|  |  |
| --- | --- |
| **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;  } | **Dry Run**  **Input Grid:**  grid = {  {8, 2, 1, 6},  {6, 5, 5, 2},  {2, 1, 0, 3},  {7, 2, 2, 4}  }  **Steps:**   1. **Initialization**:    * n = 4 (rows), m = 4 (columns).    * Create a dp table with the same dimensions as grid. 2. **Filling DP Table**:    * Start from the last column (j = 3) and work backward to the first column (j = 0).   **Filling DP Table:**   * **Column 3 (last column)**:   dp[i][3] = grid[i][3] for all i  dp = {  {0, 0, 0, 6},  {0, 0, 0, 2},  {0, 0, 0, 3},  {0, 0, 0, 4}  }   * **Column 2**:   dp[0][2] = grid[0][2] + max(dp[0][3], dp[1][3]) = 1 + max(6, 2) = 7  dp[1][2] = grid[1][2] + max(dp[0][3], dp[1][3], dp[2][3]) = 5 + max(6, 2, 3) = 11  dp[2][2] = grid[2][2] + max(dp[1][3], dp[2][3], dp[3][3]) = 0 + max(2, 3, 4) = 4  dp[3][2] = grid[3][2] + max(dp[2][3], dp[3][3]) = 2 + max(3, 4) = 6  dp = {  {0, 0, 7, 6},  {0, 0, 11, 2},  {0, 0, 4, 3},  {0, 0, 6, 4}  }   * **Column 1**:   dp[0][1] = grid[0][1] + max(dp[0][2], dp[1][2]) = 2 + max(7, 11) = 13  dp[1][1] = grid[1][1] + max(dp[0][2], dp[1][2], dp[2][2]) = 5 + max(7, 11, 4) = 16  dp[2][1] = grid[2][1] + max(dp[1][2], dp[2][2], dp[3][2]) = 1 + max(11, 4, 6) = 12  dp[3][1] = grid[3][1] + max(dp[2][2], dp[3][2]) = 2 + max(4, 6) = 8  dp = {  {0, 13, 7, 6},  {0, 16, 11, 2},  {0, 12, 4, 3},  {0, 8, 6, 4}  }   * **Column 0**:   dp[0][0] = grid[0][0] + max(dp[0][1], dp[1][1]) = 8 + max(13, 16) = 24  dp[1][0] = grid[1][0] + max(dp[0][1], dp[1][1], dp[2][1]) = 6 + max(13, 16, 12) = 22  dp[2][0] = grid[2][0] + max(dp[1][1], dp[2][1], dp[3][1]) = 2 + max(16, 12, 8) = 18  dp[3][0] = grid[3][0] + max(dp[2][1], dp[3][1]) = 7 + max(12, 8) = 19  dp = {  {24, 13, 7, 6},  {22, 16, 11, 2},  {18, 12, 4, 3},  {19, 8, 6, 4}  }  **Final Step:**   * The maximum value in the first column (dp[i][0] for all i) is:   maxGold = max(24, 22, 18, 19) = 24 |
| Output: 24 | |