Print all path with max gold In C++

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct Pair {
    int i, j;
    string psf;

   Pair(int i, int j, string psf) {
        this->i = i;
        this->j = j;
        this->psf = psf;
    }
};

void
printMaxGoldPath(vector<vector<int>>&
arr) {
    int m = arr.size();
    int n = arr[0].size();

    // dp array to store maximum gold
collected to reach each cell
    vector<vector<int>> dp(m,
vector<int>(n, 0));

    // Initialize dp array for the last column
    for (int i = 0; i < m; i++) {
        dp[i][n - 1] = arr[i][n - 1];
    }

    // Fill dp array using dynamic
programming approach
    for (int j = n - 2; j >= 0; j--) {
        for (int i = 0; i < m; i++) {
            int maxGold = dp[i][j + 1]; //
Maximum gold by going right from current
cell
            if (i > 0) {
                maxGold = max(maxGold, dp[i -
1][j + 1]); // Maximum gold by going
diagonal-up-right
            }
            if (i < m - 1) {
                maxGold = max(maxGold, dp[i +
1][j + 1]); // Maximum gold by going
diagonal-down-right
            }
            dp[i][j] = arr[i][j] + maxGold; //
Total gold collected to reach current cell
        }
```

Step-by-Step Execution:

Input:

```
arr = [
    [3, 2, 3, 1],
    [2, 4, 6, 0],
    [5, 0, 1, 3],
    [9, 1, 5, 1]
]
```

Step 1: Initialize dp Array

We create a dp matrix of the same dimensions as arr and initialize the last column with the values of arr's last column:

```
dp = [
    [0, 0, 0, 1],
    [0, 0, 0, 0],
    [0, 0, 0, 3],
    [0, 0, 0, 1]
]
```

Step 2: Fill dp Array (Dynamic Programming)

We calculate the maximum gold collectible for each cell in reverse column order (from right to left):

- **Column 2 (j = 2):**
  - Row 0: dp[0][2] = arr[0][2] + max(dp[0][3], dp[1][3]) = 3 + max(1, 0) = 4
  - Row 1: dp[1][2] = arr[1][2] + max(dp[1][3], dp[0][3], dp[2][3]) = 6 + max(0, 1, 3) = 9
  - Row 2: dp[2][2] = arr[2][2] + max(dp[2][3], dp[1][3], dp[3][3]) = 1 + max(3, 0, 1) = 4
  - Row 3: dp[3][2] = arr[3][2] + max(dp[3][3], dp[2][3]) = 5 + max(1, 3) = 8
  - Updated dp:

```
dp = [
    [0, 0, 4, 1],
    [0, 0, 9, 0],
    [0, 0, 4, 3],
    [0, 0, 8, 1]
]
```

- **Column 1 (j = 1):**

```
    }

    // Find the maximum gold collected in
the first column
    int maxGold = dp[0][0];
    int maxRow = 0;
    for (int i = 1; i < m; i++) {
        if (dp[i][0] > maxGold) {
            maxGold = dp[i][0];
            maxRow = i;
        }
    }

    // Print the maximum gold collected
    cout << maxGold << endl;

    // Queue to perform BFS for path tracing
    queue<Pair> q;
    q.push(Pair(maxRow, 0,
to_string(maxRow))); // Start from the cell
with maximum gold in the first column

    // BFS to print all paths with maximum
gold collected
    while (!q.empty()) {
        Pair rem = q.front();
        q.pop();

        if (rem.j == n - 1) {
            cout << rem.psf << endl; // Print
path when reaching the last column
        } else {
            int currentGold = dp[rem.i][rem.j];
            int rightGold = dp[rem.i][rem.j + 1];
            int diagonalUpGold = (rem.i > 0) ?
dp[rem.i - 1][rem.j + 1] : -1;
            int diagonalDownGold = (rem.i < m
- 1) ? dp[rem.i + 1][rem.j + 1] : -1;

            // Add paths to queue based on the
direction with maximum gold
            if (rightGold == currentGold -
arr[rem.i][rem.j + 1]) {
                q.push(Pair(rem.i, rem.j + 1,
rem.psf + " H")); // Move horizontally to the
right
            }
            if (diagonalUpGold == currentGold
- arr[rem.i - 1][rem.j + 1]) {
                q.push(Pair(rem.i - 1, rem.j + 1,
rem.psf + " LU")); // Move diagonally up-
right
            }
```

- Row 0: dp[0][1] = arr[0][1] + max(dp[0][2], dp[1][2]) = 2 + max(4, 9) = 11
- Row 1: dp[1][1] = arr[1][1] + max(dp[1][2], dp[0][2], dp[2][2]) = 4 + max(9, 4, 4) = 13
- Row 2: dp[2][1] = arr[2][1] + max(dp[2][2], dp[1][2], dp[3][2]) = 0 + max(4, 9, 8) = 9
- Row 3: dp[3][1] = arr[3][1] + max(dp[3][2], dp[2][2]) = 1 + max(8, 4) = 9
- Updated dp:

```
dp = [
    [0, 11, 4, 1],
    [0, 13, 9, 0],
    [0,  9, 4, 3],
    [0,  9, 8, 1]
]
```

- **Column 0 (j = 0):**
  - Row 0: dp[0][0] = arr[0][0] + max(dp[0][1], dp[1][1]) = 3 + max(11, 13) = 16
  - Row 1: dp[1][0] = arr[1][0] + max(dp[1][1], dp[0][1], dp[2][1]) = 2 + max(13, 11, 9) = 15
  - Row 2: dp[2][0] = arr[2][0] + max(dp[2][1], dp[1][1], dp[3][1]) = 5 + max(9, 13, 9) = 18
  - Row 3: dp[3][0] = arr[3][0] + max(dp[3][1], dp[2][1]) = 9 + max(9, 9) = 18
  - Updated dp:

```
dp = [
    [16, 11, 4, 1],
    [15, 13, 9, 0],
    [18,  9, 4, 3],
    [18,  9, 8, 1]
]
```

Step 3: Find Maximum Gold

- The maximum gold collectible is max(dp[0][0], dp[1][0], dp[2][0], dp[3][0]) = 18.
- Starting rows for this maximum gold: Row 2 and Row 3.

Step 4: Trace All Paths Using BFS

Start BFS from the cells with maximum gold in

```
            if (diagonalDownGold ==
currentGold - arr[rem.i + 1][rem.j + 1]) {
                q.push(Pair(rem.i + 1, rem.j + 1,
rem.psf + " LD")); // Move diagonally down-
right
            }
        }
    }
}

int main() {
    vector<vector<int>> arr = {
        {3, 2, 3, 1},
        {2, 4, 6, 0},
        {5, 0, 1, 3},
        {9, 1, 5, 1}
    };

    printMaxGoldPath(arr);

    return 0;
}
```

the first column (Row 2 and Row 3):

1. **Starting from Row 2, Column 0 (psf = "2"):**
    - Move diagonally up-right (LU): dp[1][1] = 13 → New path: "2 LU".
    - Move right (H): dp[2][1] = 9 → New path: "2 H".
    - Move diagonally down-right (LD): dp[3][1] = 9 → New path: "2 LD".
2. **Starting from Row 3, Column 0 (psf = "3"):**
    - Move diagonally up-right (LU): dp[2][1] = 9 → New path: "3 LU".
    - Move right (H): dp[3][1] = 9 → New path: "3 H".

Final Output:

18

Paths:
2 LU ... (continue tracing)
2 H ...
2 LD ...
3 LU ...
3 H ...

Output:-
18