“

Dynamic Programming

Video - 131

Note :- This playlist is only for explanation of Qns & solutions.

See my "DP Concepts & Qns" playlist for understanding DP from scratch...

Leetcode - 3651 Hard

codestorywithmiK

CSwithMIK

codestorywithMIK

← My life behind scen
+
Tech News/updates

Motivation :-

Whatever happens,
just Don't Quit.

3651. Minimum Cost Path with Teleportations

Hard | Topics | Companies | Hint

You are given a `m x n` 2D integer array `grid` and an integer `k`. You start at the top-left cell `(0, 0)` and your goal is to reach the bottom-right cell `(m - 1, n - 1)`.

There are two types of moves available:

Cost →
Cost[i][j+1]

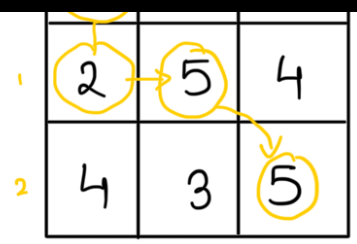
- **Normal move**: You can move right or down from your current cell `(i, j)`, i.e. you can move to `(i, j + 1)` (right) or `(i + 1, j)` (down). The cost is the value of the destination cell.
- **Teleportation**: You can teleport from any cell `(i, j)`, to any cell `(x, y)` such that `grid[x][y] <= grid[i][j]`; the cost of this move is 0. You may teleport at most `k` times. ← K

Return the **minimum** total cost to reach cell `(m - 1, n - 1)` from `(0, 0)`.

Example:- grid =

	0	1	2
0	1	3	3

, K = 2



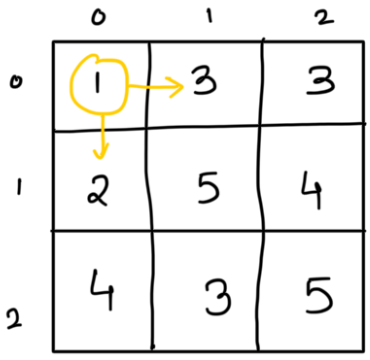
2+5 = 7
using
1 skip

Output : 7

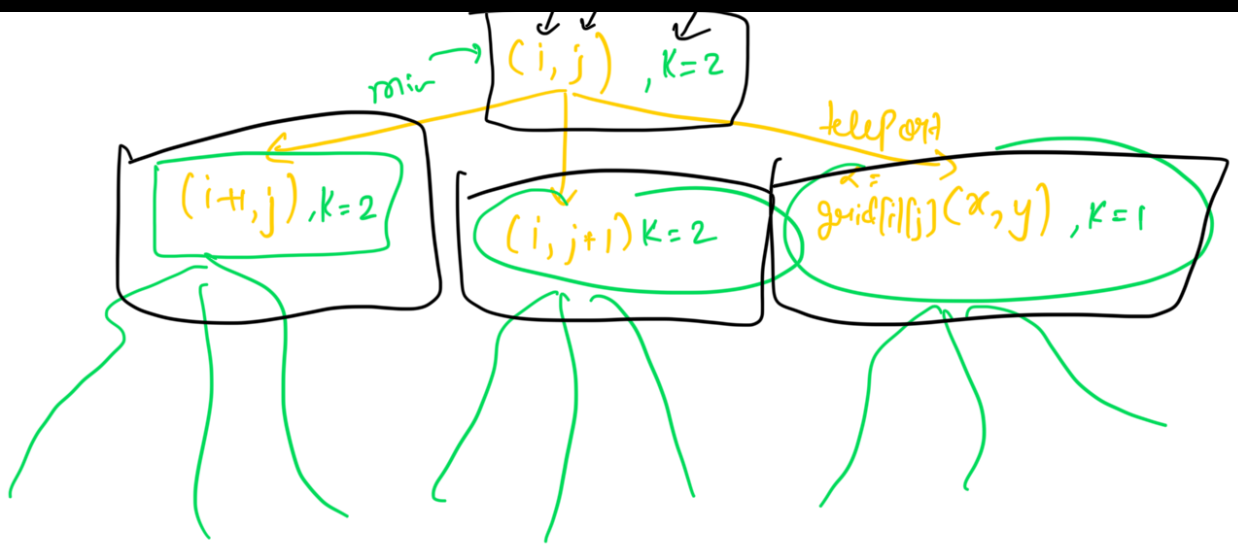
Constraints:

- `2 <= m, n <= 80`
- `m == grid.length`
- `n == grid[i].length`
- `0 <= grid[i][j] <= 10^4`
- `0 <= k <= 10`

Thought Process



K = 2



(m-1)(n-1)

Memoiz. [ ][ ][ ]

3D - DP

Solve (0, 0, 0, K, grid);

Solve (i, j, t, K, grid) {

if (i == m-1 && j == n-1) {

return 0;

}

int result = 1e9;

// Right (i,j) → (i,j+1)

grid[i][j+1];

if (j+1 < n) {

int next = Solve (i, j+1, t, K, grid);

result = min(result, grid[i][j+1] + next);

}

// (i,j) → (i+1, j)

if (i+1 < m) {

```
        int next = Solve(i+1, j, t, K, grid);
        result = min(result, grid[i+1][j] + next);
    }

    // teleport (i,j) → (x,y)
    if( t < K ) {
        currval = grid[i][j];
        for(int x=0; x<m; x++)
            for(int y=0; y<n; y++) {
                if( x == i && y == j )
                    continue;
                if(grid[x][y] <= grid[i][j]) {
                    result = min(result,
                        Solve(x, y, t+1, K, grid));
                }
    }

    return result;
}
```

[m] [n] [K]

T.C = m.n.K * (m*n)

= O($m^2 . n^2 . K$) T.L.E.

Bottom up :-

```cpp
int solve(int i, int j, int tPort, int k, vector<vector<int>>& grid) {
    // Reached destination
    if (i == m - 1 && j == n - 1)
        return 0;

    if (t[i][j][tPort] != -1)
        return t[i][j][tPort];

    int result = 1e9; //Large value
    int curVal = grid[i][j];

    // Move Right
    if (j + 1 < n) {
        int next = solve(i, j + 1, tPort, k, grid);
        result = min(result, grid[i][j + 1] + next);
    }

    // Move Down
    if (i + 1 < m) {
        int next = solve(i + 1, j, tPort, k, grid);
        result = min(result, grid[i + 1][j] + next);
    }

    // Teleport
    if (tPort < k) {
        for (int x = 0; x < m; x++) {
            for (int y = 0; y < n; y++) {
                if ((x != i || y != j) && grid[x][y] <= curVal) {
                    result = min(result, solve(x, y, tPort + 1, k, grid)); // cost
                }
            }
        }
    }

    return t[i][j][tPort] = result;
}
```

dp = [m+1] [n+1] [K+1]

dp[i][j][tPort] = min cost to reach (m-1),(n-1) from (i,j) having used tPort

dp[i][j+1][tPort]

dp[i+1][j][tPort]

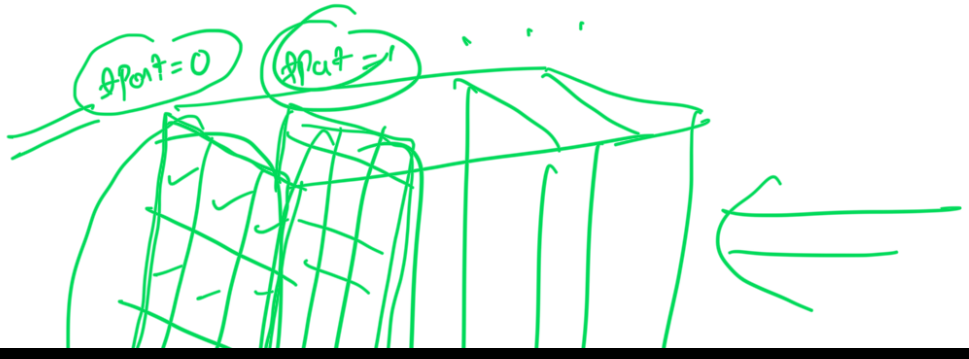
(i) [j] [tPort] =

dp[x][y][tPort+1]

(1)

```
for( t = 0 ; t<=K ; t++) {
    dp[m-1][n-1][t] = 0 ;
}
```

(2)

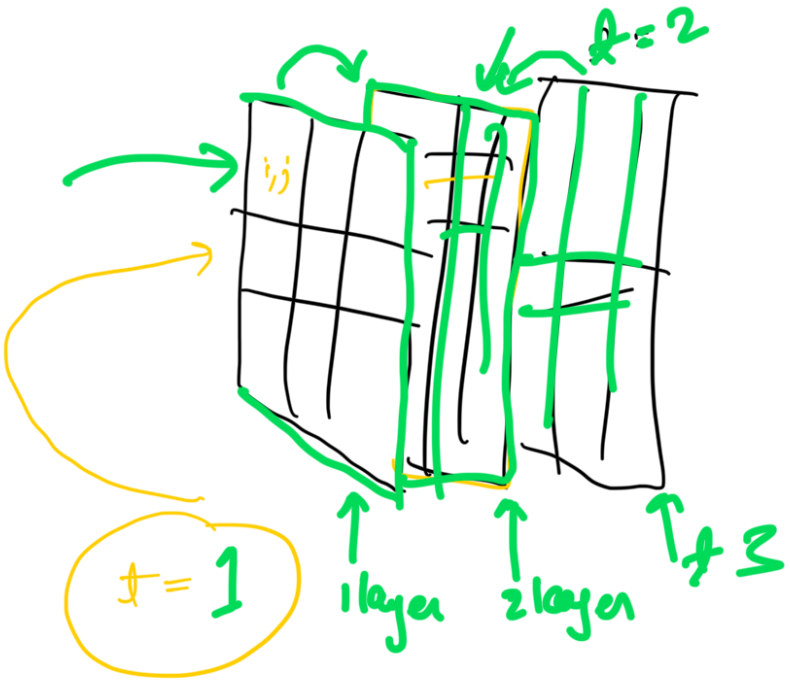


```
for (kport = K; kport >= 0; kport--) {

    for (int i = m-1; i >= 0; i--) {

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

            if (i == m-1 && j == n-1) {
                continue;
            }

            int result = 1e9;
                cost = grid[i][j];
```

T.L.E.

```
    }
}

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

How to further optimize it ???

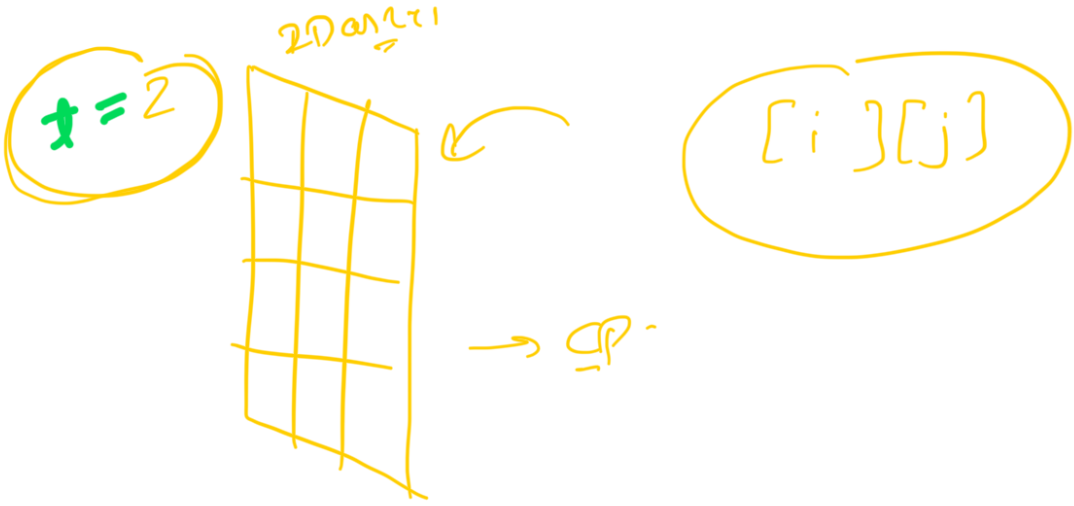


3D array.
=
[i][j]

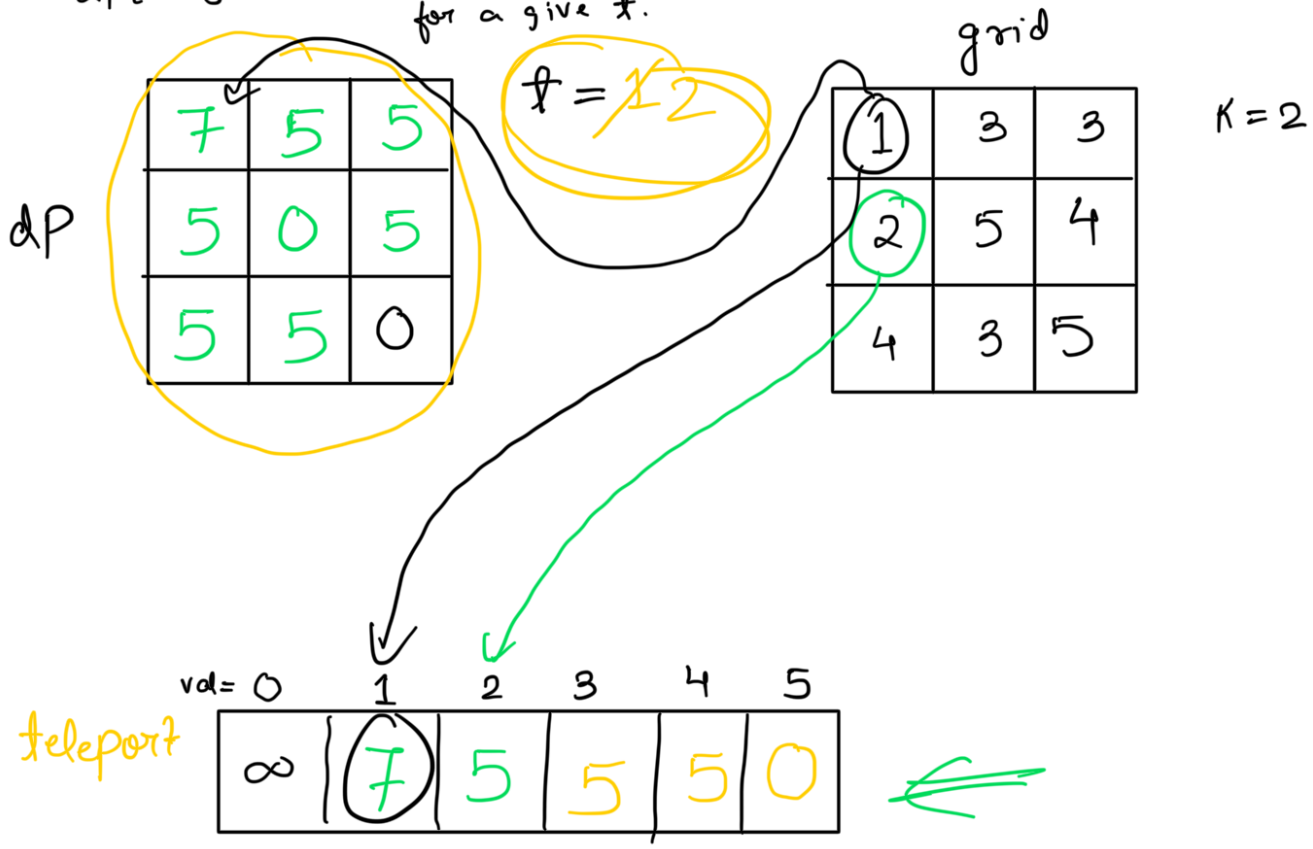
"Layered D.P"

↓

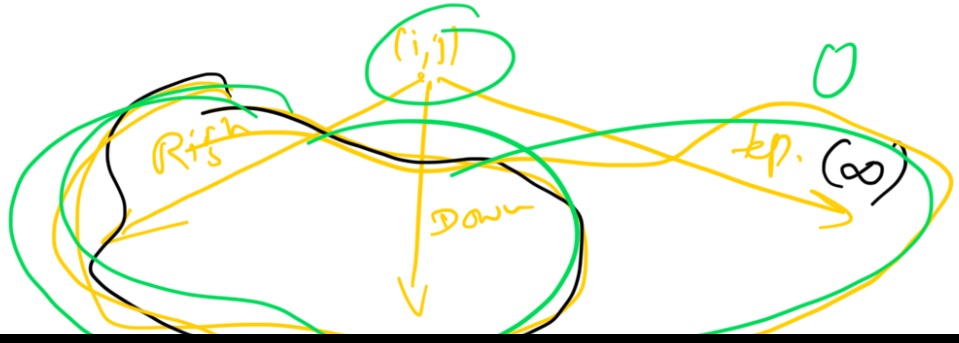
It's just a DP technique in which we solve the same problem multiple times, each time allowing one more unit of a limite resource (telep -)



dp[i][j] = min cost (i,j) → (m-1, n-1)
for a give t.



teleport [val] =



0

```
for (t=0; t<=k; t++) {                                   → O(k)

    for (i=m-1; i>=0; i--)                               → O(m)
        for (j=n-1; j>=0; j--) {                         O(n)
            //
            if(i+1<m) {
                dp[i][j] = min(dp[i][j], dp[i+1][j]
                    + grid[i+1][j];
            }

            if(j+1<n) {
                dp[i][j] = min(dp[i][j], dp[i][j+1] +
                    grid[i][j+1];
            }

            if(t>0) {
                dp[i][j] = min(dp[i][j], teleport
                    [grid[i][j]];
            }
        }

    for(int i=0; i<m; i++) {                             O(m·n)
        for(int j=0; j<n; j++) {
            teleport[grid[i][j]] = min(teleport[grid[i][j]],
                dp[i][j]);
        }
    }
```

```
// Prefix.min -> Correctly update the dp
for(int i=1; i<temp.size(); i++) {
    temp[i] = min(temp[i], temp[i-1]);
}
}
return t[0][0];
```

$$T.C = O(K \cdot (m \cdot n + m \cdot n))$$

$$T.C = O(K \cdot m \cdot n)$$

What went wrong ?

dp

5	4
4	0

dp

t = 1

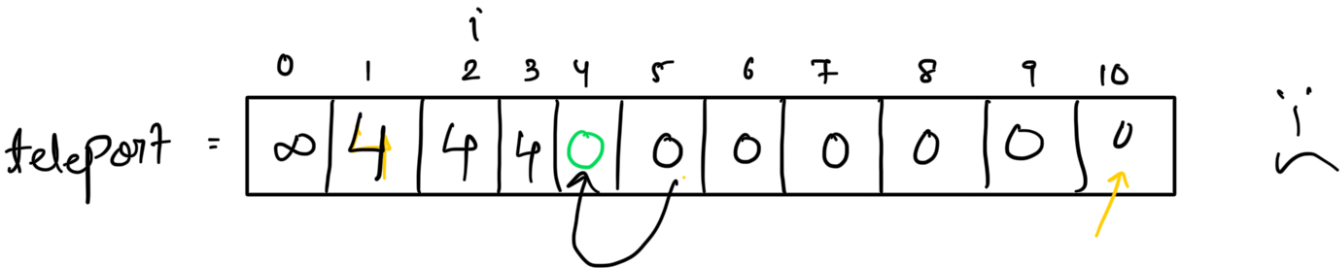
grid

3	1
10	4

K = 2

0 + 4 = 4

0


i
0 1 2 3 4 5 6 7 8 9 10
teleport =
∞ 4 4 4 0 0 0 0 0 0 0
i