GoldMine in C++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int grid[4][4] = {
        {8, 2, 1, 6},
        {6, 5, 5, 2},
        {2, 1, 0, 3},
        {7, 2, 2, 4}
    };

    int n = 4; // Number of rows
    int m = 4; // Number of columns

    // Initialize dp array
    vector<vector<int>> dp(n, vector<int>(m, 0));

    // Fill dp array from rightmost column to left
    for (int j = m - 1; j >= 0; j--) {
        for (int i = n - 1; i >= 0; i--) {
            if (j == m - 1) {
                dp[i][j] = grid[i][j];
            } else if (i == n - 1) {
                dp[i][j] = grid[i][j] + max(dp[i][j + 1], dp[i - 1][j +
1]);
            } else if (i == 0) {
                dp[i][j] = grid[i][j] + max(dp[i][j + 1], dp[i + 1][j
+ 1]);
            } else {
                dp[i][j] = grid[i][j] + max(dp[i][j + 1], max(dp[i -
1][j + 1], dp[i + 1][j + 1]));
            }
        }
    }

    // Find the maximum value in the first column of dp
array
    int maxGold = dp[0][0];
    for (int i = 1; i < n; i++) {
        if (dp[i][0] > maxGold) {
            maxGold = dp[i][0];
        }
    }

    cout << maxGold << endl;

    return 0;
}
```

Let's do a **tabular dry run** of your gold mine problem (classic DP), where the goal is to find the **maximum gold** that can be collected from **any cell in the first column** to the **last column**, moving only in:

- right (→)
- right-up (↗)
- right-down (↘)

🧮 Given grid[4][4]:

[8, 2, 1, 6]
[6, 5, 5, 2]
[2, 1, 0, 3]
[7, 2, 2, 4]

🧠 DP Formula:

For dp[i][j]:

- If j == last column: dp[i][j] = grid[i][j]
- If i == 0: no up → use right and right-down
- If i == n-1: no down → use right and right-up
- Else: consider all 3 → right, right-up, right-down

🔄 Filling dp from right to left:

We'll fill the DP matrix from column j = 3 to 0.

Step-by-step (column by column):

i\j	0	1	2	3
0	?	?	?	6
1	?	?	?	2
2	?	?	?	3
3	?	?	?	4

Fill Column 2 (j = 2):

dp[i][2] = grid[i][2] + max(dp[i][3], dp[i-1][3], dp[i+1][3])

i	grid[i][2]	dp options	max	dp[i][2]
3	2	dp[3][3]=4, dp[2][3]=3	4	6
2	0	3, 2, 4	4	4

i	grid[i][2]	dp options	max	dp[i][2]
1	5	2, 6, 3	6	11
0	1	6, 2	6	7

Fill Column 1 (j = 1):

i	grid[i][1]	dp options	max	dp[i][1]
3	2	6, 4	6	8
2	1	4, 11, 6	11	12
1	5	11, 7, 4	11	16
0	2	7, 11	11	13

Fill Column 0 (j = 0):

i	grid[i][0]	dp options	max	dp[i][0]
3	7	8, 12	12	19
2	2	12, 16, 8	16	18
1	6	16, 13, 12	16	22
0	8	13, 16	16	24

✅ Final dp Table:

i\j	0	1	2	3
0	24	13	7	6
1	22	16	11	2
2	18	12	4	3
3	19	8	6	4

📜 Max Gold = max(dp[0][0], dp[1][0], dp[2][0], dp[3][0]) = 24

✅ Output:

24

Output:
24