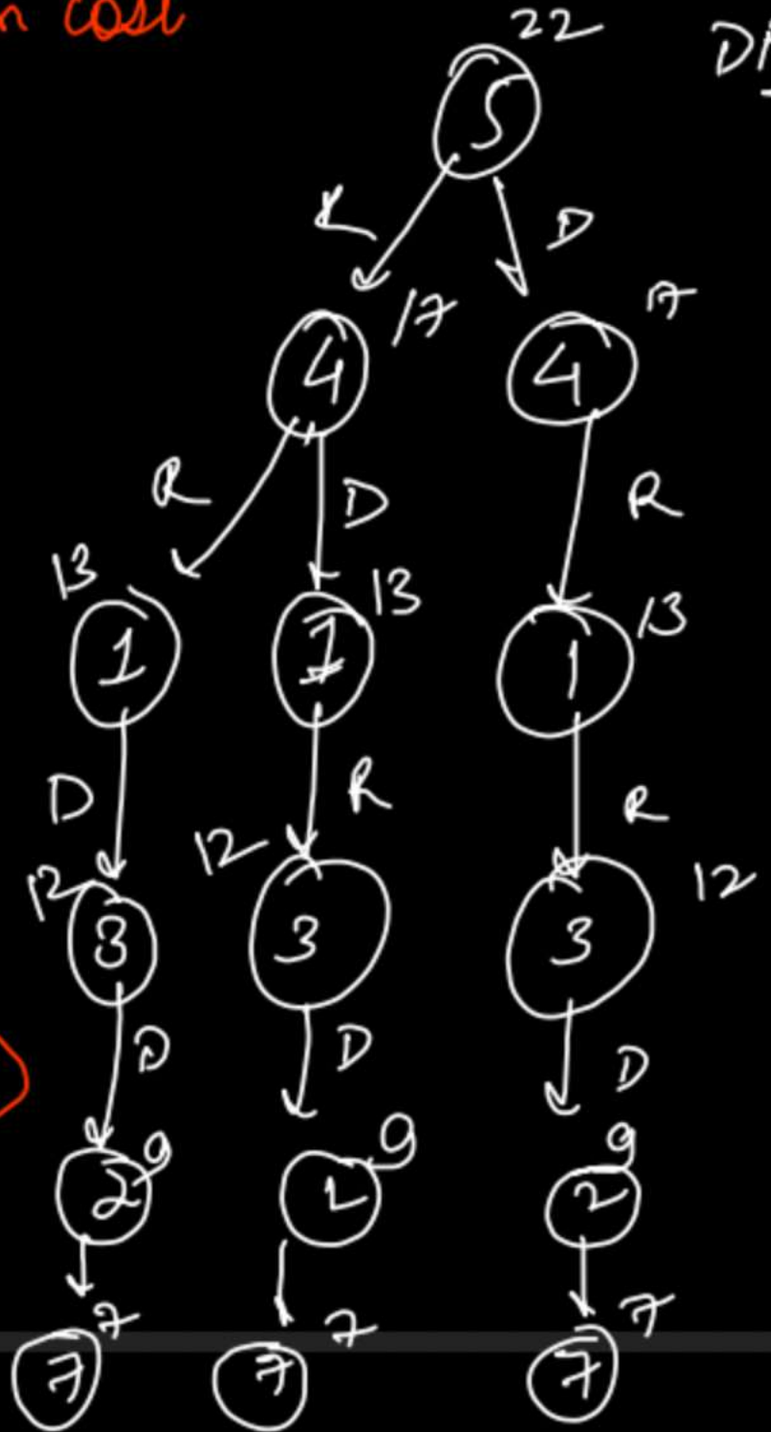
O(N*M) time

O(M) Space

1D DP

Print All paths with min cost





11:10

Time → O(Polynomial)

Space → O(N+M-2) R.C.S

O(N*M) 2D DP table


```java
public static void DFS(int row, int col, int[][] grid, int[][] dp, String psf){
    if(row == grid.length - 1 && col == grid[0].length - 1){
        System.out.println(psf);
        return;
    }

    int min = Math.min(dp[row + 1][col], dp[row][col + 1]);

    if(dp[row + 1][col] == min){
        DFS(row + 1, col, grid, dp, psf + "V");
    }

    if(dp[row][col + 1] == min){
      DFS(row, col + 1, grid, dp, psf + "H");
    }

}
```

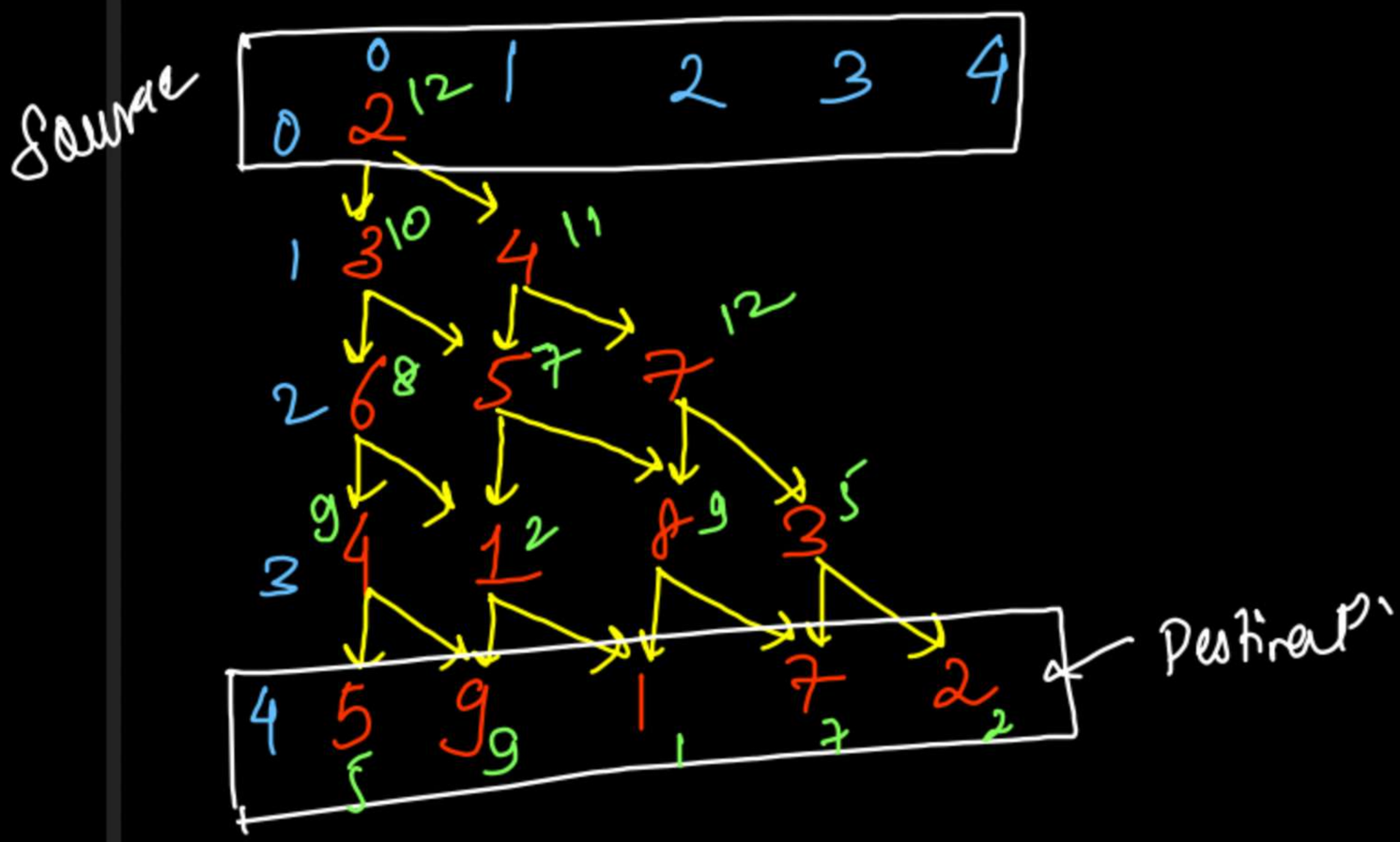
```java
//write your code here
int[][] dp = minPathSum(arr);
System.out.println(dp[0][0]);
DFS(0, 0, arr, dp, "");
```

Triangle (LC 120)

Given a `triangle` array, return *the minimum path sum from top to bottom*.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index `i` on the current row, you may move to either index `i` or index `i + 1` on the next row.

$$dp[i][j] = \min(dp[i+1][j], dp[i+1][j+1]) + grid[i][j]$$



Single Source (0,0)

multiple destination
↳ last row
↳ Base case

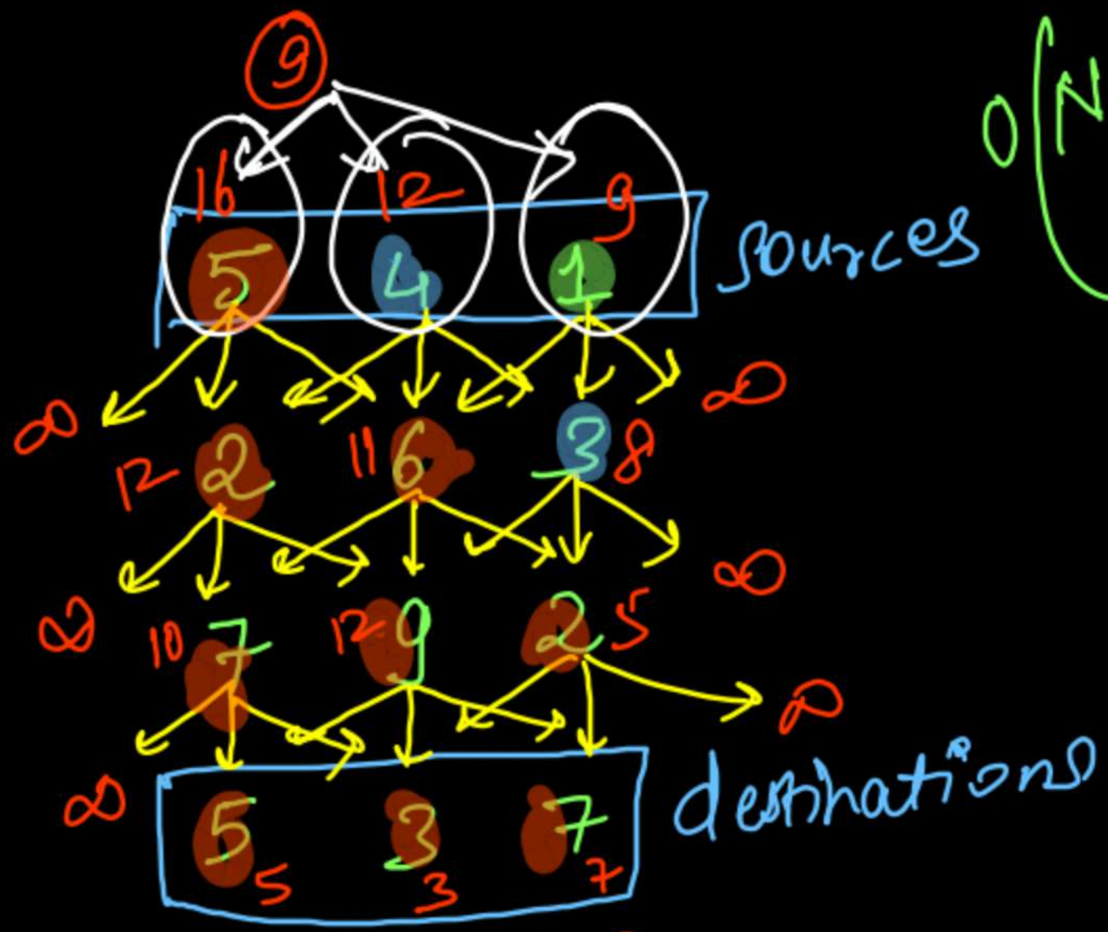
```java
class Solution {
    public int helper(List<List<Integer>> triangle, int row, int col, Integer[][] dp){
        if(row == triangle.size() - 1){
            // Entire Last Row is the Destination
            return triangle.get(row).get(col);
        }

        if(dp[row][col] != null) return dp[row][col];

        int down = helper(triangle, row + 1, col, dp);
        int rightdown = helper(triangle, row + 1, col + 1, dp);

        return dp[row][col] = (triangle.get(row).get(col) + Math.min(down, rightdown));
    }

    public int minimumTotal(List<List<Integer>> triangle) {
        int n = triangle.size();
        Integer[][] dp = new Integer[n + 1][n + 1];
        return helper(triangle, 0, 0, dp);
    }
}
```

Recursion
→ $O(2^{Rows})$ time
→ $O(Rows)$ R.C.S

Memoization
$O(Rows * Rows)$ time
$O(Rows * Rows)$ 2D DP
$O(Rows)$ R.C.S.

Tabulation
→ W/o space opt ↔ $O(R*R)$ time, $O(R*R)$ 2D DP
→ with space opt ↔ $O(R*R)$ time, $O(2*R)$ 2 rows of 1D DP

931 Minimum Falling Path Sum



$O(N*M + M) \approx O(N*M)$

Multiple source → Multiple destination

#Greedy → fail?

Possible moves

- left down → $(i+1, j-1)$
- down → $(i+1, j)$
- right down → $(i+1, j+1)$

```java
public int memo(int row, int col, int[][] grid, int[][] dp){
    if(col < 0 || col >= grid[0].length){
        // Out of Matrix: Negative Base Case
        return Integer.MAX_VALUE;
    }

    if(row == grid.length - 1){
        // Entire last row is the destination
        return grid[row][col];
    }

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

    int leftDown = memo(row + 1, col - 1, grid, dp);
    int down = memo(row + 1, col, grid, dp);
    int rightDown = memo(row + 1, col + 1, grid, dp);

    return dp[row][col] = Math.min(down, Math.min(leftDown, rightDown)) + grid[row][col];
}
```

Multiple destination

DP table → only filled only once

Time → O(N*M)

Space → O(N*M)

2D DP

```java
public int minFallingPathSum(int[][] grid) {
    int[][] dp = new int[grid.length + 1][grid[0].length + 1];
    for(int i=0; i<dp.length; i++){
        for(int j=0; j<dp[0].length; j++){
            dp[i][j] = -1;
        }
    }

    int min = Integer.MAX_VALUE;

    // Call for Each Element of the first row as the source node
    // DP table is going to be filled only once: Overall Time: O(N * M)
    for(int src=0; src<grid[0].length; src++){
        int path = memo(0, src, grid, dp);
        min = Math.min(min, path);
    }

    return min;
}
```

Multiple Sources

Dynamic Programming

Lecture - 18

17th May, 10 PM to 12 AM

62 Unique Paths - 1

Count of ways from source to destination

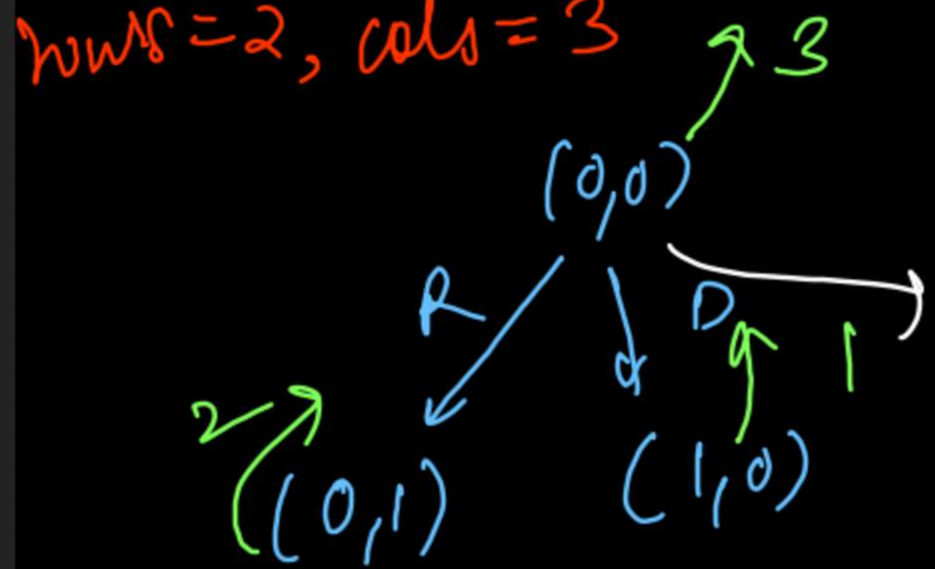
source → top left

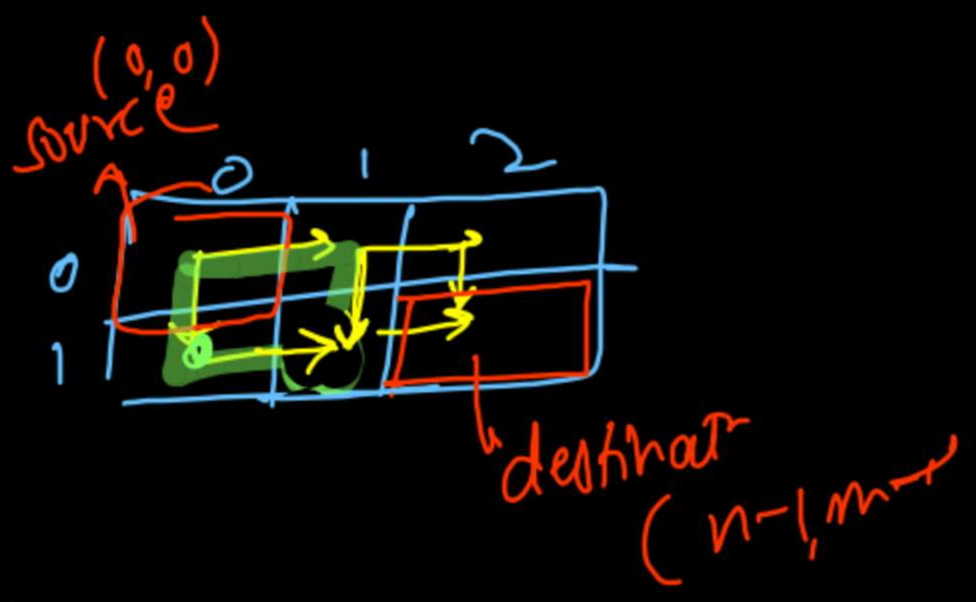
destination → bottom right

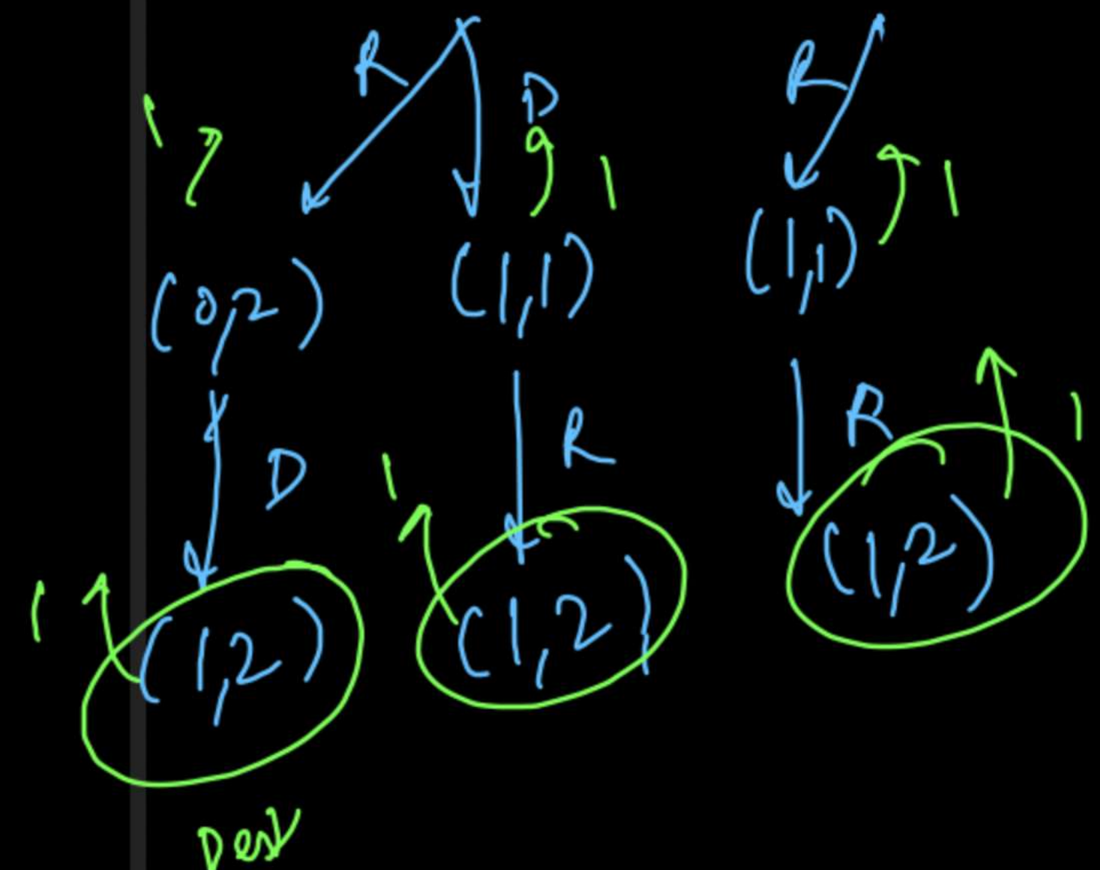
Possible moves → horizontally right

Possible moves → vertically down

rows = 2, cols = 3







Time

Recursion → $O(2^{n+m-2})$ expo

DP → $O(N*M)$ quadratic

RC Combinatorics

RRDDD
RDRDD
RDDRD
RDDDR
DRRDD
DRDRD
DRDDR
DDRRD
DDRDR
DDDRR

10

R → 2 → M-1

D → 3 → N-1

$\frac{(R+D)!}{R!\,D!}$

Rows = 4

Cols = 3

$^{M+N-2}C_{M-1}$

$\frac{(M-1+N-1)!}{(M-1)! * (N-1)!}$

nr of ways to arrange R & D in the string

Time $O(2*(M+N))$

Factorial → Integer overflow

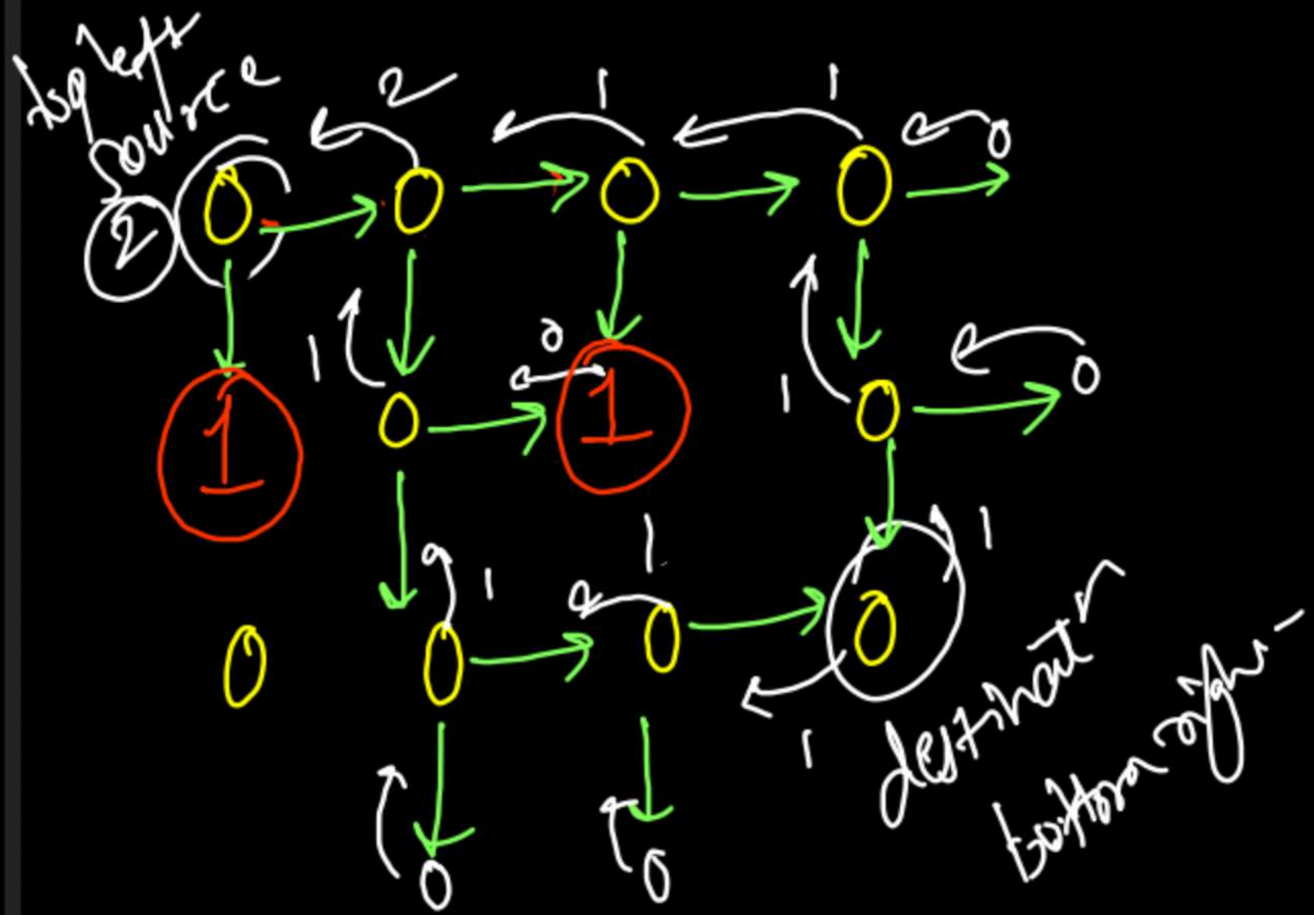
Unique Paths - I

```java
public int memo(int sr, int sc, int dr, int dc, int[][] dp){
    if(sr > dr || sc > dc) return 0;
    if(sr == dr && sc == dc) return 1;
    if(dp[sr][sc] != -1) return dp[sr][sc];

    return dp[sr][sc] = (memo(sr + 1, sc, dr, dc, dp)
                        + memo(sr, sc + 1, dr, dc, dp));
}

public int uniquePaths(int m, int n) {
    int[][] dp = new int[m + 1][n + 1];
    for(int i=0; i<=m; i++)
        for(int j=0; j<=n; j++)
            dp[i][j] = -1;

    return memo(0, 0, m - 1, n - 1, dp);
}
```

Memoization

Time → O(N*M)

Space → O(N*M)



0 → free cell

1 → blockage node

Unique Paths - 11

Possible ways

horizontally → right

vertically → down

blockage node → dest

0 ways

```java
public int memo(int sr, int sc, int dr, int dc, int[][] grid, int[][] dp){
    if(sr > dr || sc > dc) return 0;
    if(grid[sr][sc] == 1) return 0;
    if(sr == dr && sc == dc) return 1;
    if(dp[sr][sc] != -1) return dp[sr][sc];

    return dp[sr][sc] = (memo(sr + 1, sc, dr, dc, grid, dp)
                        + memo(sr, sc + 1, dr, dc, grid, dp));
}

public int uniquePathsWithObstacles(int[][] grid) {
    // Source is blocked
    if(grid[0][0] == 1) return 0;

    int m = grid.length, n = grid[0].length;
    // Destination is blocked
    if(grid[m - 1][n - 1] == 1)
        return 0;

    int[][] dp = new int[m + 1][n + 1];
    for(int i=0; i<=m; i++)
        for(int j=0; j<=n; j++)
            dp[i][j] = -1;

    return memo(0, 0, m - 1, n - 1, grid, dp);
}
```

Time → O(M*N)

Space → O(M*N)
(2D DP)

GFG Keypad Problem

N = numbers of keys to be pressed

N = 1

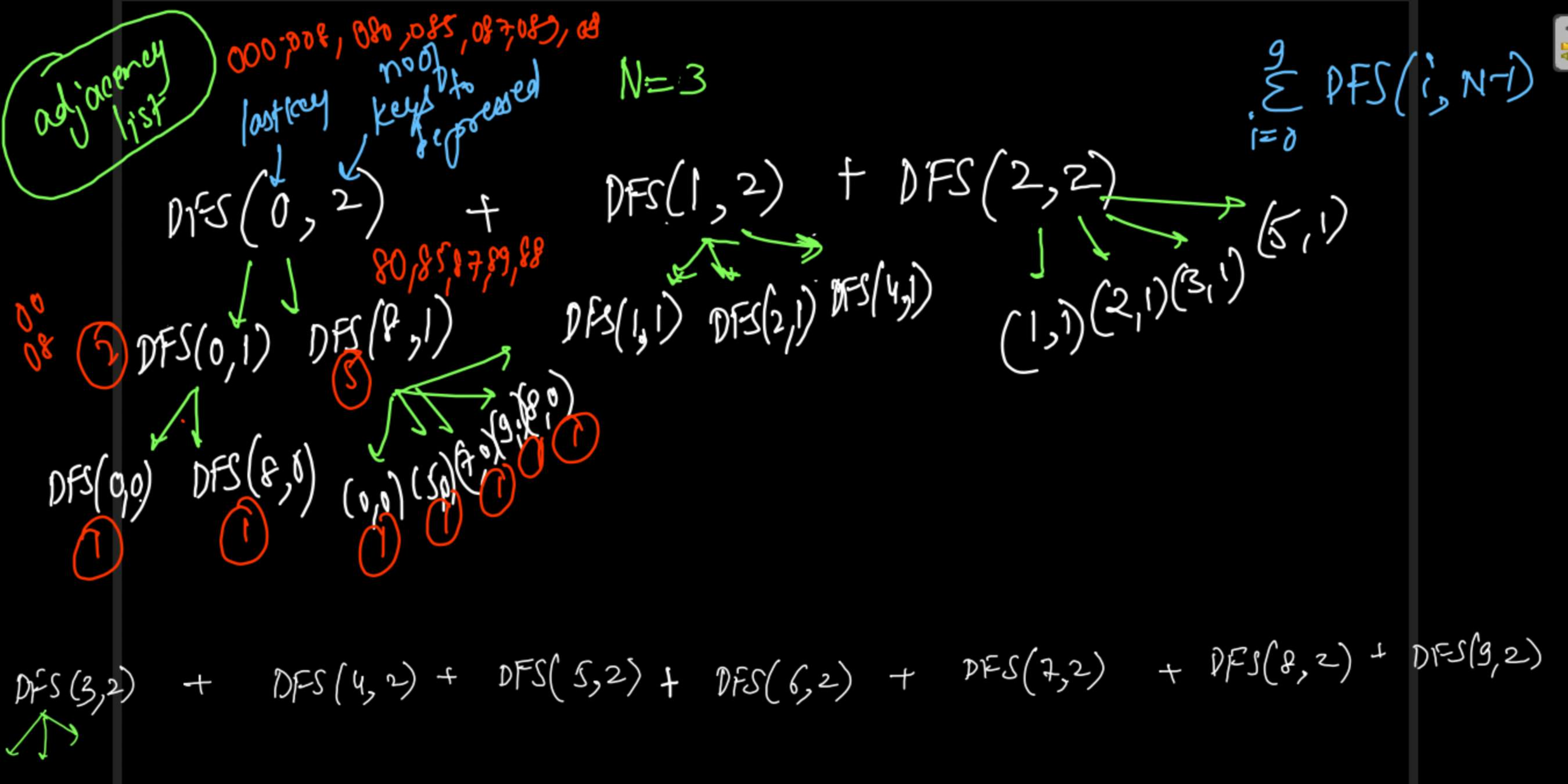
1, 2, 3, 4, 5, 6, 7, 8, 9, 0 → (10) ways

N = 2

→ (36) ways

11, 12, 14, 22, 21, 23, 25,
33, 32, 36, 44, 41, 45, 47, 55, 52, 54,
56, 58, 66, 63, 65, 69, 77,
79, 78, 88, 85, 87, 89, 80
99, 96, 98, 00, 08

After pressing any key
the next move is → pressing the same key again
→ press any one of its 4 neighbours (T, L, D, R)


adjacency list
000;008, 080, 085, 087, 089, 88
last key
no of keys to be processed
N=3
$\sum_{i=0}^{9} DFS(i, N-1)$
DFS(0, 2) + DFS(1, 2) + DFS(2, 2)
80, 85, 87, 89, 88
DFS(0,1) DFS(8,1)
DFS(1,1) DFS(2,1) DFS(4,1)
(1,1) (2,1) (3,1) (5,1)
DFS(0,0) DFS(8,0)
(0,0) (5,0) (7,0) (9,0) (8,0)
DFS(3,2) + DFS(4,2) + DFS(5,2) + DFS(6,2) + DFS(7,2) + DFS(8,2) + DFS(9,2)

Recursion

Worst case → $O(5^N)$

DP

(lastKey, keys)

(0-9) (0-N-1)

$O(N * 10)$

space ↳ $O(N*10)$ DP table

```java
// For Each Key, store the neighbouring keys and the same key itself.
int[][] adj = {{0, 8}, {1, 2, 4}, {1, 2, 3, 5}, {2, 3, 6}, {1, 4, 5, 7},
        {2, 4, 5, 6, 8}, {3, 5, 6, 9}, {4, 7, 8}, {0, 5, 7, 8, 9}, {6, 8, 9}};

public long DFS(int lastKey, int keys, long[][] dp){
    if(keys == 0) return 1l;
    if(dp[lastKey][keys] != -1l) return dp[lastKey][keys];

    long ans = 0;
    for(int nbr: adj[lastKey]){
        ans = ans + DFS(nbr, keys - 1, dp);
    }

    return dp[lastKey][keys] = ans;
}
```

```java
public long getCount(int N)
{
    long[][] dp = new long[10][N];
    for(int i=0; i<10; i++){
        for(int j=0; j<N; j++){
            dp[i][j] = -1;
        }
    }

    long ans = 0;
    for(int i=0; i<10; i++){
        ans = ans + DFS(i, N - 1, dp);
    }

    return ans;
}
```

Tug of War → Equal size

$S_1 - S_2$ → minimum diff

mutually exclusive
exhaustive
Equal size

1. You are given an array of n integers.
2. You have to divide these n integers into 2 subsets such that the difference of sum of two subsets is as minimum as possible.
3. If n is even, both set will contain exactly n/2 elements. If is odd, one set will contain (n-1)/2 and other set will contain (n+1)/2 elements.
3. If it is not possible to divide, then print "-1".

Target Sum Subset

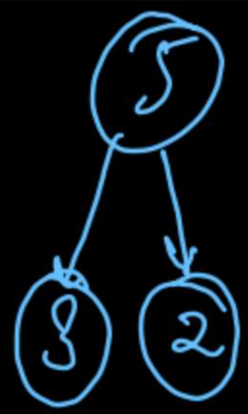
{ 1, 2, 3, 10, 16 }

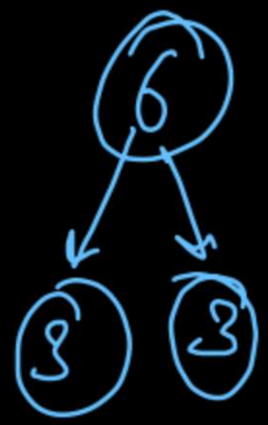
Diff Size

$S_1 - S_2 = \min \text{diff}$

{1, 2, 3, 10} {16}

16 − 16 = 0





{1, 2, 3, 10}

0 1 0 1 0 1 0 1

Recursion tree

size sum idx

subset

- $N \leq 20 \longrightarrow$ Recursion $\rightarrow$ Exponential time
- $N \leq 10^3 \rightarrow O(N^3)$ $\rightleftharpoons$ Floyd warshall, Bellman ford — 3D DP
- $N \leq 10^4 \rightarrow O(N^2)$ $\rightleftharpoons$ Dijkstra, Prim's, Kruskal's — 2D DP
- $N \leq 10^6 \rightarrow O(N \log N) \longrightarrow$ Sorting (merge/quick)
- $N \leq 10^8 \rightarrow O(N) \longrightarrow$ DFS, BFS — 1D DP

$O(2^N)$ Time Complexity

```
        helper(arr, 0, "", 0, 0, "", 0, 0);
        System.out.println("[" + s1Ans + "] [" + s2Ans + "]");
    }

    static int mindiff = Integer.MAX_VALUE;
    static String s1Ans = "", s2Ans = "";


    public static void helper(int[] arr, int idx, String s1, int s1Sum, int s1Len, String s2, int s2Sum, int s2Len){
        if(idx == arr.length){
            int diff = Math.abs(s1Sum - s2Sum);

            if(arr.length % 2 == 1 && (s1Len == s2Len + 1 || s2Len == s1Len + 1) && diff < mindiff){
                s1Ans = s1.substring(2); s2Ans = s2.substring(2);
                mindiff = diff;
            }

            else if(arr.length % 2 == 0 && s1Len == s2Len && diff < mindiff){
                s1Ans = s1.substring(2); s2Ans = s2.substring(2);
                mindiff = diff;
            }

            return;
        }

        // Element is inserted in S1
        helper(arr, idx + 1, s1 + ", " + arr[idx], s1Sum + arr[idx], s1Len + 1, s2, s2Sum, s2Len);

        // Element in inserted in S2
        helper(arr, idx + 1, s1, s1Sum, s1Len, s2 + ", " + arr[idx], s2Sum + arr[idx], s2Len + 1);
    }
```

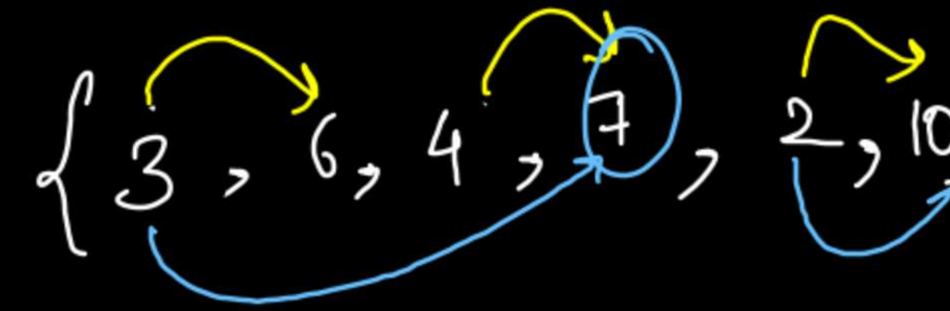
↳ to remove the leading space & zero

① equal length
② min diff
③ MS & E

B&S Study Striver

Buy & Sell Stocks → K Transactions

→ Leetcode 188



K = 3 transactions ⇒ (6-3) + (7-4) + (10-2) = 3+3+8 = 14

K = 2 transactions ⇒ (7-3) + (10-2) = 4+8 = 12

k = 1 transaction ⇒ (10-2) = 8

$$dp[i][t] = \max_{j<i} dp[j][t-1] + (arr[i] - arr[j])$$

Stocks = {3, 6, 4, 7, 2, 10}

(3) (6) (4) (7) (2) (10)

0	0	0	0	0	0	0
1	3-3 = 0	6-3 = 3	4-3 vs 4-6 vs 6-3 = 3	7-4 vs 7-6 vs 7-3 = 4	2-2 = 0 vs 4	10-2 = 8
2	0	6-3 = 3	4-6+3 vs 4-3+0 vs 6-3 = 3	7-4+(6-3) = 6	6	8
3	0	3	3	6	6	10-2+6 = 14

Overlapping transactions cannot occur

Atmost 2 transactions

10-2+6
10-7+6
10-4+3
10-6+3
R=3

2-7+4=-1
2-4+3=1
2-6+3=-1
2-3+0

10-2+4
10-7+4
10-4+3
10-6+3
10-3+0

2-7+6
2-4+3
2-6+3
2-3+0

7-4+3 vs 7-6+3 vs 7-3+0

dp[t][i] = Atmost T transactions upto ith day & max^m profit

```java
class Solution {
    public int maxProfit(int k, int[] prices) {
        if(prices.length == 0 || k == 0) return 0;

        int[][] dp = new int[k + 1][prices.length];

        for(int t=1; t<=k; t++){
            for(int i=0; i<prices.length; i++){
                dp[t][i] = (i - 1 >= 0) ? dp[t][i - 1] : 0;

                for(int j=i-1; j>=0; j--){
                    // Last Transaction is between j and i, and remaining (t - 1)
                    // transactions are before the jth day (Non-overlapping).

                    dp[t][i] = Math.max(dp[t][i], (prices[i] - prices[j]) + dp[t - 1][j]);
                }
            }
        }

        return dp[k][prices.length - 1];
    }
}
```

#Time → O(K * N * N)

DP on Grids

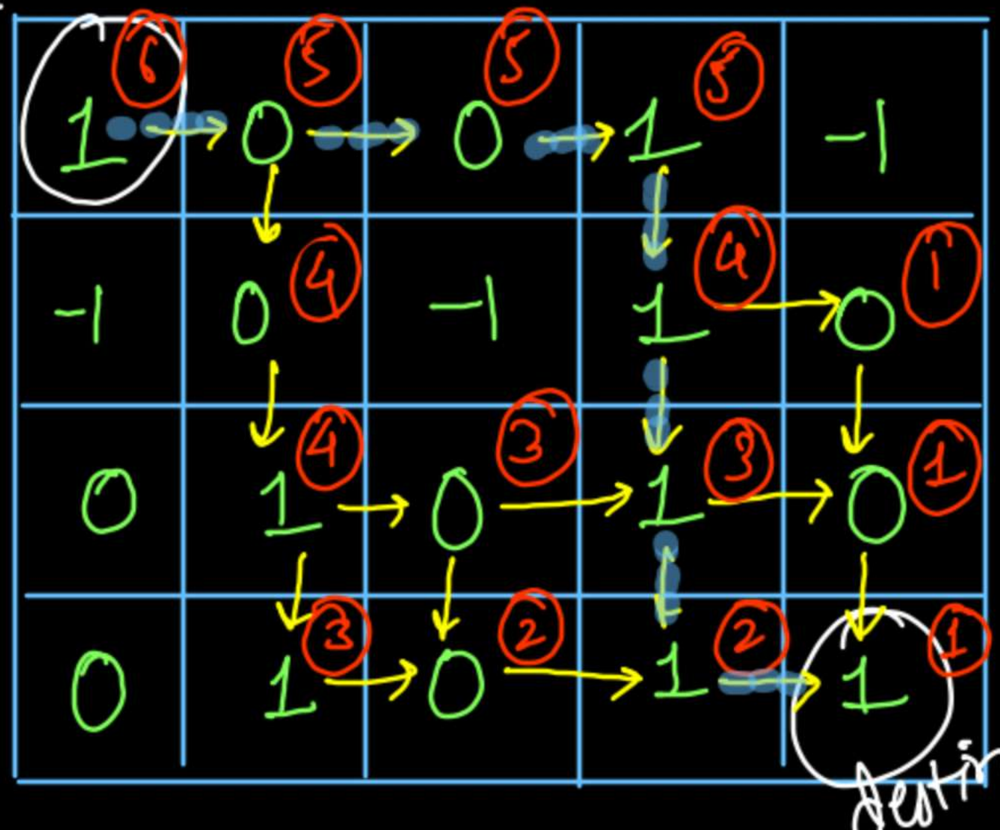
Maxm path sum with Blockage

Single source → Top left Single Dest → Bottom Right.

Possible moves → Right or Down

src → intermediate → dp[in] - destination

source



0 → empty cell

1 → cherry

-1 → blockage

DP → (Row, Col)

```java
public int helper(int row, int col, int[][] grid, int[][] dp){
    if(row >= grid.length || col >= grid[0].length || grid[row][col] == -1){
       // Negative Base Case (Out of Matrix or Blockage Node)
       return 0;
    }

    if(row == grid.length - 1 && col == grid[0].length - 1){
        // Positive Base Case: Destination Cell
        return grid[row][col];
    }

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

    int right = helper(row, col + 1, grid, dp);
    int down = helper(row + 1, col, grid, dp);

    return dp[row][col] =  Math.max(right, down) + grid[row][col];
}
```

```java
int[][] dp = new int[grid.length][grid[0].length];
for(int i=0; i<dp.length; i++){
    for(int j=0; j<dp[0].length; j++){
        dp[i][j] = -1;
    }
}

return helper(0, 0, grid, dp);
```

Time → O(N*M)

2D DP

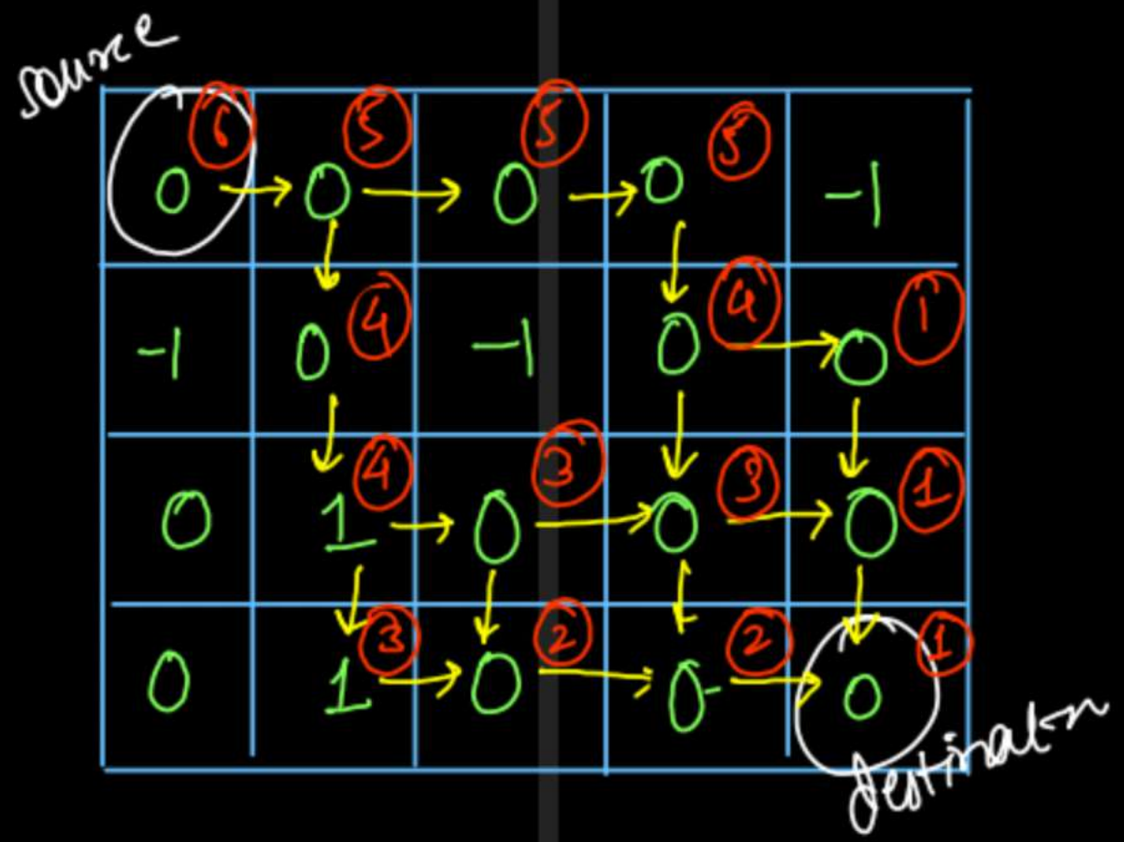
Space → O(N*M)

Cherry Pickup - I

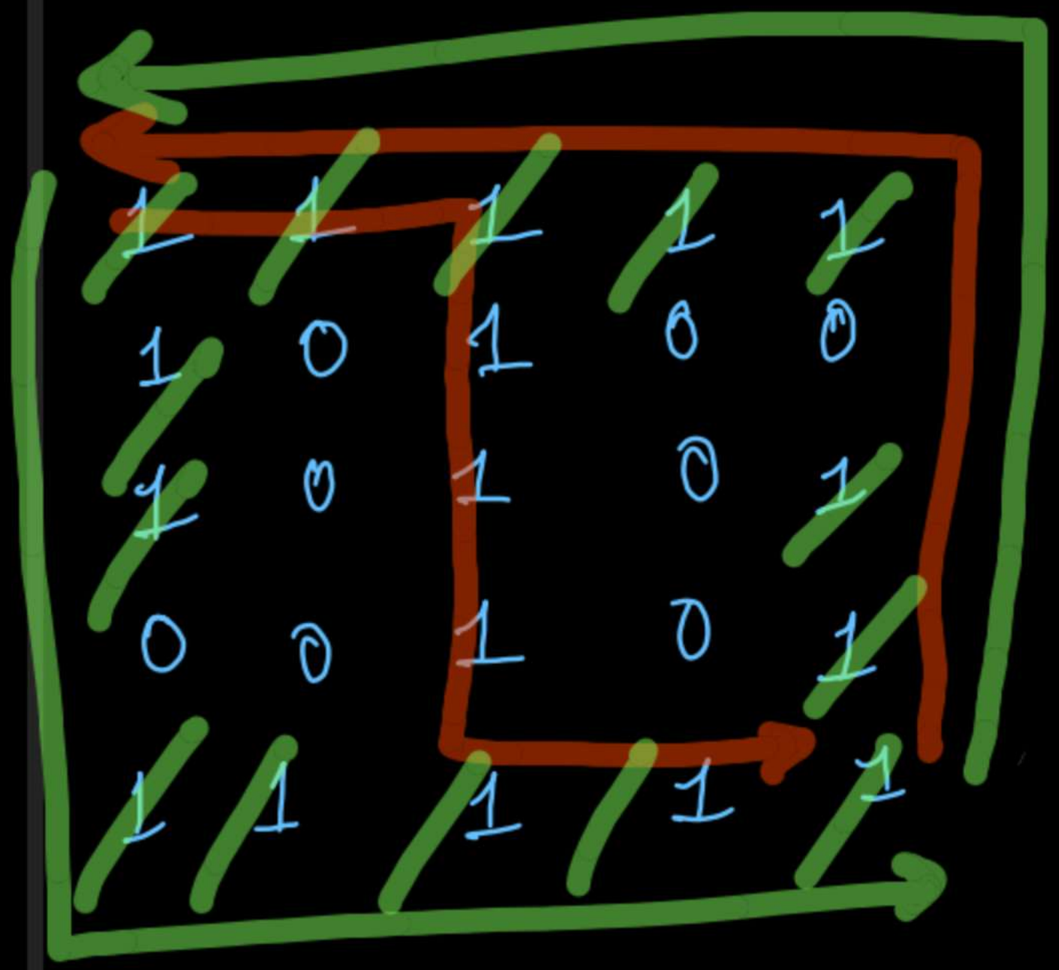
Leetcode 741

I^st iteration → Source → Dest
Top left → Bottom right

II^nd iteration → Dest → Source
Bottom right → Top left



Both iterations are not independent, rather they are inter-related



your answer

TL → BR + BR → TL

9 + 4 = 13

expected output

TL → BR + BR → TL = 14

8 + 6

P_1 & P_2 destination

Single source, single dest two simultaneous paths

P_1 P_2

source

1	1	1	1	1
1	0	1	0	0
1	-1	1	-1	1
0	0	1	0	1
1	1	1	1	1

1 → dest

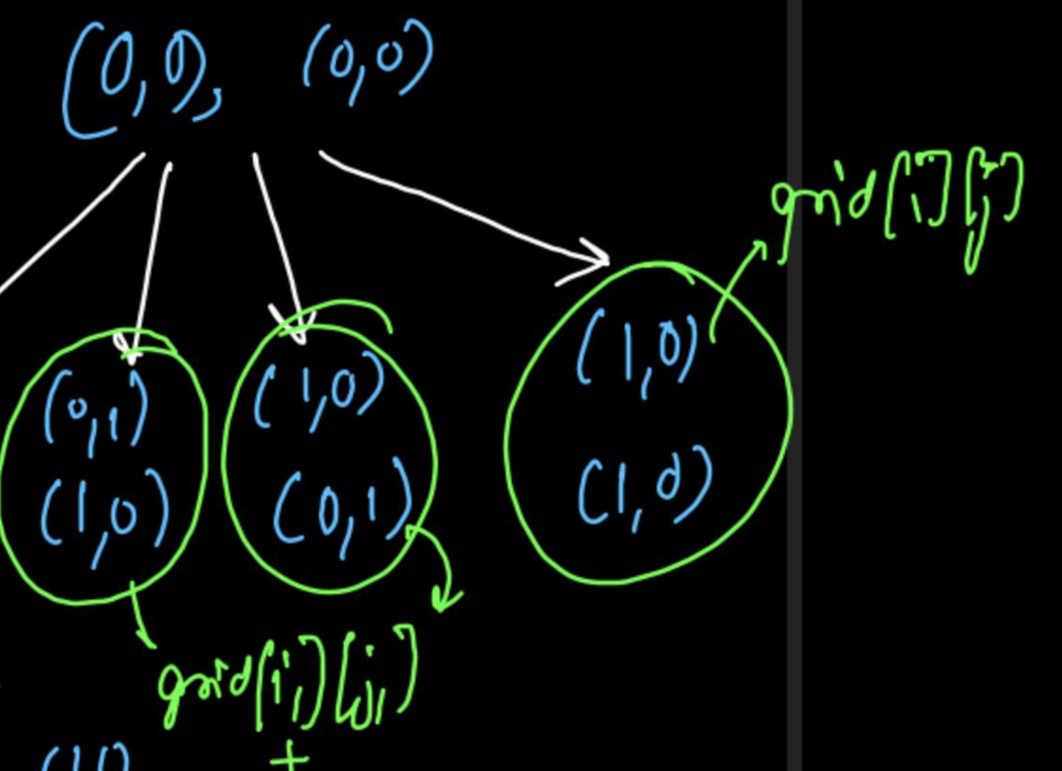
P_1 HR VD * P_2 HR VD

P_1 → P_2 → ①

P_1 → P_2 ↓ → ②

P_1 ↓ P_2 → → ③

P_1 ↓ P_2 ↓ → ④



Negative Base Case

P_1 → out of matrix P_2 → out of matrix

P_1 → blockage P_2 → blockage

Positive Base Case

$P_1 = P_2$ = destination

P_1 & P_2 ↔ starting from same source calls (steps distance)

⇒ Total dist by P_1 = Total Dist by P_2

$i_1 + j_1 = i_2 + j_2$

```java
public int helper(int row1, int col1, int row2, int col2, int[][] grid, int[][][] dp){
    // int col2 = row1 + col1 - row2; Col2 is not a variable

    if(row1 >= grid.length || col1 >= grid[0].length || grid[row1][col1] == -1
      || row2 >= grid.length || col2 >= grid[0].length || grid[row2][col2] == -1){
       // Negative Base Case (Out of Matrix or Blockage Node)
       return Integer.MIN_VALUE;
    }

    if(row1 == grid.length - 1 && col1 == grid[0].length - 1){
        // Positive Base Case: Destination Cell
        return grid[row1][col1];
    }

    if(dp[row1][col1][row2] != -1) return dp[row1][col1][row2];

    int ans = grid[row1][col1] + grid[row2][col2];
    if(row1 == row2 && col1 == col2) ans -= grid[row1][col1];

    int RR = helper(row1, col1 + 1, row2, col2 + 1, grid, dp);
    int RD = helper(row1, col1 + 1, row2 + 1, col2, grid, dp);
    int DR = helper(row1 + 1, col1, row2, col2 + 1, grid, dp);
    int DD = helper(row1 + 1, col1, row2 + 1, col2, grid, dp);

    return dp[row1][col1][row2] = Math.max(Math.max(RR, RD), Math.max(DR, DD)) + ans;
}
```

1. Indexing
2. Same - Simult
3. 4D → 3D

Time → O(N*N*N*4)

Space → 3D DP → O(N*N*N)

```java
public int cherryPickup(int[][] grid) {
    int[][][] dp = new int[grid.length][grid[0].length][grid.length];
    for(int i=0; i<dp.length; i++){
        for(int j=0; j<dp[0].length; j++){
            for(int k=0; k<dp[0][0].length; k++){
                dp[i][j][k] = -1;
            }
        }
    }

    int ans = helper(0, 0, 0, 0, grid, dp);
    if(ans <= 0) return 0;
    return ans;
}
```