Dynamic Programming

Q: Climb Stairs:

Tabulation concept

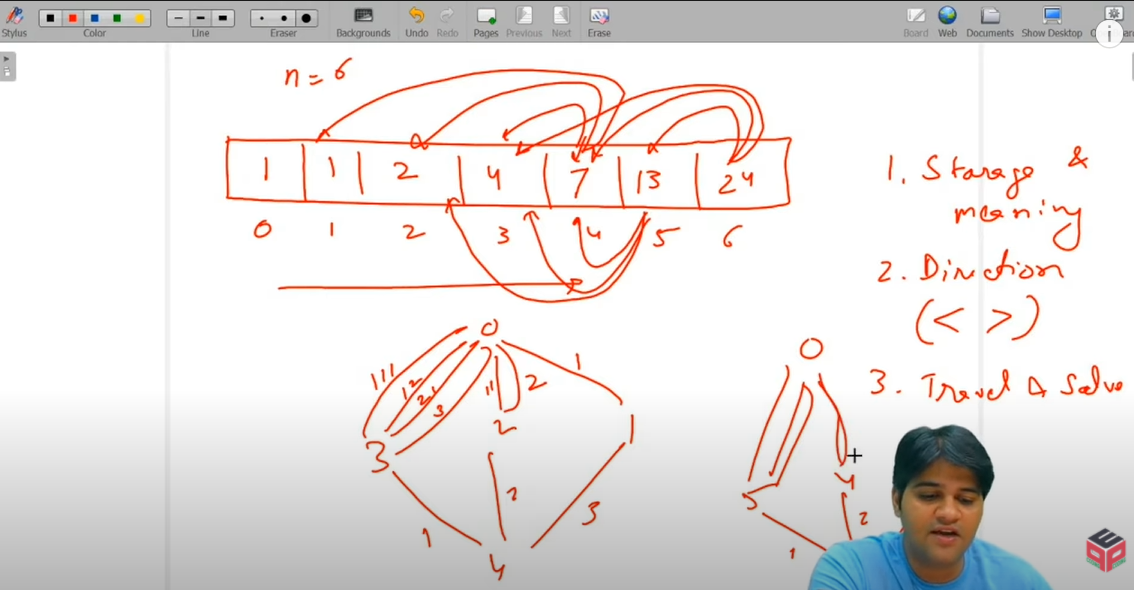
Steps:

(i) Assign storage (maybe in array or in a 2D vector or whatever) and meaning to it

(ii) Identify the direction of problem i.e. in figure below, we’ve to go to 6th to 0th stair,

So, we’ll consider (6 to 0) and (0 to 0). (0 to 0) is a smaller problem. So, we’ll start from 0th step and go till 6th step.

(iii) Travel and solve



Q: Min Cost Maze Traversal:

How to pass a 2D vector to a function:

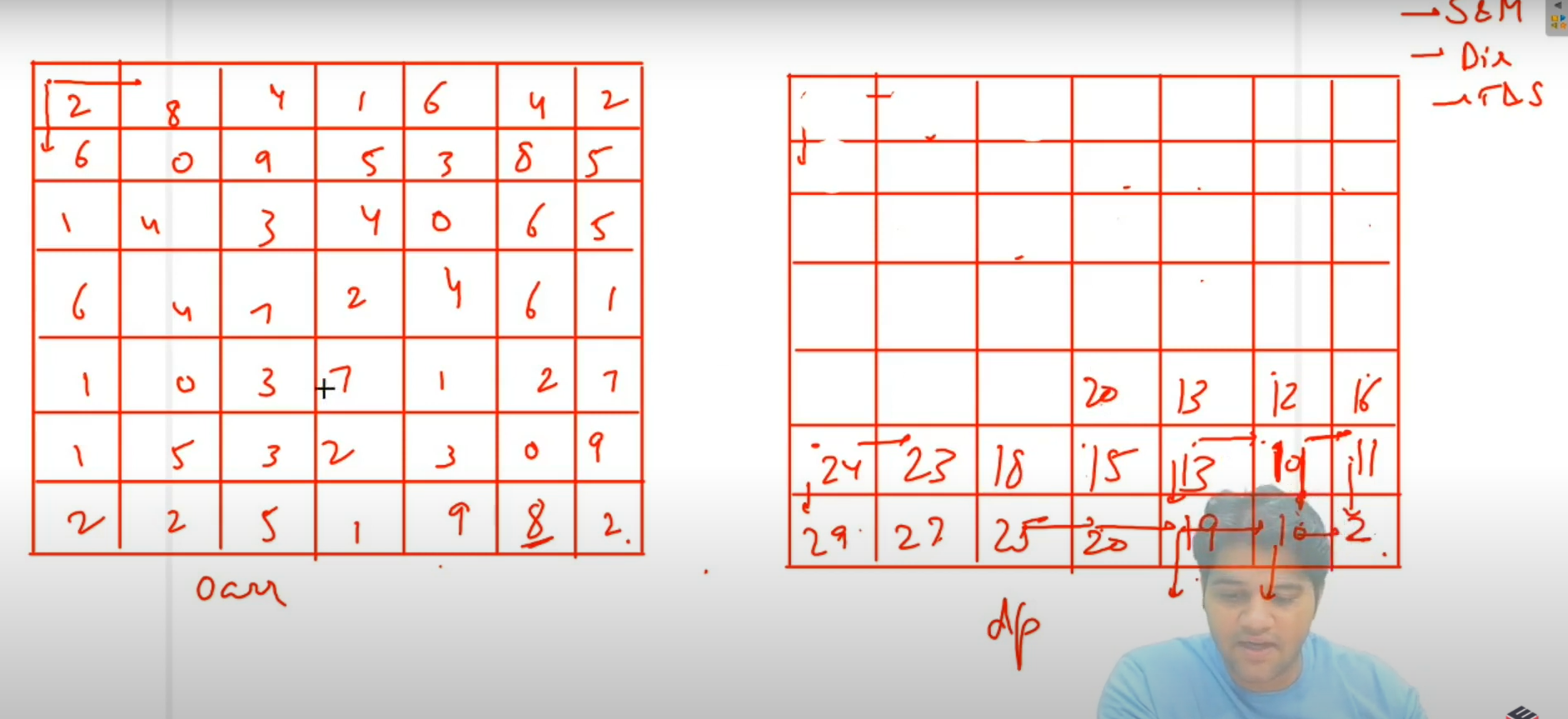
leastCost(mazeCost, 0, 0, rows, columns, prevSol);

mazeCost and prevSol are 2D vectors here. Function is below

int leastCost(vector<vector<int>>&mazeCost, int curr\_row, int curr\_col, int destRow,

int destCol, vector<vector<int>>&prevSol)

Tabulation approach for the same problem:



We’ve to go from (0, 0) to (6, 6). (6, 6) is smaller problem. So, we’ll start from there. And then we’ll fill horizontally like this ( ) by choosing whichever direction is costing less. In that way, each cell will contain the least value from that cell to reach the destination (6, 6). For example: The next cell to be filled will have the value 21. Because 18+3 < 20+3. And so on. Like that we’ll keep going and then finally the 0th cell will have the cost from 0th cell to (6, 6). And that is our answer!

Java tip:

**Array declaration:**

int [][]arr = new int [5][6]

Java tip:

**Min of two no.**

Math.min(a, b);

Java tip:

**Length of array:**

int len = arr.length;

Java tip:

**Length of array:**

int len = arr.length;

Java tip:

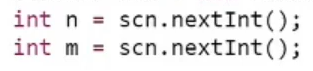
**Taking a Boolean array.**



Java tip:

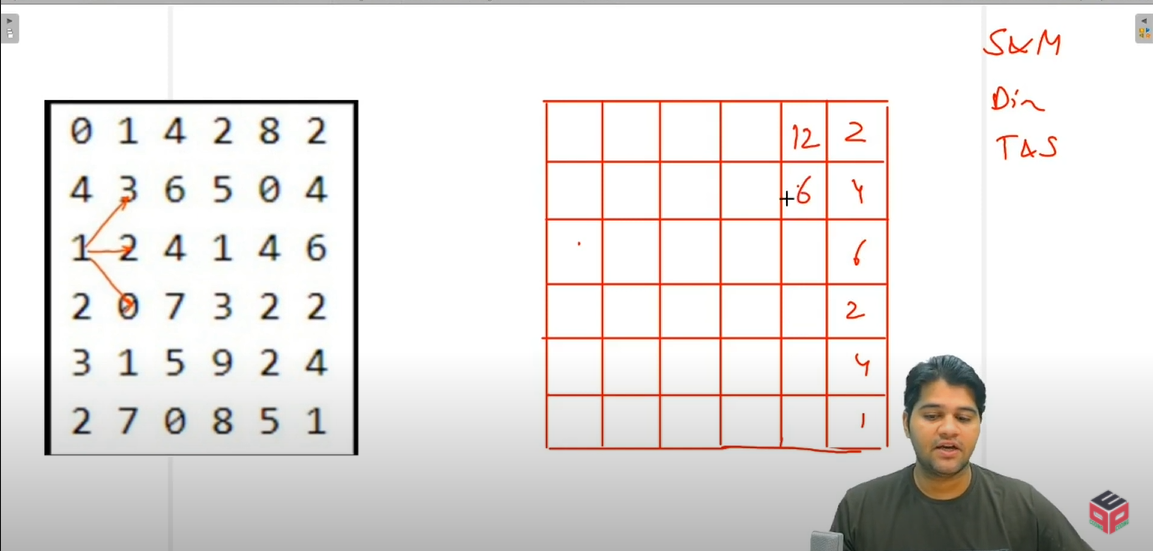
**Taking integer input**

int n=scn.nextInt();



Q: Goldmine Problem (tabulation approach)

(I’ve submitted on pepcoding using memoization approach)

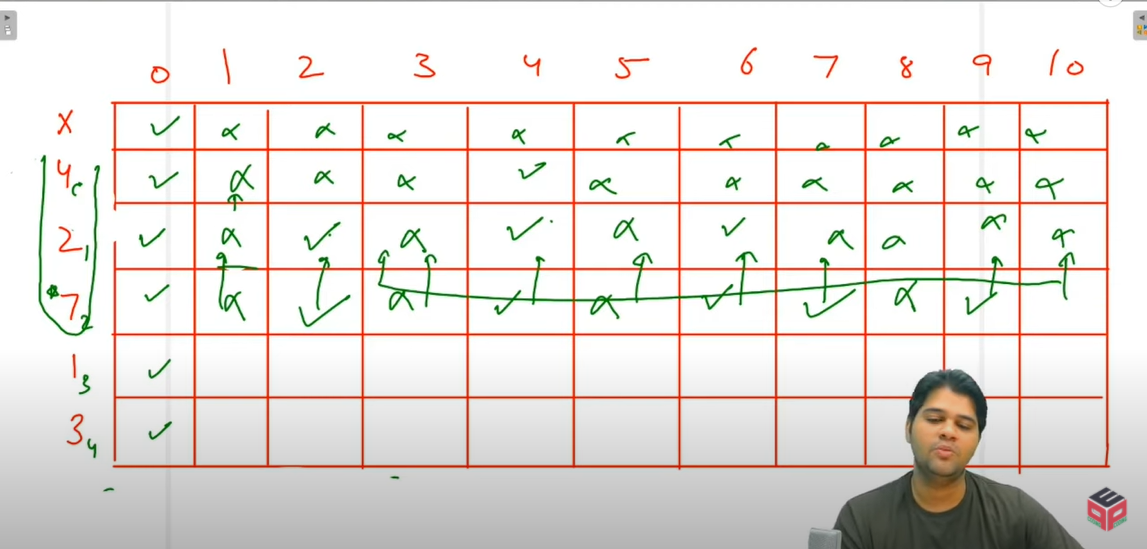


1. Storage and Meaning: Each cell will contain the maximum gold that can be collected from there. 2. Direction: Last column is the smallest problem. Because we can’t dig anywhere from there. So, it’ll have the value as it is.

3: We’ll travel and solve vertically and go wherever we get maximum gold either diagonally up or horizontally or diagonally down.

**Note:** While writing tabulation code, try to divide the 2D matrix in 4 parts – last column, last row, first row and the rest of the table, while filling it.

Q: Target Sum Subset:



No. of Rows = target+1; No. of Columns = size+1;

4, 2, 7, 1, 3 : These are numbers and on rows, we are putting true if that sum can be achieved by any combination. For ex: consider dp[3][4]. Sum of 4 can be made without including 7 by 4 and 2 (true in a bracket just above by 1 row || if 7 included, then 1 row above and 7 columns left, if that is true means can be made, so dp[3][4] = true or otherwise false)

Q: 0 1 Knapsack:

