stair	0	1	2	3	4
jumps →	3	3	2	0	1

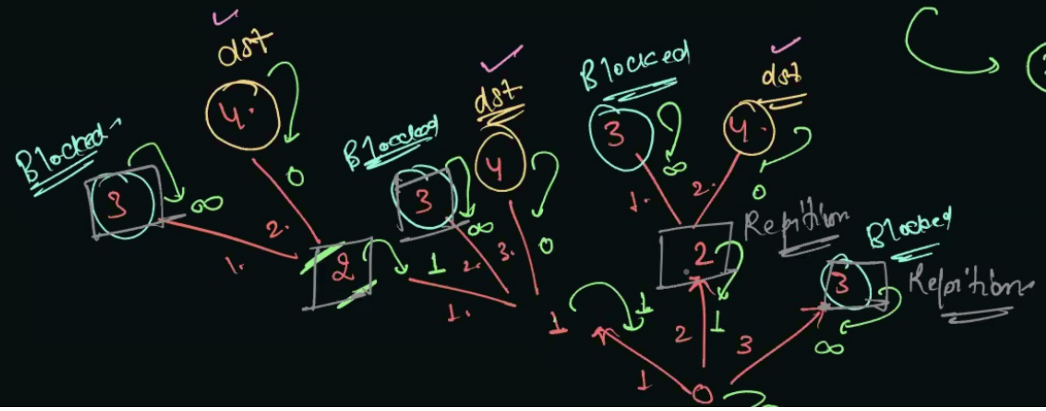
$\hookrightarrow$ 0 jump.

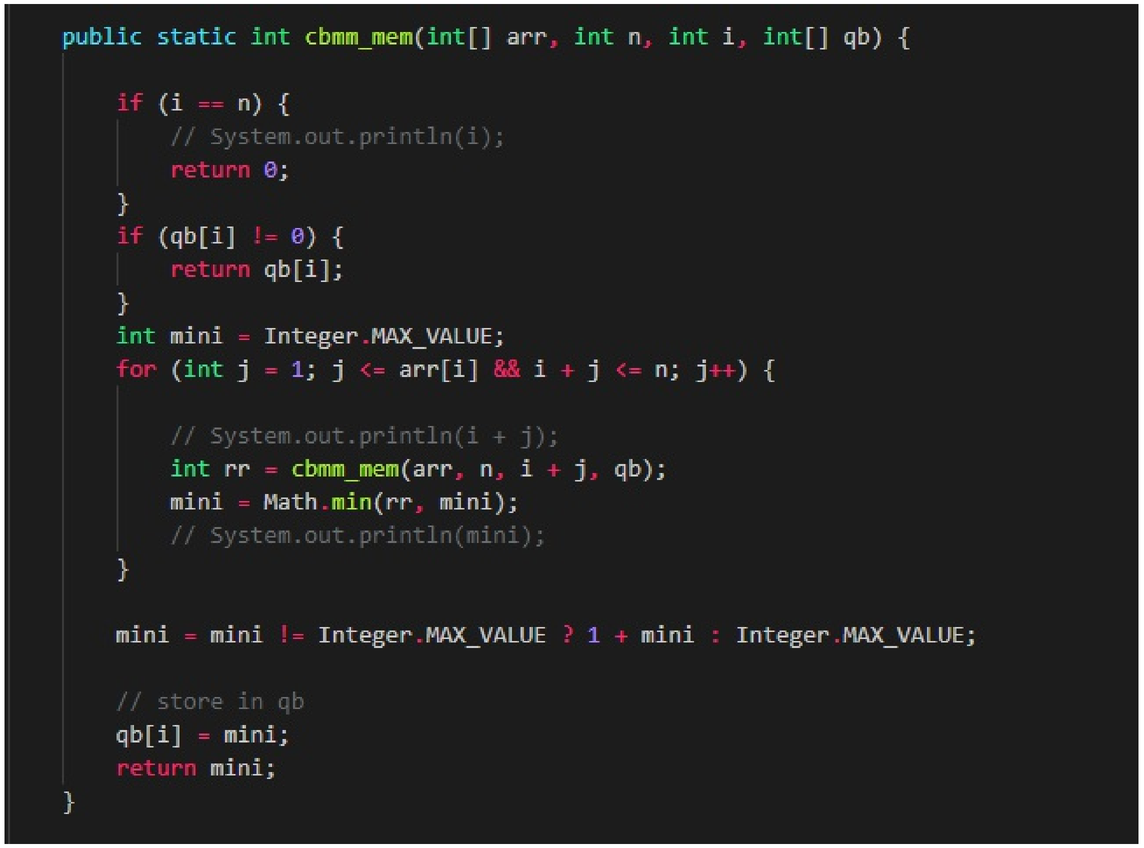
path from src to dst →

③ → 1 1 2
② → 1 3
② → 2 2

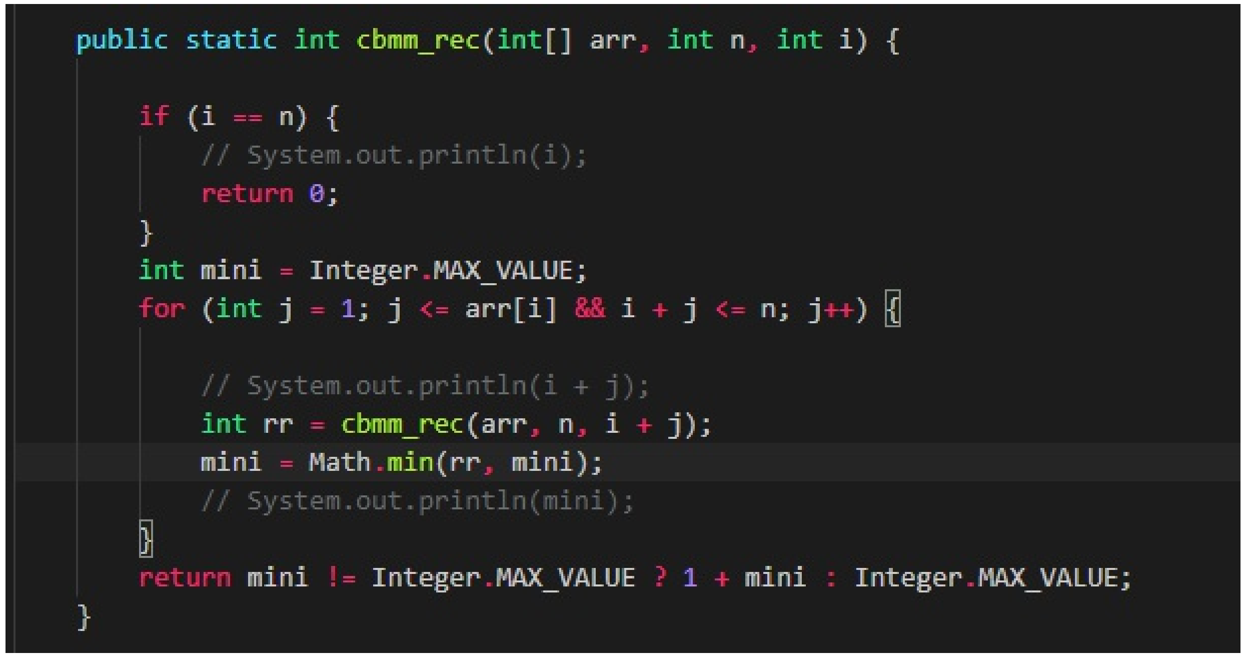
} min jumps in these path,

$\hookrightarrow$ ② is min jumps

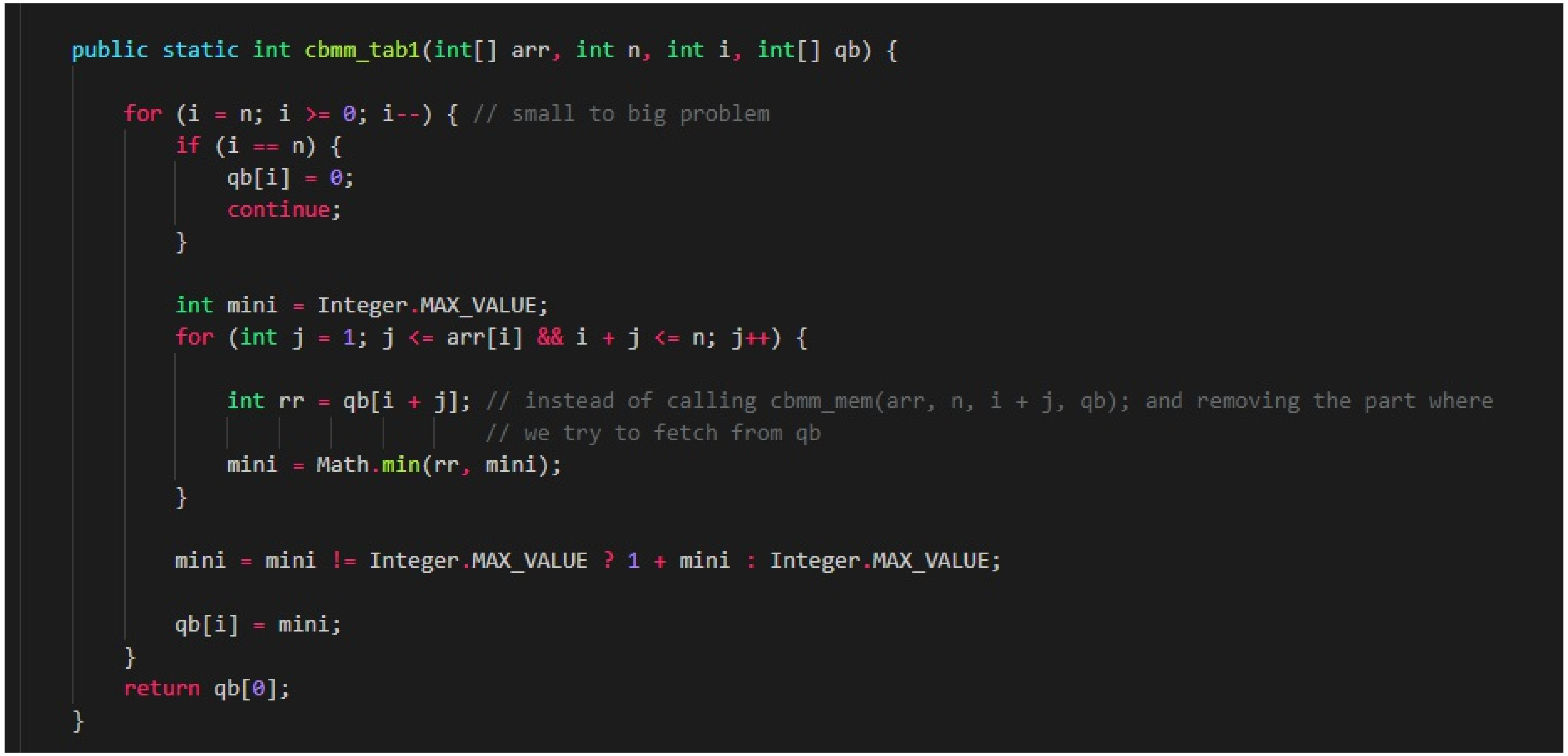




```java
public static int cbmm_mem(int[] arr, int n, int i, int[] qb) {

    if (i == n) {
        // System.out.println(i);
        return 0;
    }
    if (qb[i] != 0) {
        return qb[i];
    }
    int mini = Integer.MAX_VALUE;
    for (int j = 1; j <= arr[i] && i + j <= n; j++) {

        // System.out.println(i + j);
        int rr = cbmm_mem(arr, n, i + j, qb);
        mini = Math.min(rr, mini);
        // System.out.println(mini);
    }

    mini = mini != Integer.MAX_VALUE ? 1 + mini : Integer.MAX_VALUE;

    // store in qb
    qb[i] = mini;
    return mini;
}
```



```java
public static int cbmm_rec(int[] arr, int n, int i) {

    if (i == n) {
        // System.out.println(i);
        return 0;
    }
    int mini = Integer.MAX_VALUE;
    for (int j = 1; j <= arr[i] && i + j <= n; j++) {

        // System.out.println(i + j);
        int rr = cbmm_rec(arr, n, i + j);
        mini = Math.min(rr, mini);
        // System.out.println(mini);
    }
    return mini != Integer.MAX_VALUE ? 1 + mini : Integer.MAX_VALUE;
}
```



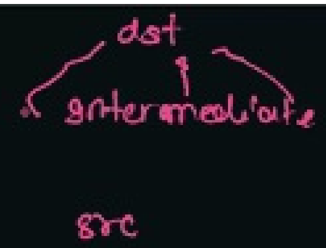
```java
public static int cbmm_tab1(int[] arr, int n, int i, int[] qb) {

    for (i = n; i >= 0; i--) { // small to big problem
        if (i == n) {
            qb[i] = 0;
            continue;
        }

        int mini = Integer.MAX_VALUE;
        for (int j = 1; j <= arr[i] && i + j <= n; j++) {

            int rr = qb[i + j]; // instead of calling cbmm_mem(arr, n, i + j, qb); and removing the part where
                                // we try to fetch from qb
            mini = Math.min(rr, mini);
        }

        mini = mini != Integer.MAX_VALUE ? 1 + mini : Integer.MAX_VALUE;

        qb[i] = mini;
    }
    return qb[0];
}
```

Min Cost In Maze Traversal

Thursday, 24 June 2021 6:12 PM

jumps allowed → right, down

(x,y) → (x, y+1)
↓
(x+1, y)

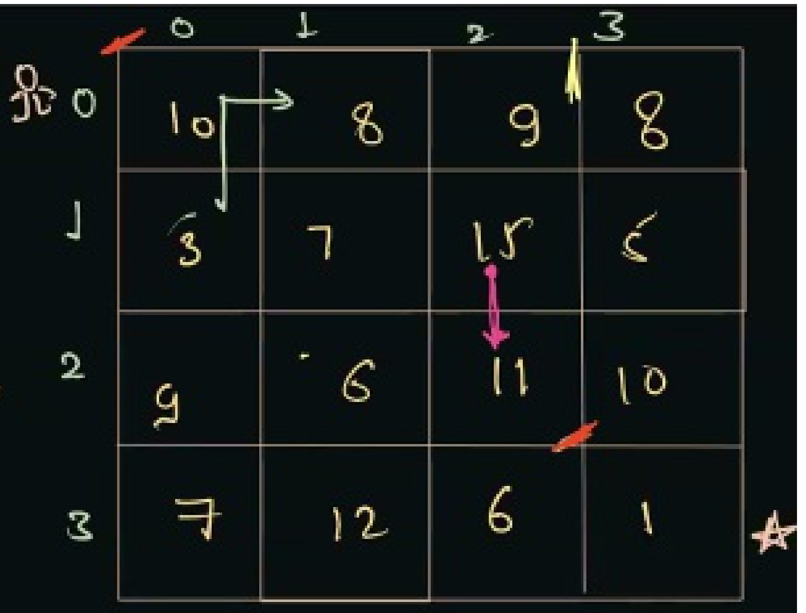


Maze with cost →

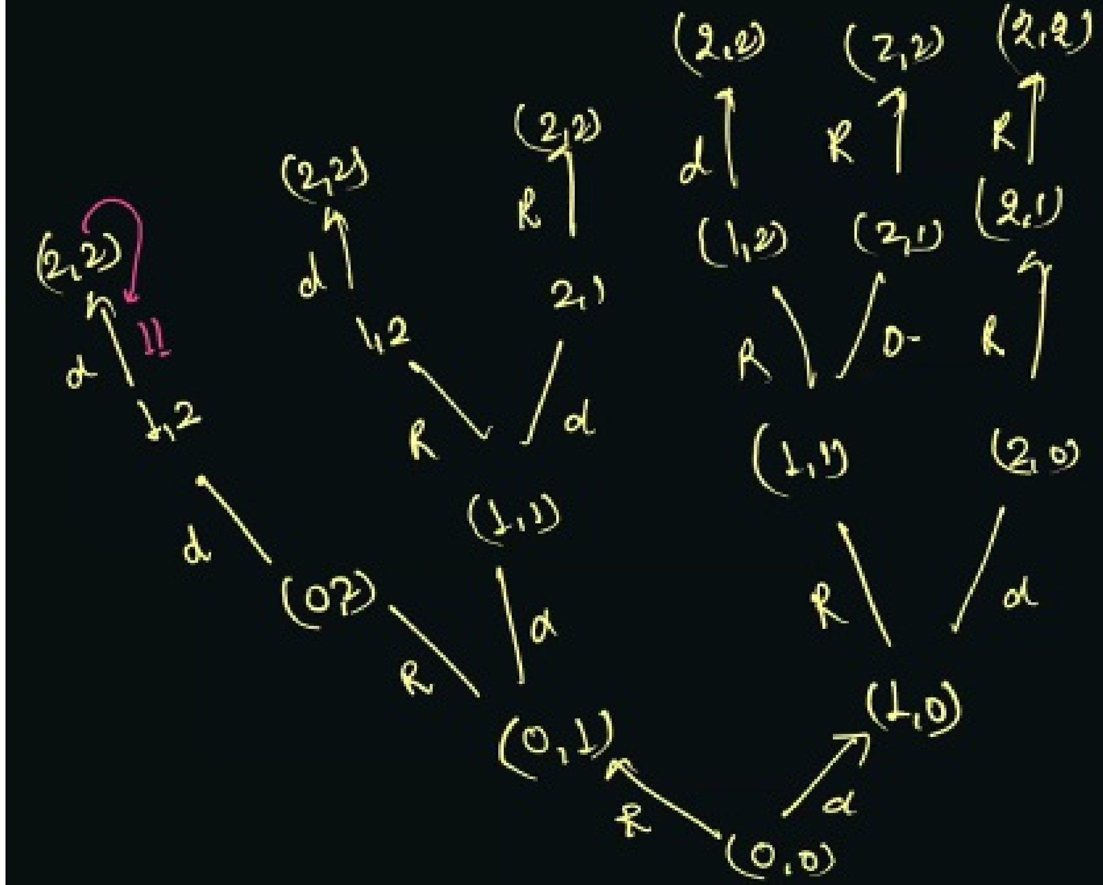
src ⇒ (0,0)

dst ⇒ (3,3)

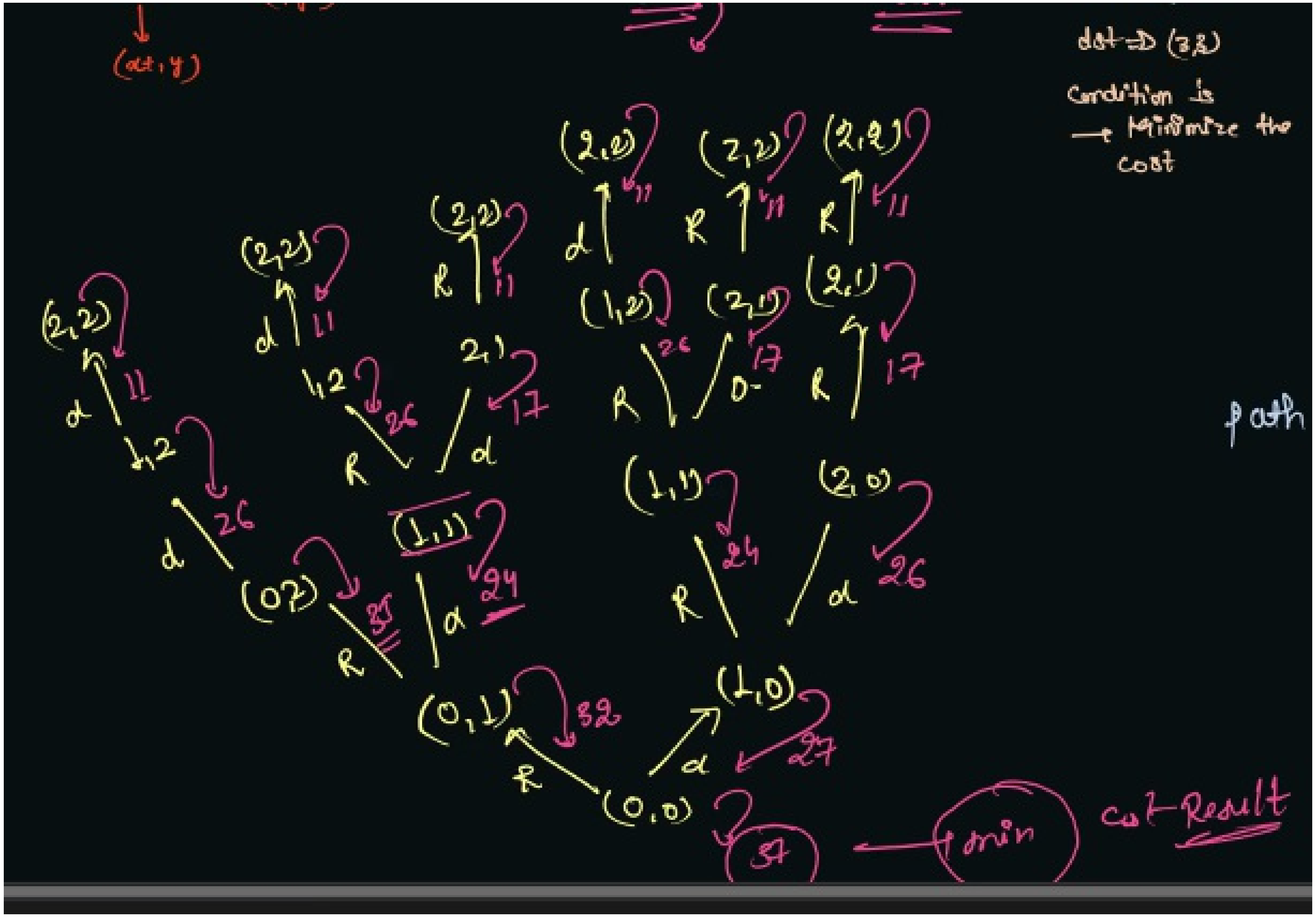
Condition is → Minimize the cost



	0	1	2	3
0	10	8	9	8
1	3	7	15	5
2	9	6	11	10
3	7	12	6	1



		Cost
path →	R R D D →	53
	R D R D →	51
	R D D R →	42
	D R R D →	46
Min cost path ⇒	D R D R →	37
	D D R R →	39

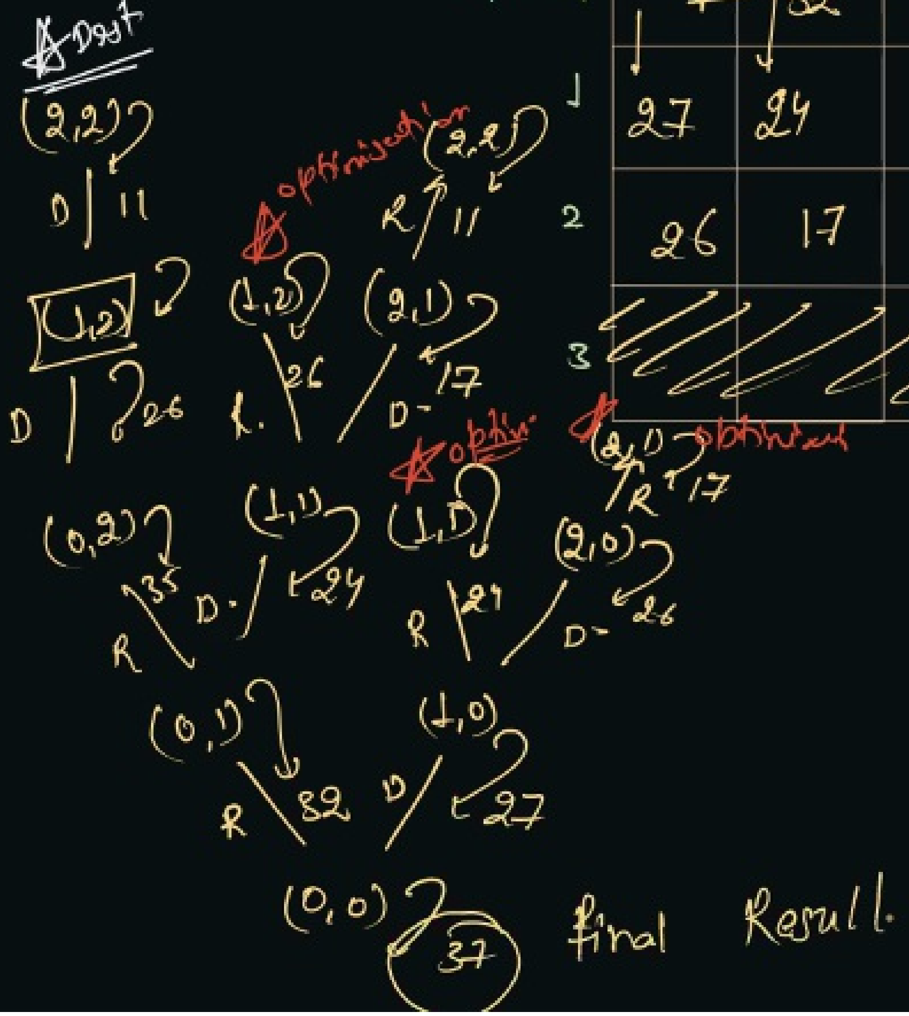

(x+1, y)
dst => (2,2)
Condition is → Minimize the cost
path
(0,0)
(0,1)
(1,0)
(0,2)
(1,1)
(2,0)
(2,1)
(1,2)
(2,2)
R
d
32
27
35
24
26
17
11
37
min
cost Result

dp →

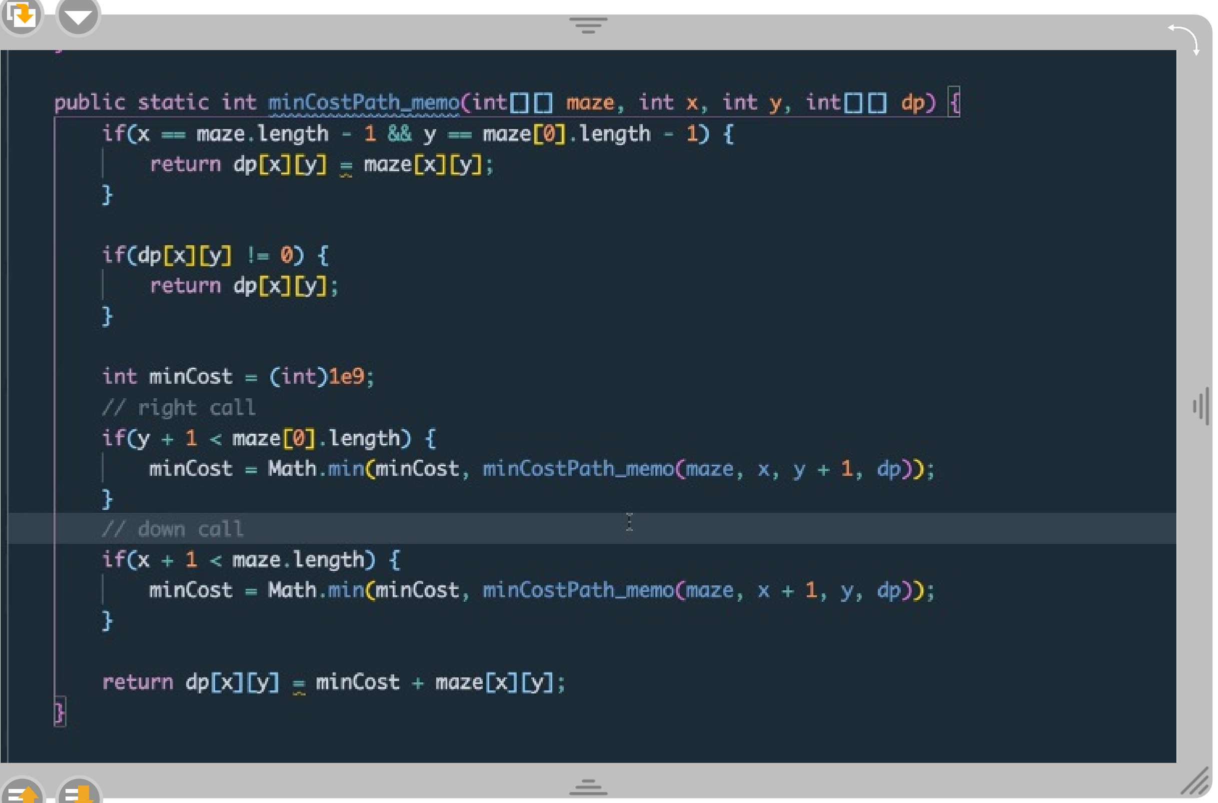
$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \leftarrow & 32 & 35 & ▨ \\ 1 & 27 & 24 & 26 & ▨ \\ 2 & 26 & 17 & 11 & ▨ \\ 3 & ▨ & ▨ & ▨ & ▨ \end{array}$$

maze →

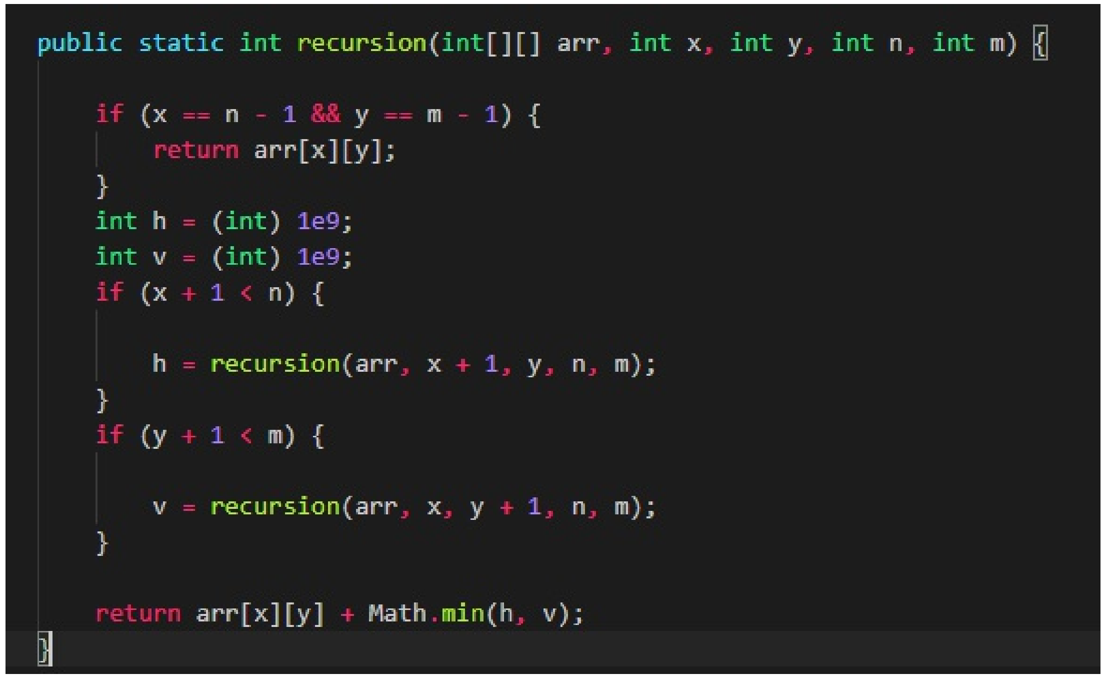
$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 10 & 8 & 9 & 8 \\ 1 & 3 & 7 & 15 & 5 \\ 2 & 9 & 6 & 11 & 10 \\ 3 & 7 & 12 & 6 & 1 \end{array}$$



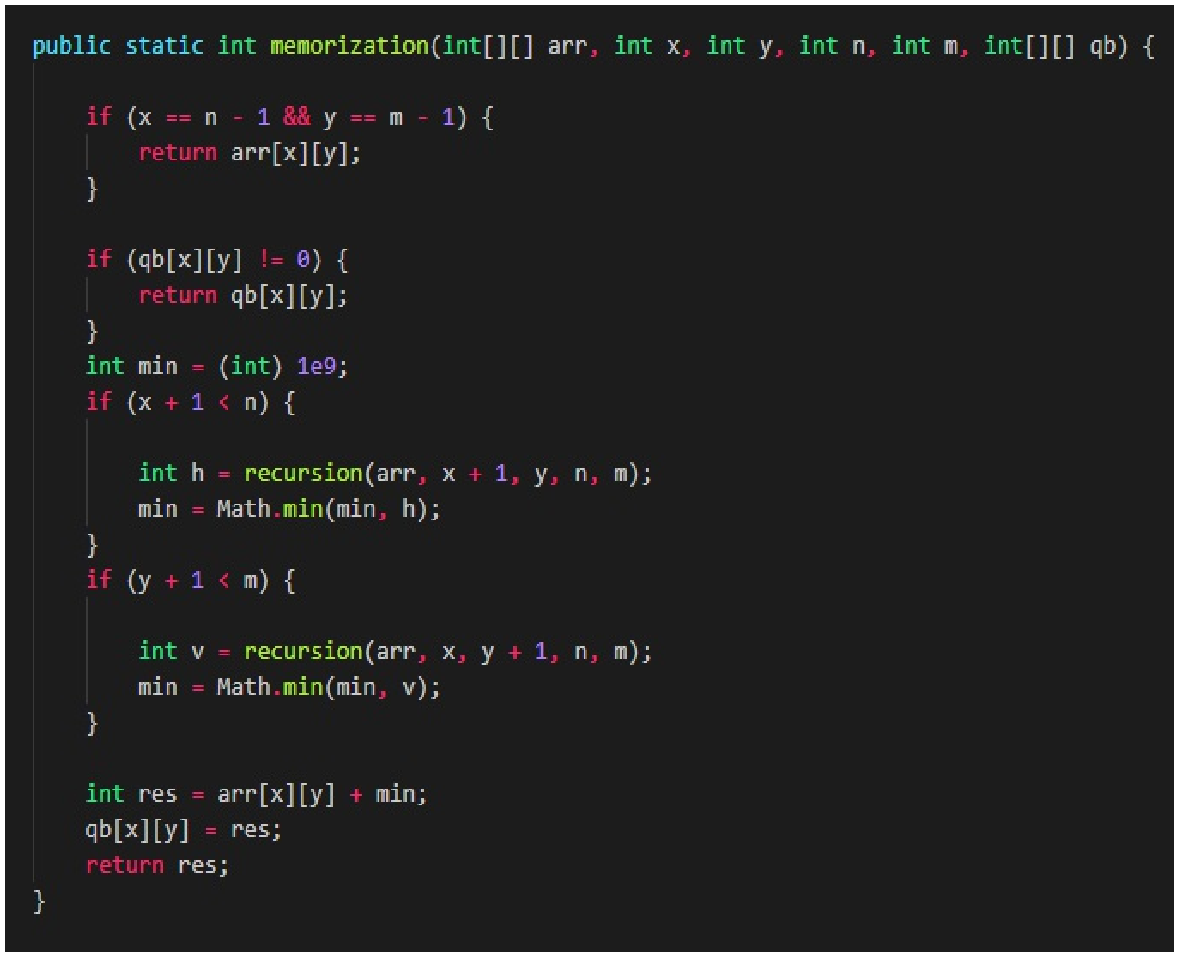
dp[r][c] = minCost + maze[r][c]



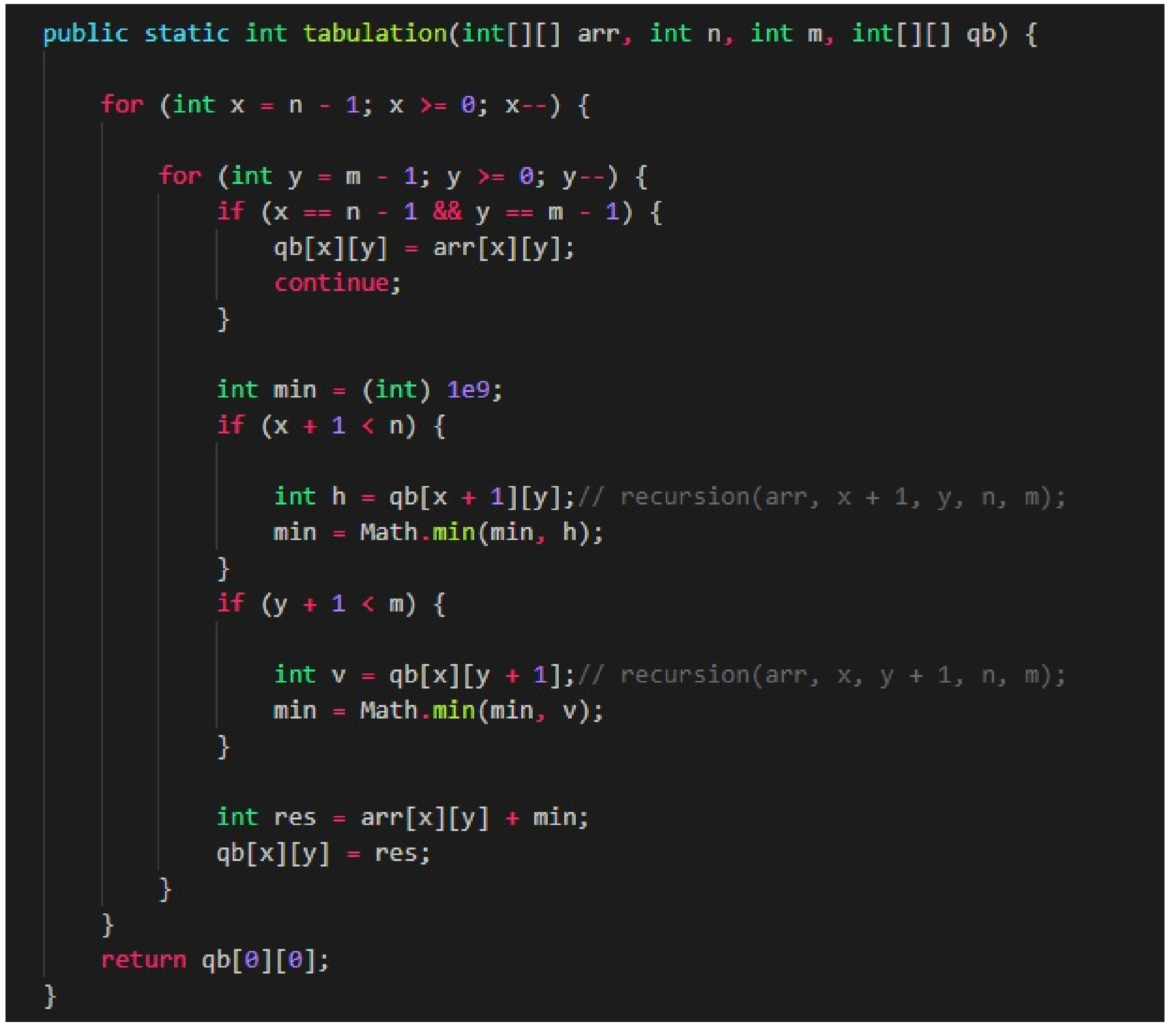
```java
public static int minCostPath_memo(int[][] maze, int x, int y, int[][] dp) {
    if(x == maze.length - 1 && y == maze[0].length - 1) {
        return dp[x][y] = maze[x][y];
    }

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

    int minCost = (int)1e9;
    // right call
    if(y + 1 < maze[0].length) {
        minCost = Math.min(minCost, minCostPath_memo(maze, x, y + 1, dp));
    }
    // down call
    if(x + 1 < maze.length) {
        minCost = Math.min(minCost, minCostPath_memo(maze, x + 1, y, dp));
    }

    return dp[x][y] = minCost + maze[x][y];
}
```



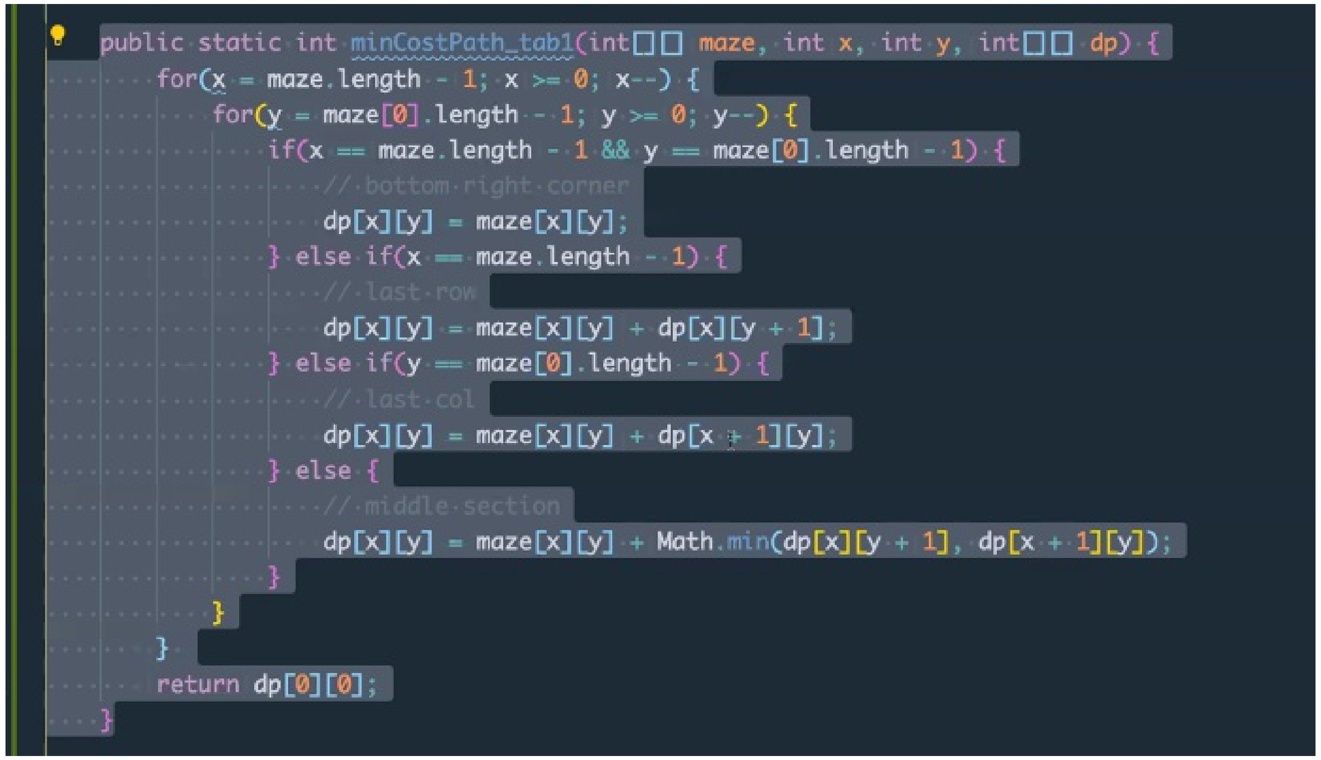
```java
public static int recursion(int[][] arr, int x, int y, int n, int m) {

    if (x == n - 1 && y == m - 1) {
        return arr[x][y];
    }
    int h = (int) 1e9;
    int v = (int) 1e9;
    if (x + 1 < n) {

        h = recursion(arr, x + 1, y, n, m);
    }
    if (y + 1 < m) {

        v = recursion(arr, x, y + 1, n, m);
    }

    return arr[x][y] + Math.min(h, v);
}
```



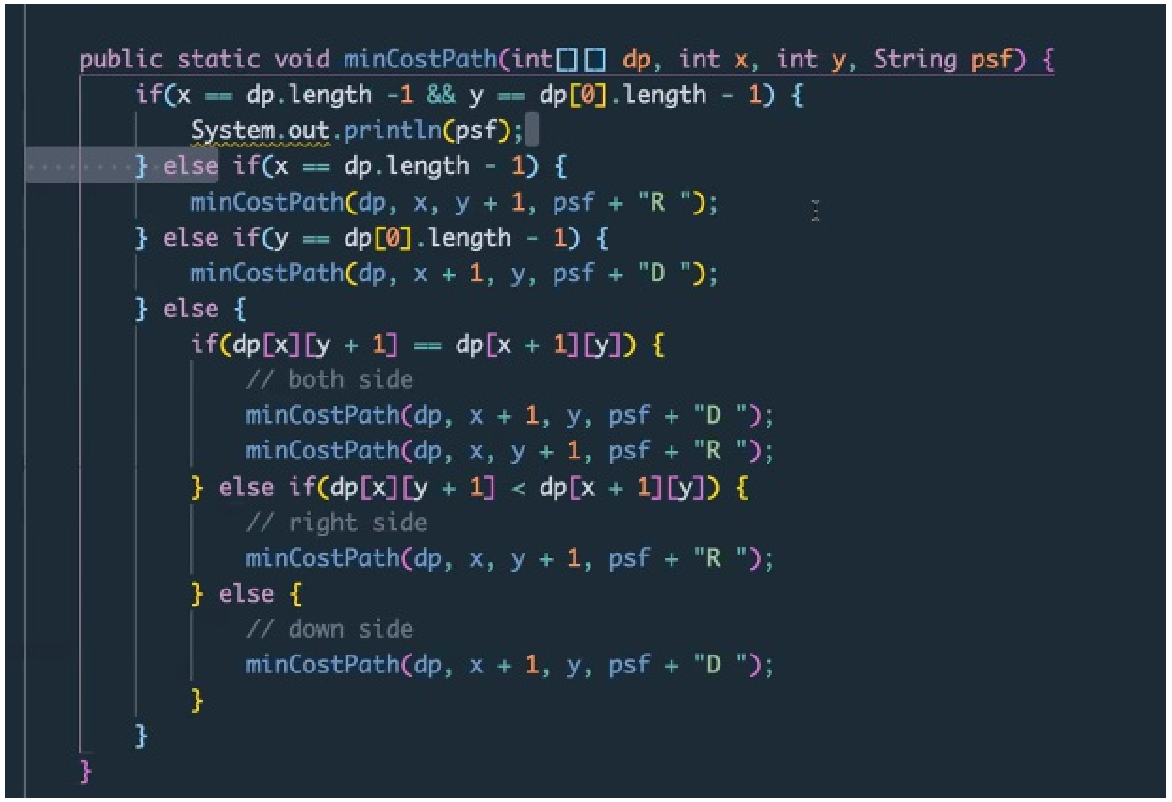
```java
public static int memorization(int[][] arr, int x, int y, int n, int m, int[][] qb) {

    if (x == n - 1 && y == m - 1) {
        return arr[x][y];
    }

    if (qb[x][y] != 0) {
        return qb[x][y];
    }
    int min = (int) 1e9;
    if (x + 1 < n) {

        int h = recursion(arr, x + 1, y, n, m);
        min = Math.min(min, h);
    }
    if (y + 1 < m) {

        int v = recursion(arr, x, y + 1, n, m);
        min = Math.min(min, v);
    }

    int res = arr[x][y] + min;
    qb[x][y] = res;
    return res;
}
```



```java
public static int tabulation(int[][] arr, int n, int m, int[][] qb) {

    for (int x = n - 1; x >= 0; x--) {

        for (int y = m - 1; y >= 0; y--) {
            if (x == n - 1 && y == m - 1) {
                qb[x][y] = arr[x][y];
                continue;
            }

            int min = (int) 1e9;
            if (x + 1 < n) {

                int h = qb[x + 1][y];// recursion(arr, x + 1, y, n, m);
                min = Math.min(min, h);
            }
            if (y + 1 < m) {

                int v = qb[x][y + 1];// recursion(arr, x, y + 1, n, m);
                min = Math.min(min, v);
            }

            int res = arr[x][y] + min;
            qb[x][y] = res;
        }
    }
    return qb[0][0];
}
```



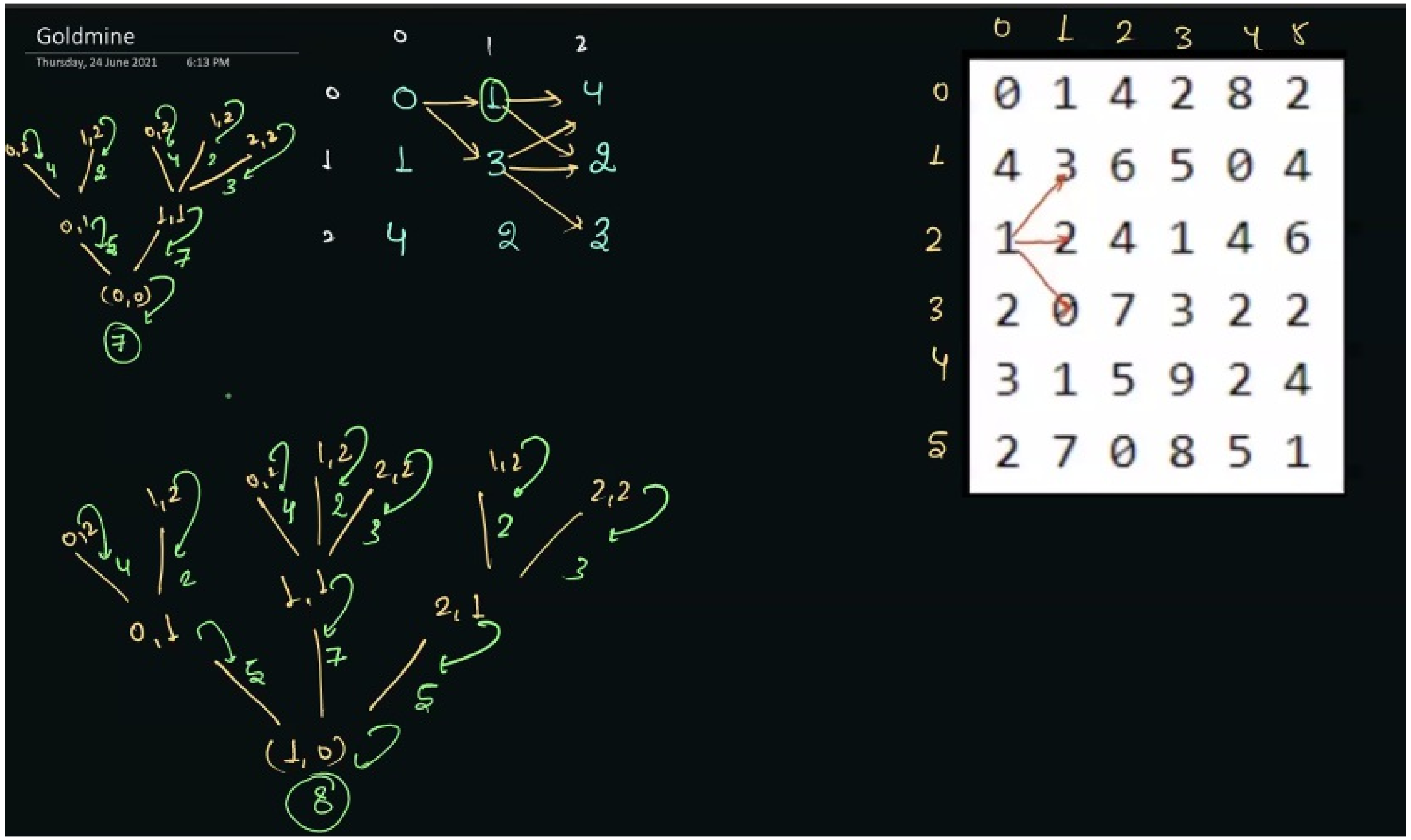
```java
public static int minCostPath_tab1(int[][] maze, int x, int y, int[][] dp) {
    for(x = maze.length - 1; x >= 0; x--) {
        for(y = maze[0].length - 1; y >= 0; y--) {
            if(x == maze.length - 1 && y == maze[0].length - 1) {
                // bottom right corner
                dp[x][y] = maze[x][y];
            } else if(x == maze.length - 1) {
                // last row
                dp[x][y] = maze[x][y] + dp[x][y + 1];
            } else if(y == maze[0].length - 1) {
                // last col
                dp[x][y] = maze[x][y] + dp[x + 1][y];
            } else {
                // middle section
                dp[x][y] = maze[x][y] + Math.min(dp[x][y + 1], dp[x + 1][y]);
            }
        }
    }
    return dp[0][0];
}
```



```java
public static void minCostPath(int[][] dp, int x, int y, String psf) {
    if(x == dp.length -1 && y == dp[0].length - 1) {
        System.out.println(psf);
    } else if(x == dp.length - 1) {
        minCostPath(dp, x, y + 1, psf + "R ");
    } else if(y == dp[0].length - 1) {
        minCostPath(dp, x + 1, y, psf + "D ");
    } else {
        if(dp[x][y + 1] == dp[x + 1][y]) {
            // both side
            minCostPath(dp, x + 1, y, psf + "D ");
            minCostPath(dp, x, y + 1, psf + "R ");
        } else if(dp[x][y + 1] < dp[x + 1][y]) {
            // right side
            minCostPath(dp, x, y + 1, psf + "R ");
        } else {
            // down side
            minCostPath(dp, x + 1, y, psf + "D ");
        }
    }
}
```

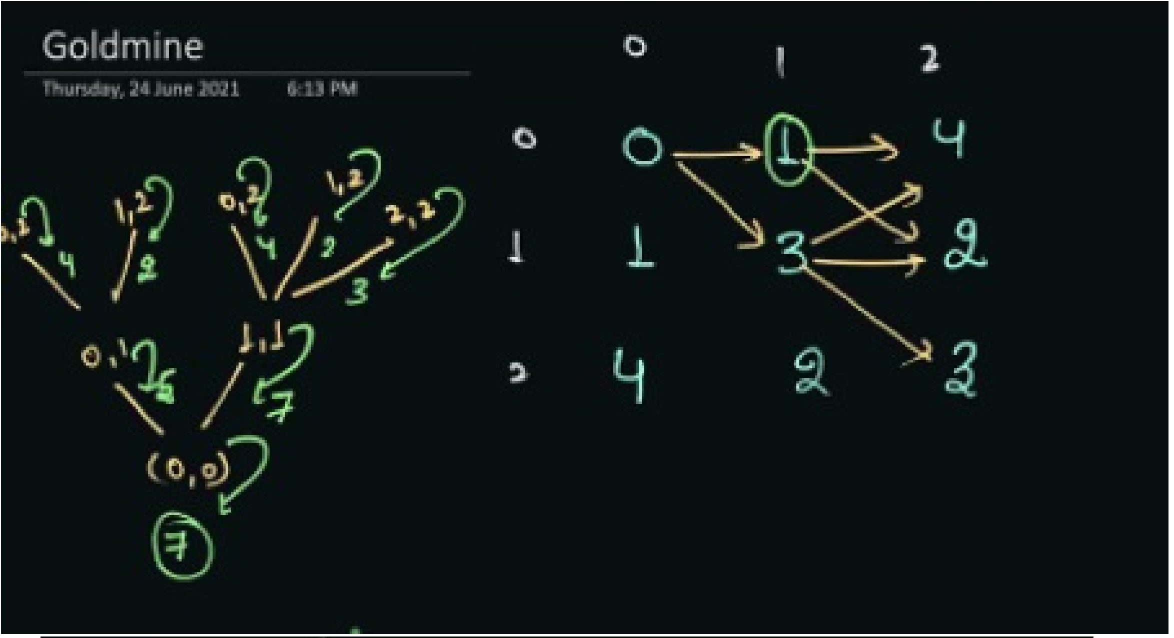
Goldmine

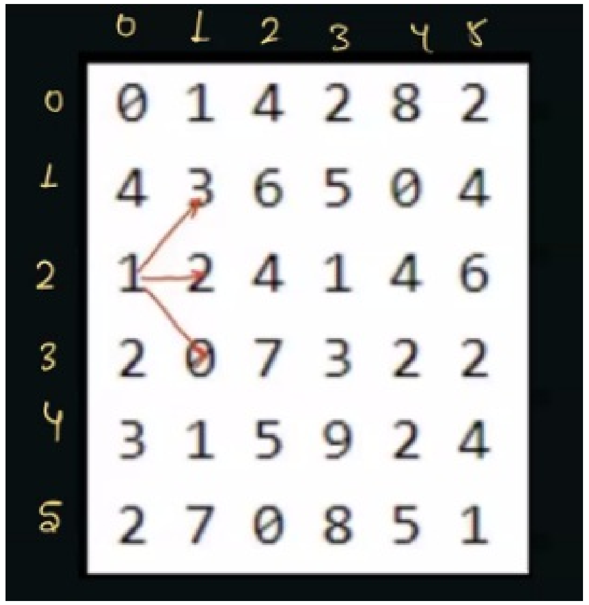
Thursday, 24 June 2021 6:13 PM



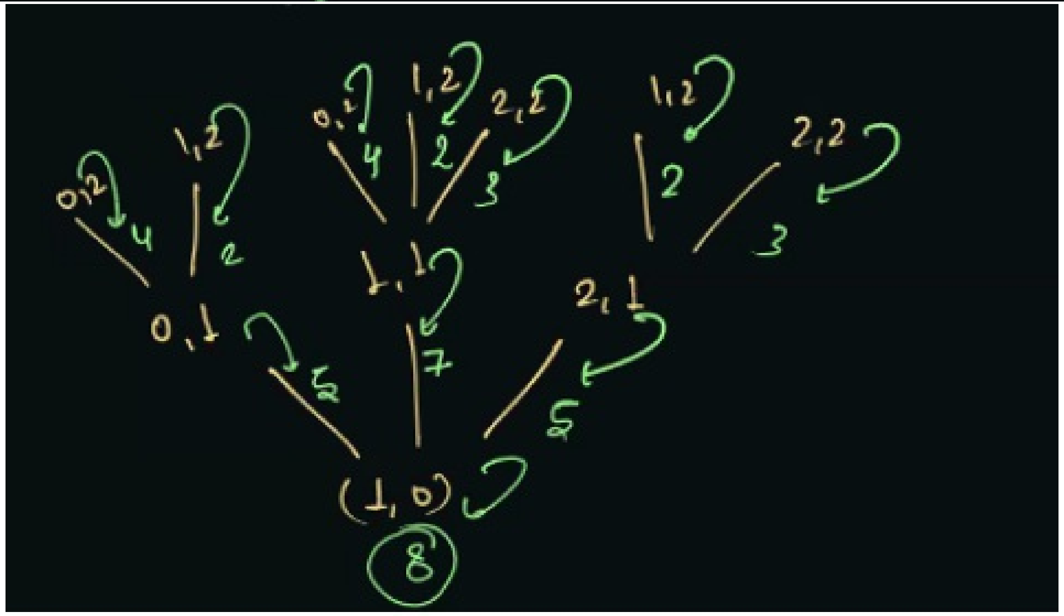
	0	1	2
0	0	1	4
1	1	3	2
2	4	2	3

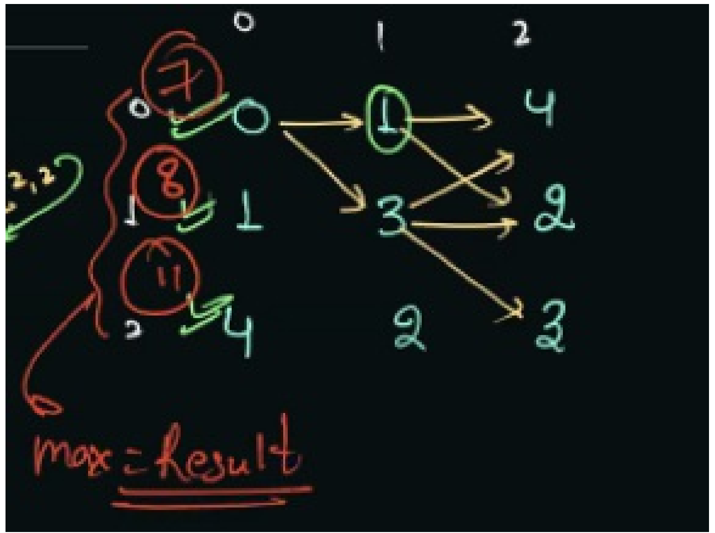
	0	1	2	3	4	5
0	0	1	4	2	8	2
1	4	3	6	5	0	4
2	1	2	4	1	4	6
3	2	0	7	3	2	2
4	3	1	5	9	2	4
5	2	7	0	8	5	1

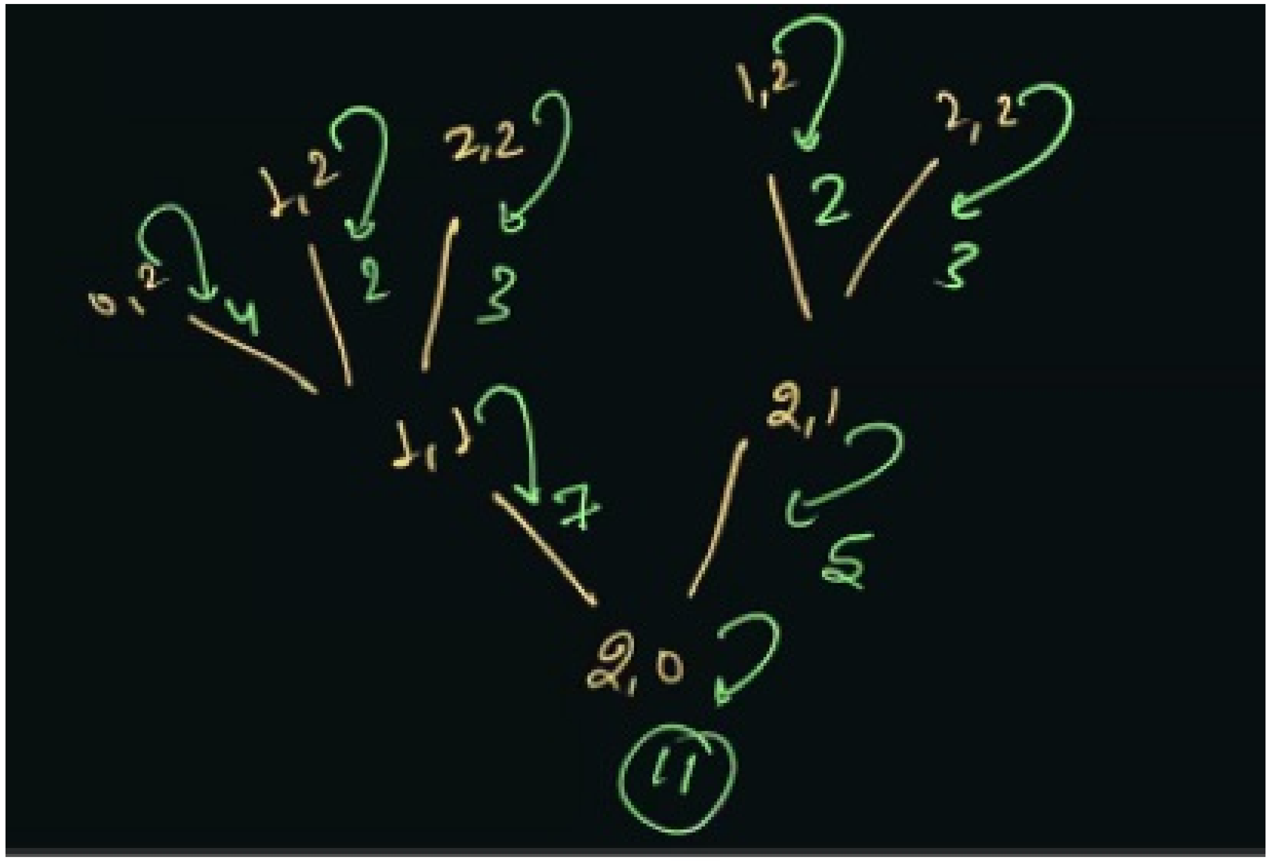


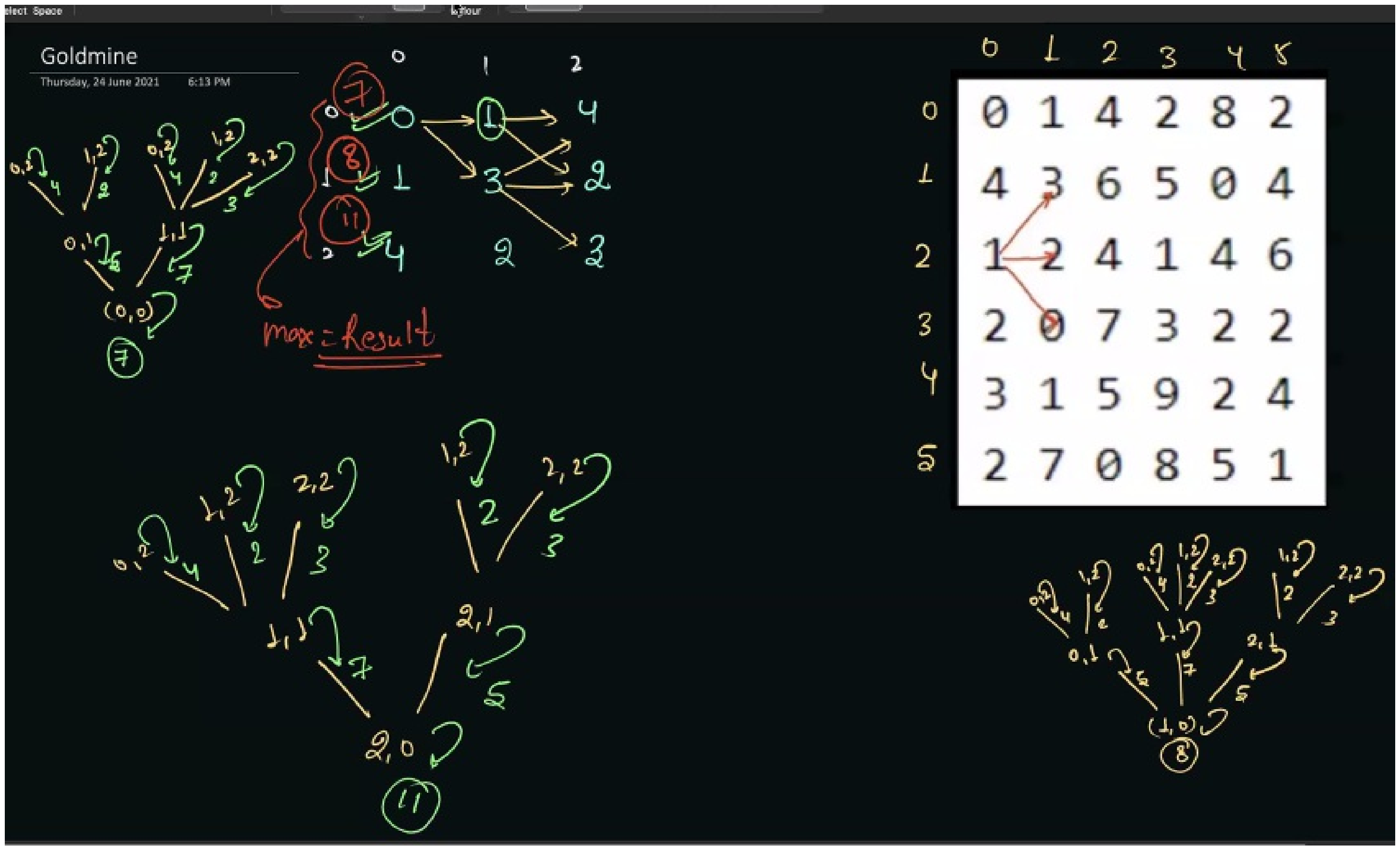


	0	1	2	3	4	5
0	0	1	4	2	8	2
1	4	3	6	5	0	4
2	1	2	4	1	4	6
3	2	0	7	3	2	2
4	3	1	5	9	2	4
5	2	7	0	8	5	1





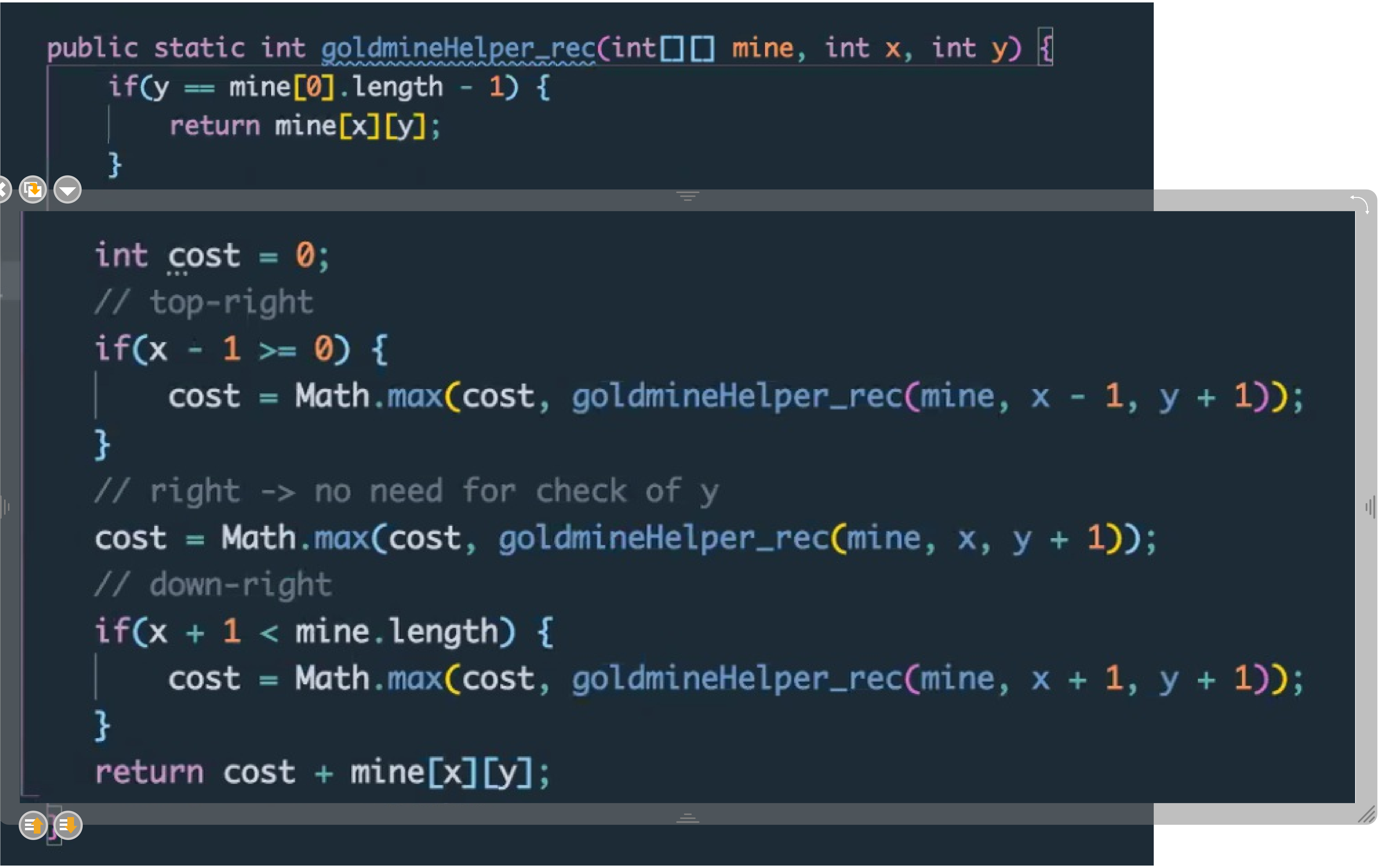




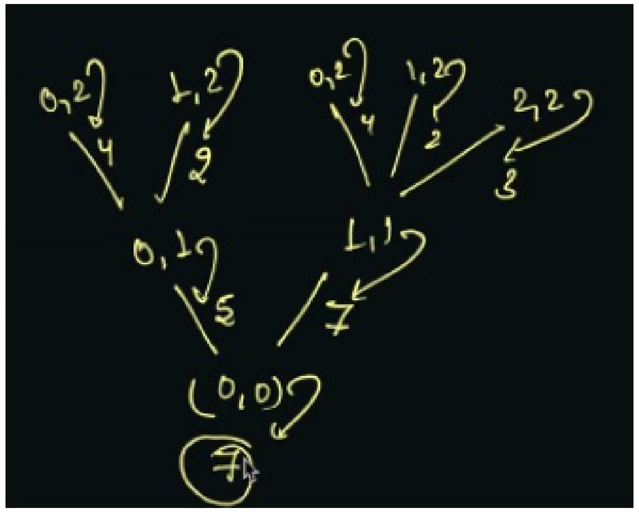
Goldmine

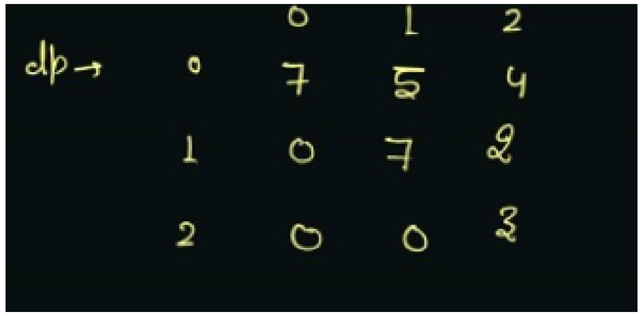
Thursday, 24 June 2021 6:13 PM

	0	1	2	3	4	5
0	0	1	4	2	8	2
1	4	3	6	5	0	4
2	1	2	4	1	4	6
3	2	0	7	3	2	2
4	3	1	5	9	2	4
5	2	7	0	8	5	1

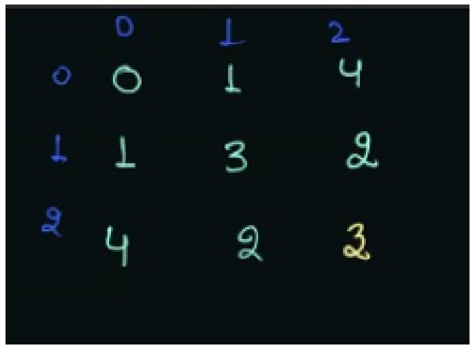


```java
public static int goldmineHelper_rec(int[][] mine, int x, int y) {
    if(y == mine[0].length - 1) {
        return mine[x][y];
    }
    int cost = 0;
    // top-right
    if(x - 1 >= 0) {
        cost = Math.max(cost, goldmineHelper_rec(mine, x - 1, y + 1));
    }
    // right -> no need for check of y
    cost = Math.max(cost, goldmineHelper_rec(mine, x, y + 1));
    // down-right
    if(x + 1 < mine.length) {
        cost = Math.max(cost, goldmineHelper_rec(mine, x + 1, y + 1));
    }
    return cost + mine[x][y];
}
```

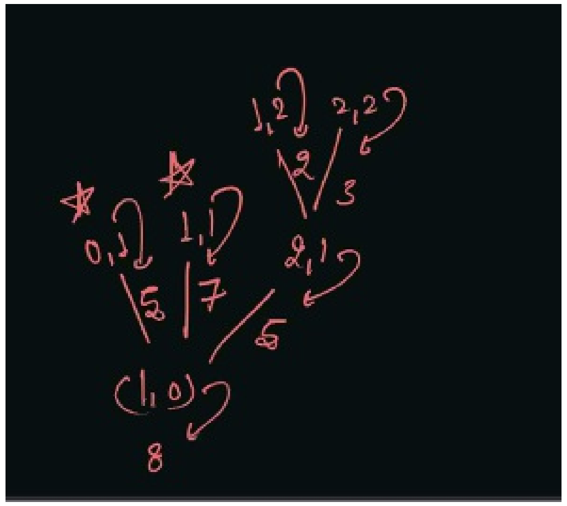


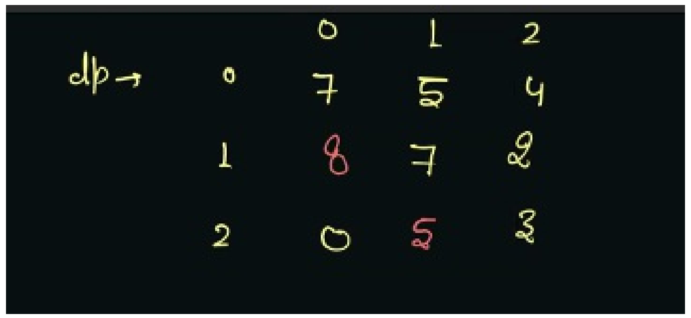


dp→	0	1	2
0	7	5	4
1	0	7	2
2	0	0	3

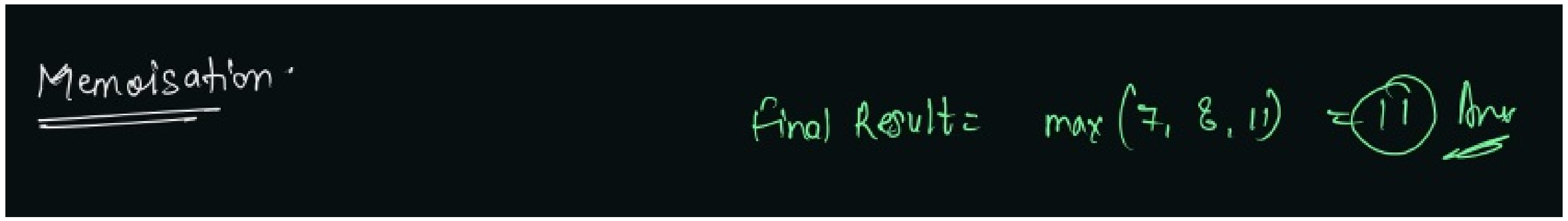


	0	1	2
0	0	1	4
1	1	3	2
2	4	2	3



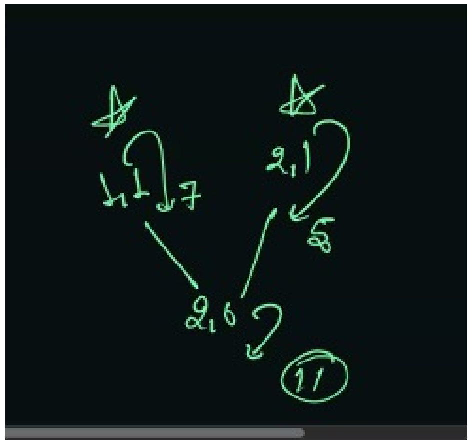


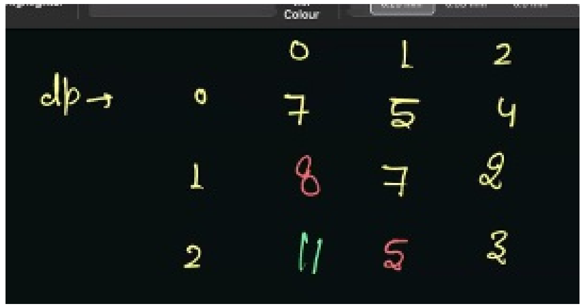
dp→	0	1	2
0	7	5	4
1	8	7	2
2	0	5	3



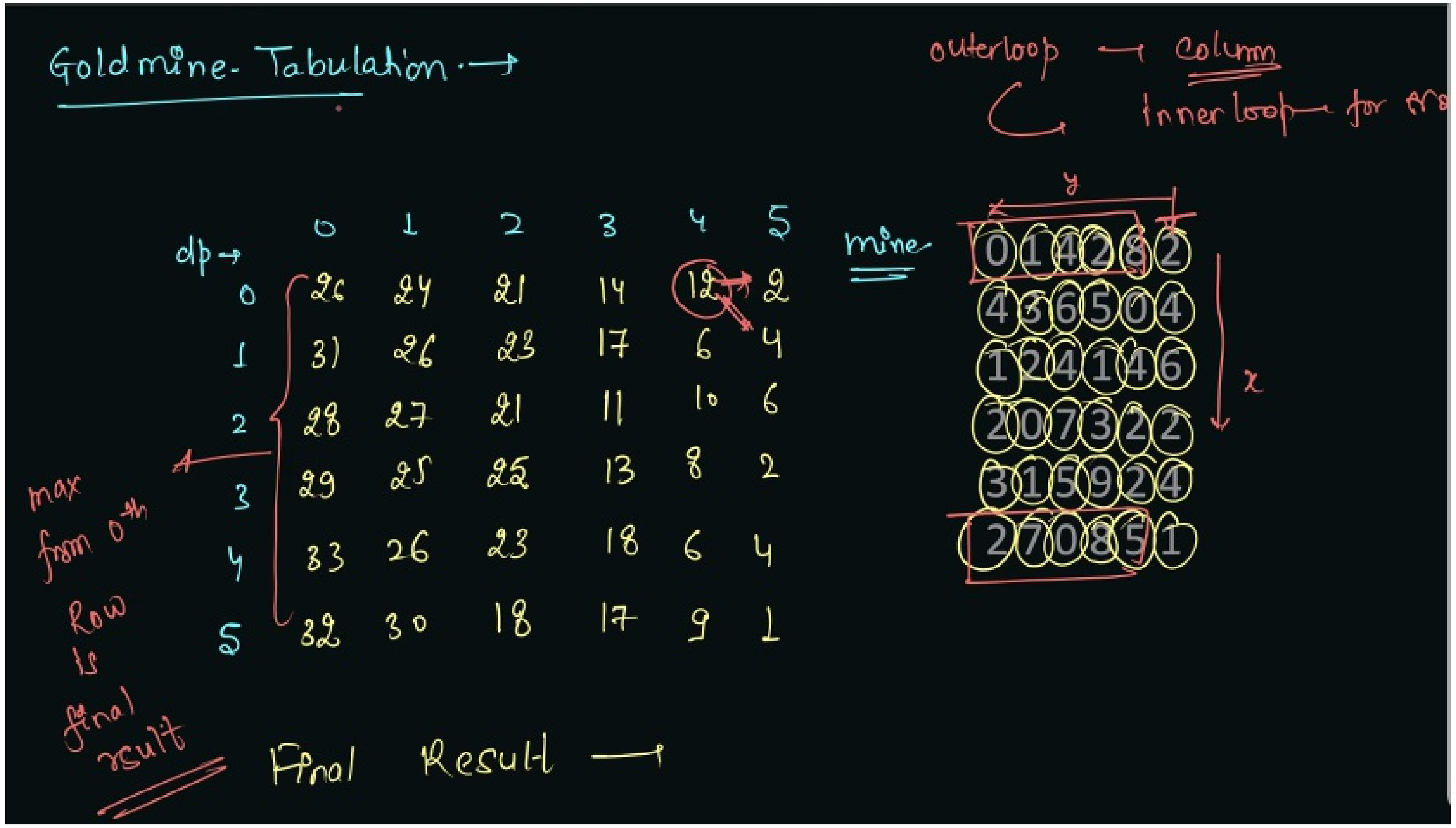
Memoisation.

final Result = max(7, 8, 11) = $\boxed{11}$ Ans





dp→	0	1	2
0	7	5	4
1	8	7	2
2	11	5	3



Goldmine- Tabulation →

outerloop → column

innerloop → for ro

dp →

	0	1	2	3	4	5
0	26	24	21	14	12	2
1	31	26	23	17	6	4
2	28	27	21	11	10	6
3	29	25	25	13	8	2
4	33	26	23	18	6	4
5	32	30	18	17	9	1

max from 0th Row is final result

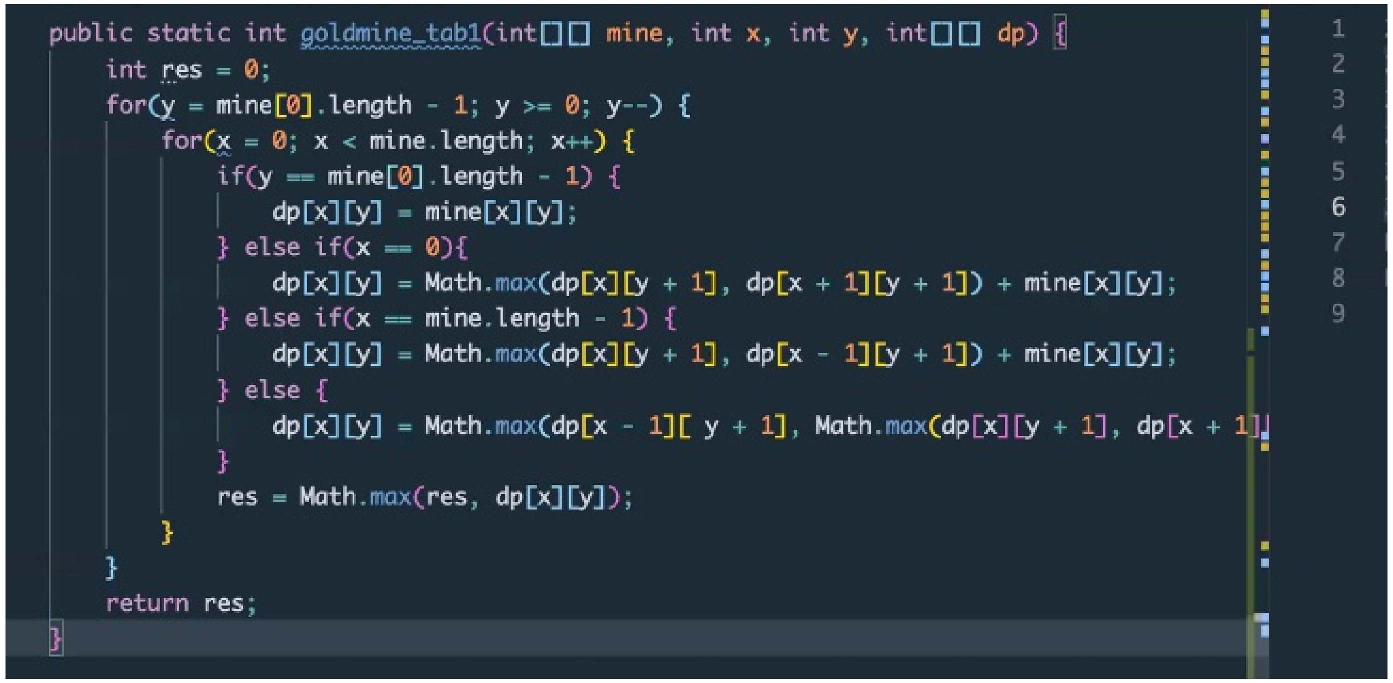
mine

y

0	1	4	2	8	2
4	3	6	5	0	4
1	2	4	1	4	6
2	0	7	3	2	2
3	1	5	9	2	4
2	7	0	8	5	1

x

Final Result →



```java
public static int goldmine_tab1(int[][] mine, int x, int y, int[][] dp) {
    int res = 0;
    for(y = mine[0].length - 1; y >= 0; y--) {
        for(x = 0; x < mine.length; x++) {
            if(y == mine[0].length - 1) {
                dp[x][y] = mine[x][y];
            } else if(x == 0){
                dp[x][y] = Math.max(dp[x][y + 1], dp[x + 1][y + 1]) + mine[x][y];
            } else if(x == mine.length - 1) {
                dp[x][y] = Math.max(dp[x][y + 1], dp[x - 1][y + 1]) + mine[x][y];
            } else {
                dp[x][y] = Math.max(dp[x - 1][ y + 1], Math.max(dp[x][y + 1], dp[x + 1]
            }
            res = Math.max(res, dp[x][y]);
        }
    }
    return res;
}
```

Target Sum Subset - DP

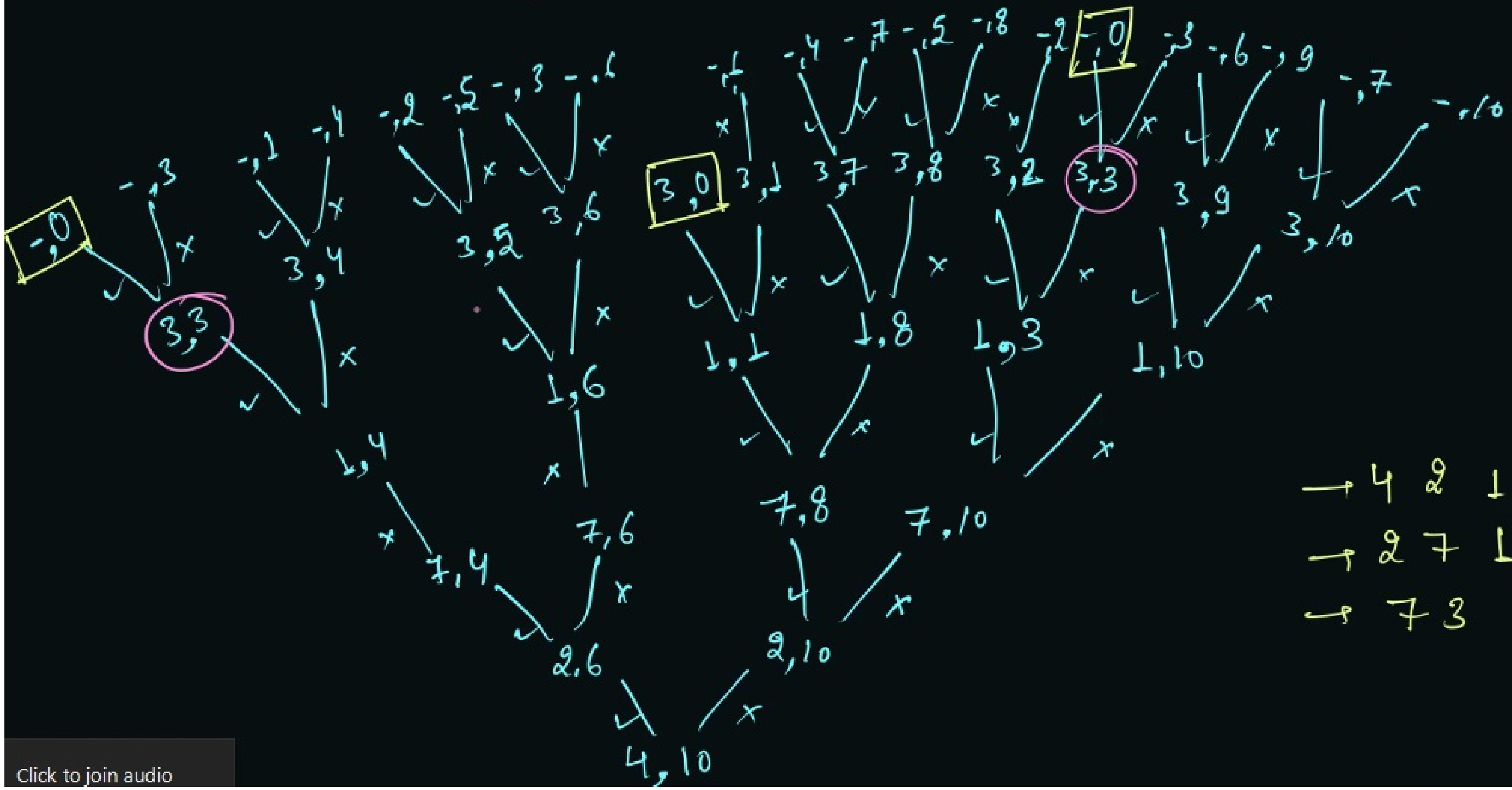
Saturday, 26 June 2021 6:54 PM

Given → $\{\overset{0}{4}, \overset{1}{2}, \overset{2}{7}, \overset{3}{1}, \overset{4}{3}\}$ ⟶

target = $\boxed{10}$ True / False

□ → Base case

□ → Repitition

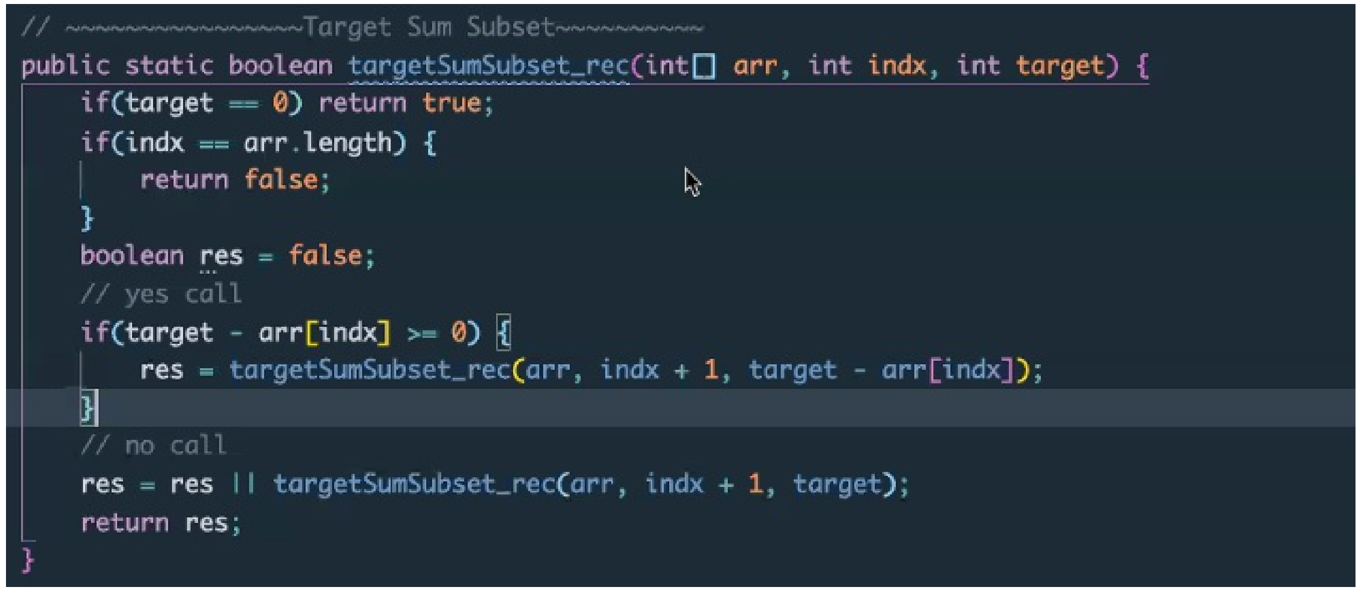


→ 4 2 1 3

→ 2 7 1

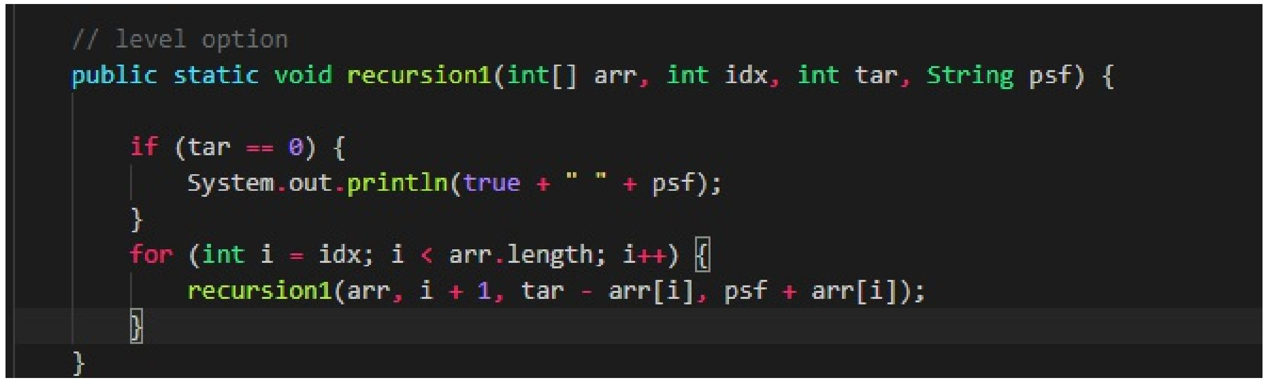
→ 7 3

Click to join audio



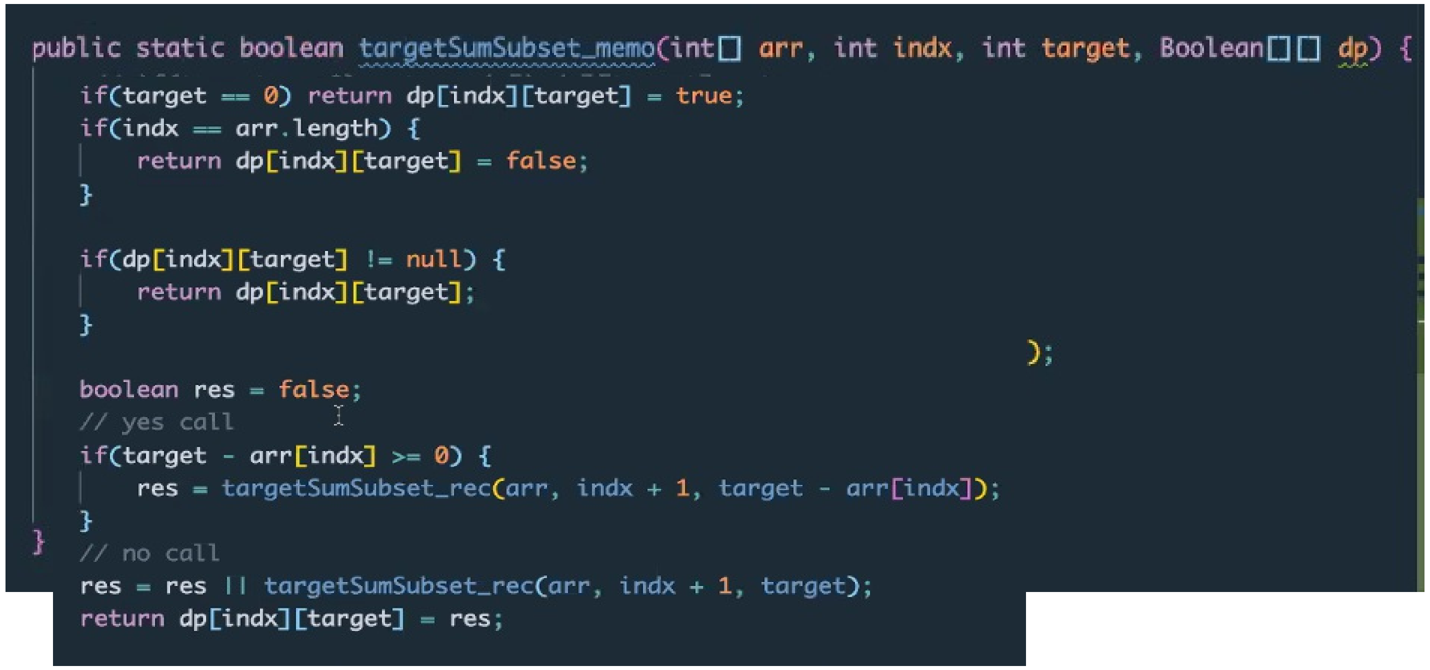
```java
// ~~~~~~~~~~~~~~~~~Target Sum Subset~~~~~~~~~~
public static boolean targetSumSubset_rec(int[] arr, int indx, int target) {
    if(target == 0) return true;
    if(indx == arr.length) {
        return false;
    }
    boolean res = false;
    // yes call
    if(target - arr[indx] >= 0) {
        res = targetSumSubset_rec(arr, indx + 1, target - arr[indx]);
    }
    // no call
    res = res || targetSumSubset_rec(arr, indx + 1, target);
    return res;
}
```

n=5
4 2 7 1 3
tar= 10

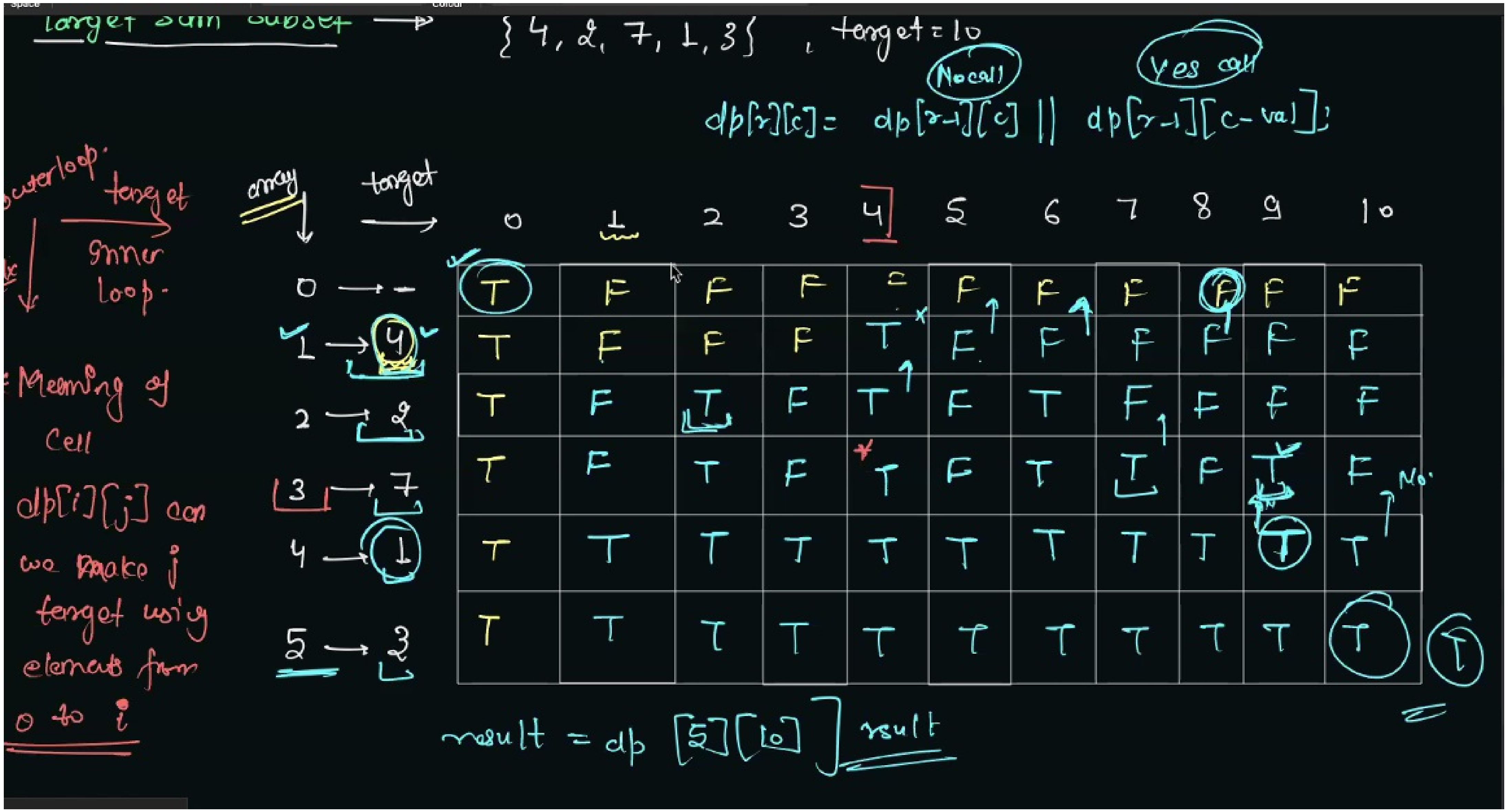


```java
// level option
public static void recursion1(int[] arr, int idx, int tar, String psf) {

    if (tar == 0) {
        System.out.println(true + " " + psf);
    }
    for (int i = idx; i < arr.length; i++) {
        recursion1(arr, i + 1, tar - arr[i], psf + arr[i]);
    }
}
```

true 4213
true 271
true 73



```java
public static boolean targetSumSubset_memo(int[] arr, int indx, int target, Boolean[][] dp) {
    if(target == 0) return dp[indx][target] = true;
    if(indx == arr.length) {
        return dp[indx][target] = false;
    }

    if(dp[indx][target] != null) {
        return dp[indx][target];
    }
                                                                              );
    boolean res = false;
    // yes call
    if(target - arr[indx] >= 0) {
        res = targetSumSubset_rec(arr, indx + 1, target - arr[indx]);
    }
}   // no call
    res = res || targetSumSubset_rec(arr, indx + 1, target);
    return dp[indx][target] = res;
```



Target sum subset → {4, 2, 7, 1, 3} , target = 10

No call   Yes call

$dp[r][c] = dp[r-1][c] \mid\mid dp[r-1][c-val];$

outerloop. target → inner loop.

Meaning of Cell

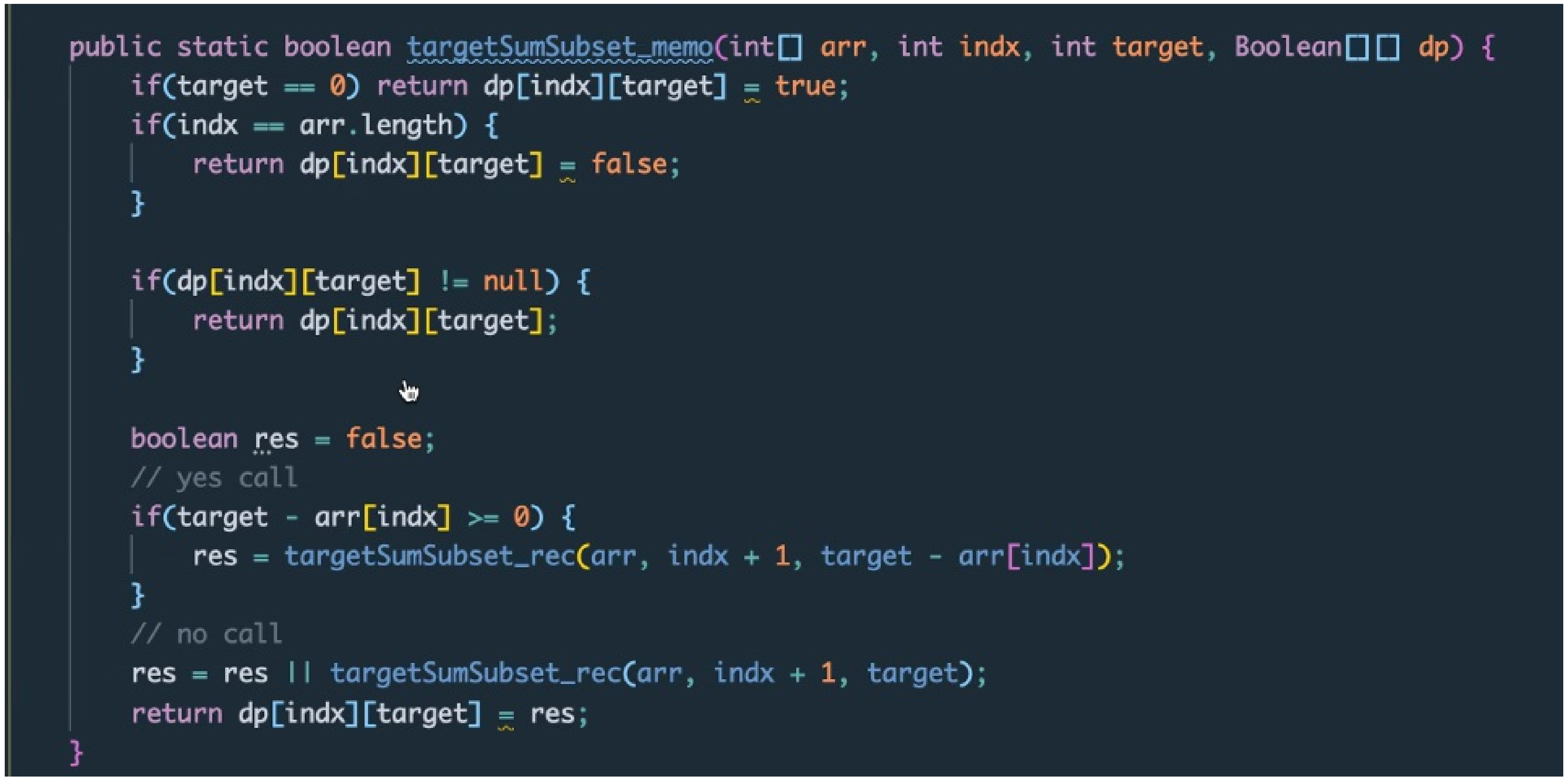
dp[i][j] can we make j target using elements from 0 to i

array ↓ target →	0	1	2	3	4	5	6	7	8	9	10
0 → -	T	F	F	F	F	F	F	F	F	F	F
1 → 4	T	F	F	F	T	F	F	F	F	F	F
2 → 2	T	F	T	F	T	F	T	F	F	F	F
3 → 7	T	F	T	F	T	F	T	T	F	T	F
4 → 1	T	T	T	T	T	T	T	T	T	T	T
5 → 3	T	T	T	T	T	T	T	T	T	T	T

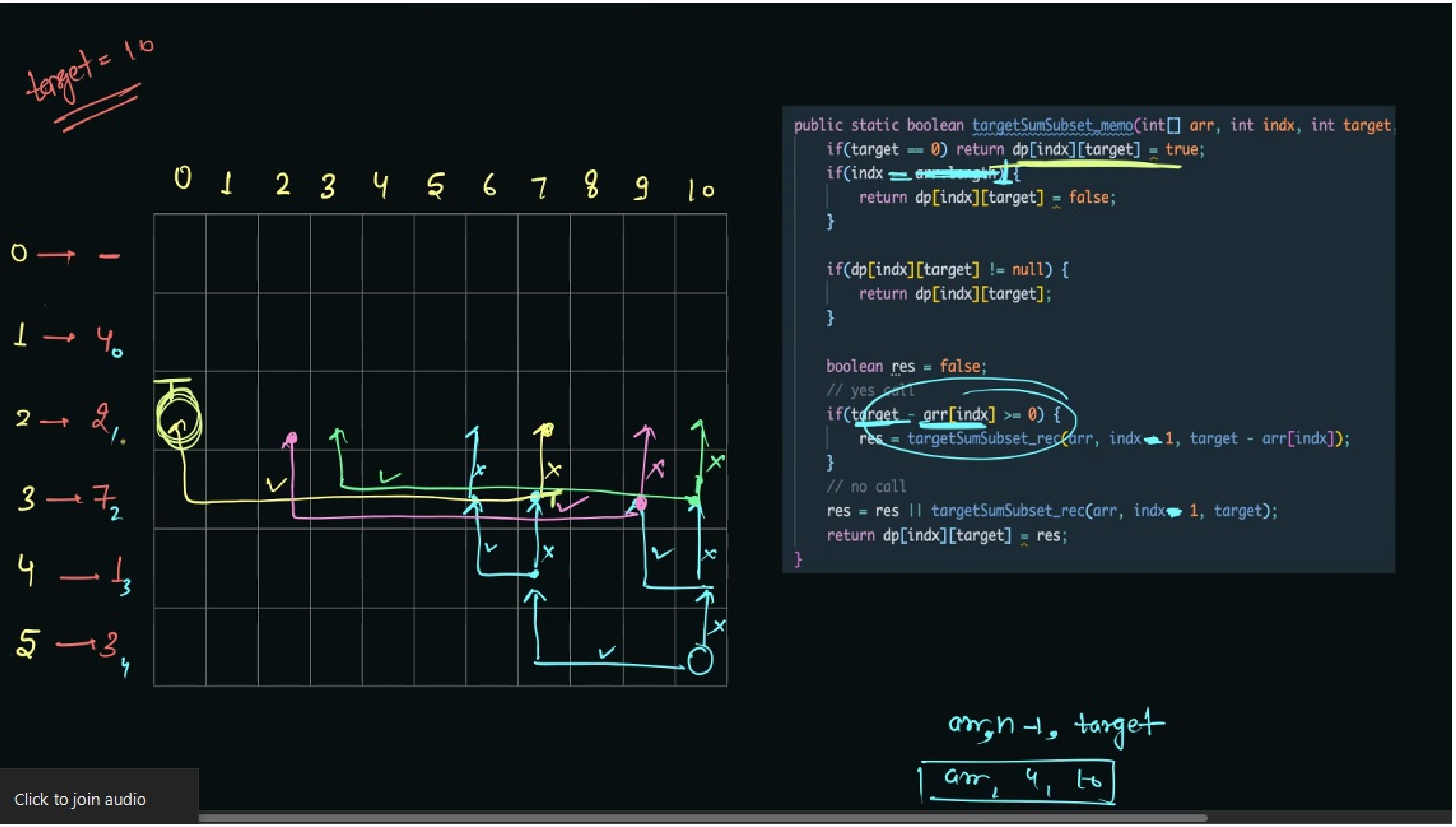
No.

T

result = dp [5][10] result



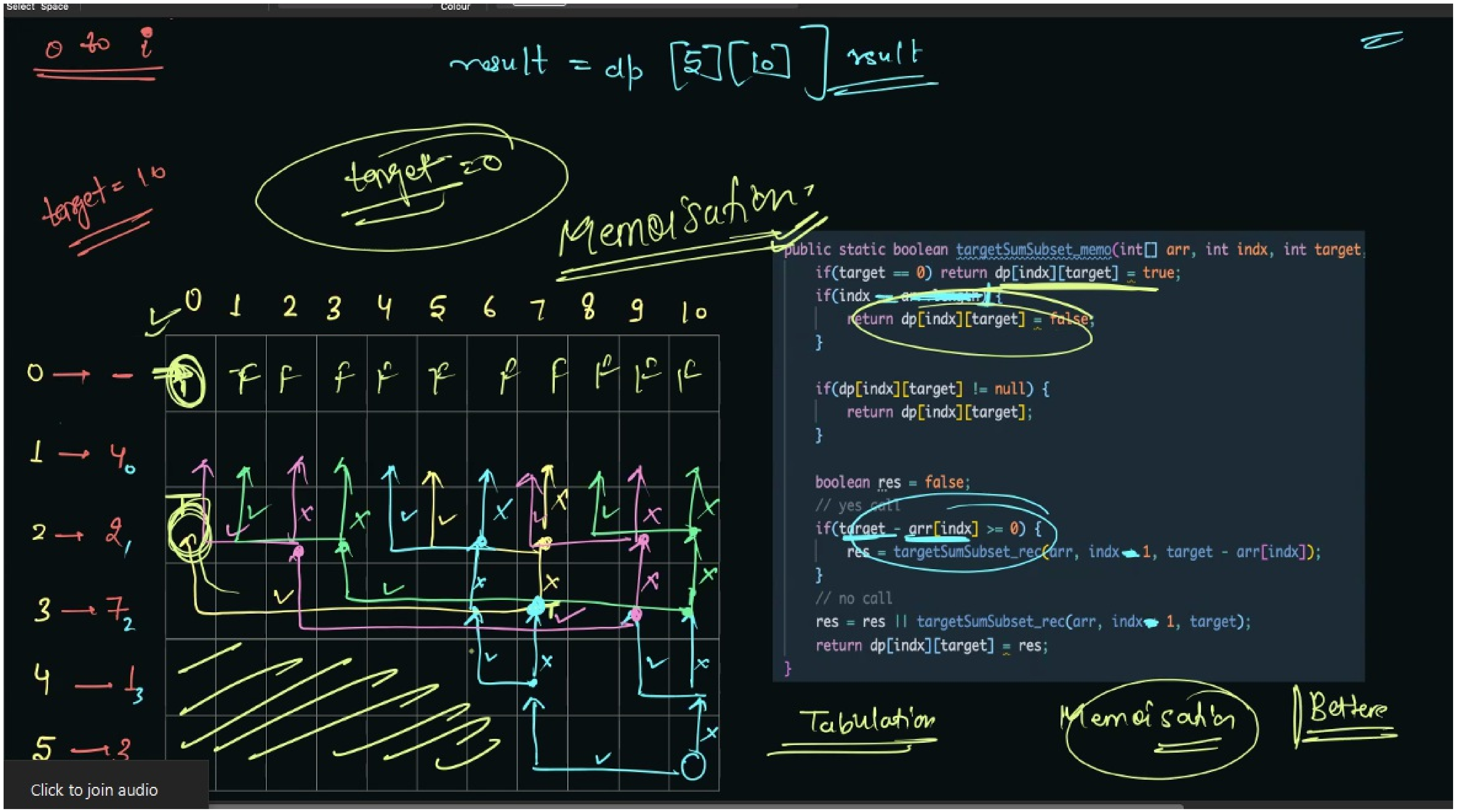
```
public static boolean targetSumSubset_memo(int[] arr, int indx, int target, Boolean[][] dp) {
    if(target == 0) return dp[indx][target] = true;
    if(indx == arr.length) {
        return dp[indx][target] = false;
    }

    if(dp[indx][target] != null) {
        return dp[indx][target];
    }

    boolean res = false;
    // yes call
    if(target - arr[indx] >= 0) {
        res = targetSumSubset_rec(arr, indx + 1, target - arr[indx]);
    }
    // no call
    res = res || targetSumSubset_rec(arr, indx + 1, target);
    return dp[indx][target] = res;
}
```



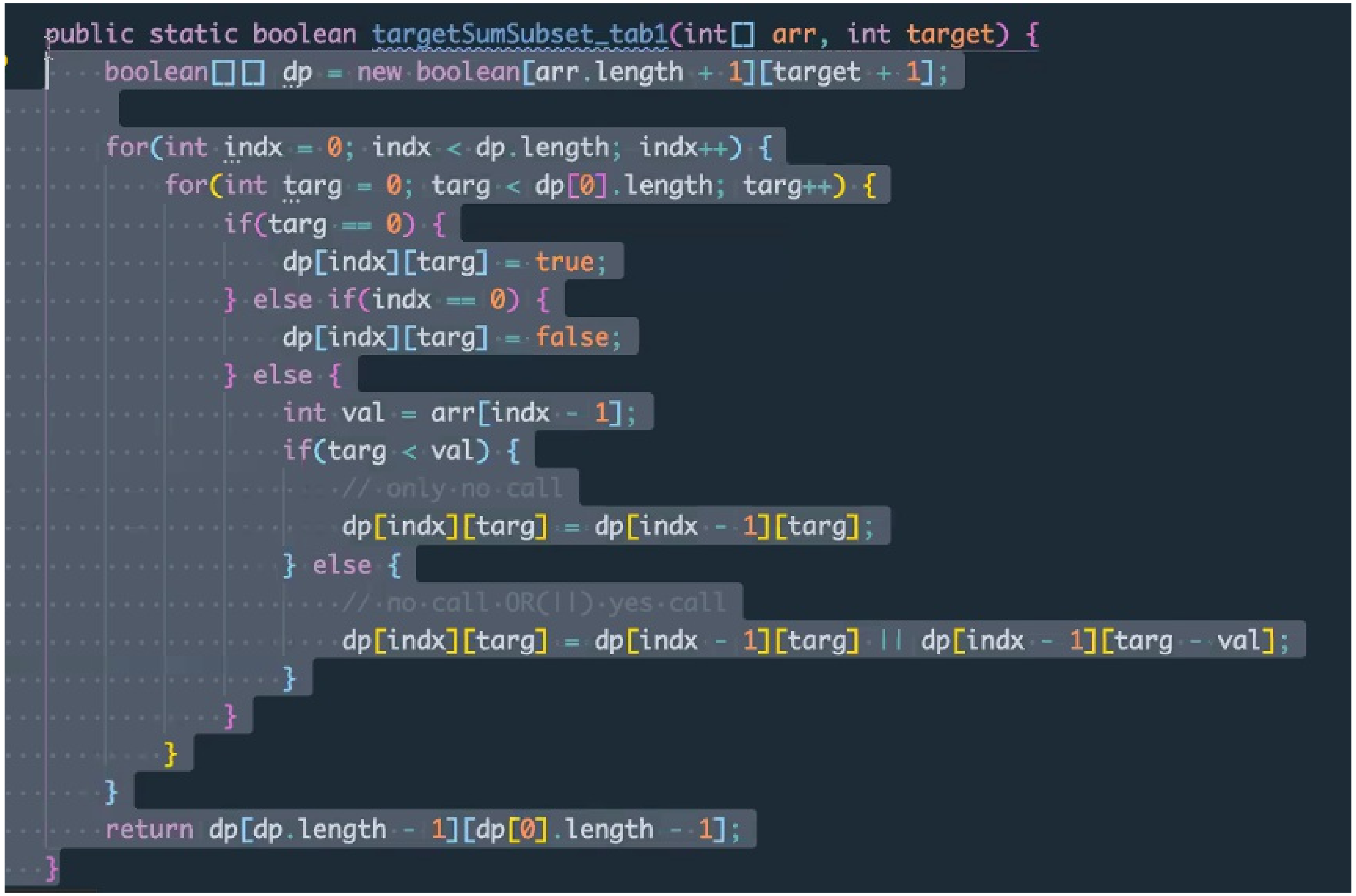
```java
public static boolean targetSumSubset_memo(int[] arr, int indx, int target,
    if(target == 0) return dp[indx][target] = true;
    if(indx == arr.length) {
        return dp[indx][target] = false;
    }

    if(dp[indx][target] != null) {
        return dp[indx][target];
    }

    boolean res = false;
    // yes call
    if(target - arr[indx] >= 0) {
        res = targetSumSubset_rec(arr, indx + 1, target - arr[indx]);
    }
    // no call
    res = res || targetSumSubset_rec(arr, indx + 1, target);
    return dp[indx][target] = res;
}
```

arr, n - 1, target

$\boxed{\text{arr, 4, 10}}$

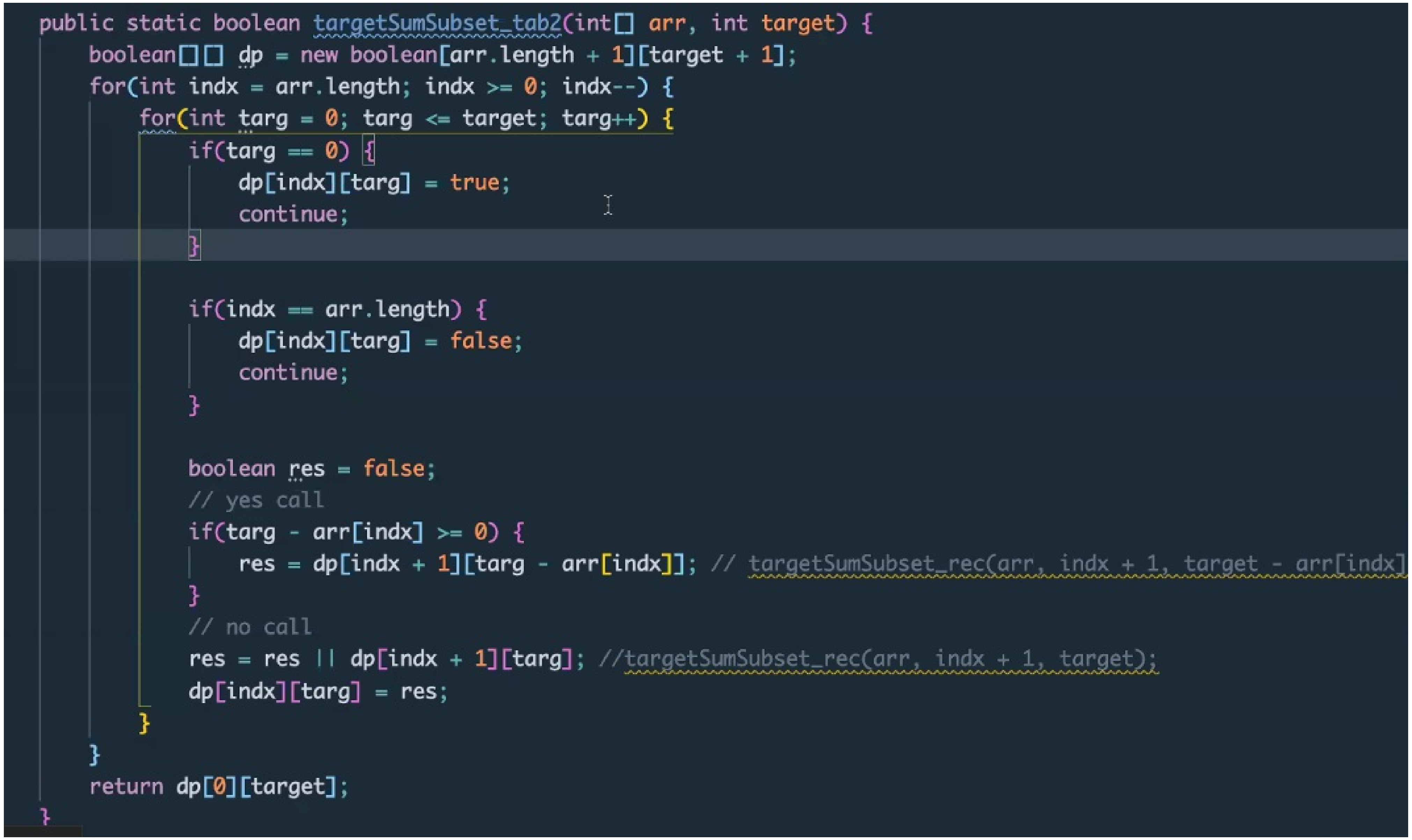


```java
public static boolean targetSumSubset_memo(int[] arr, int indx, int target,
    if(target == 0) return dp[indx][target] = true;
    if(indx == arr.length) {
        return dp[indx][target] = false;
    }

    if(dp[indx][target] != null) {
        return dp[indx][target];
    }

    boolean res = false;
    // yes call
    if(target - arr[indx] >= 0) {
        res = targetSumSubset_rec(arr, indx + 1, target - arr[indx]);
    }
    // no call
    res = res || targetSumSubset_rec(arr, indx + 1, target);
    return dp[indx][target] = res;
}
```



```java
public static boolean targetSumSubset_tab1(int[] arr, int target) {
    boolean[][] dp = new boolean[arr.length + 1][target + 1];

    for(int indx = 0; indx < dp.length; indx++) {
        for(int targ = 0; targ < dp[0].length; targ++) {
            if(targ == 0) {
                dp[indx][targ] = true;
            } else if(indx == 0) {
                dp[indx][targ] = false;
            } else {
                int val = arr[indx - 1];
                if(targ < val) {
                    // only no call
                    dp[indx][targ] = dp[indx - 1][targ];
                } else {
                    // no call OR(||) yes call
                    dp[indx][targ] = dp[indx - 1][targ] || dp[indx - 1][targ - val];
                }
            }
        }
    }
    return dp[dp.length - 1][dp[0].length - 1];
}
```

mem to tab



```java
public static boolean targetSumSubset_tab2(int[] arr, int target) {
    boolean[][] dp = new boolean[arr.length + 1][target + 1];
    for(int indx = arr.length; indx >= 0; indx--) {
        for(int targ = 0; targ <= target; targ++) {
            if(targ == 0) {
                dp[indx][targ] = true;
                continue;
            }

            if(indx == arr.length) {
                dp[indx][targ] = false;
                continue;
            }

            boolean res = false;
            // yes call
            if(targ - arr[indx] >= 0) {
                res = dp[indx + 1][targ - arr[indx]]; // targetSumSubset_rec(arr, indx + 1, target - arr[indx]
            }
            // no call
            res = res || dp[indx + 1][targ]; //targetSumSubset_rec(arr, indx + 1, target);
            dp[indx][targ] = res;
        }
    }
    return dp[0][target];
}
```