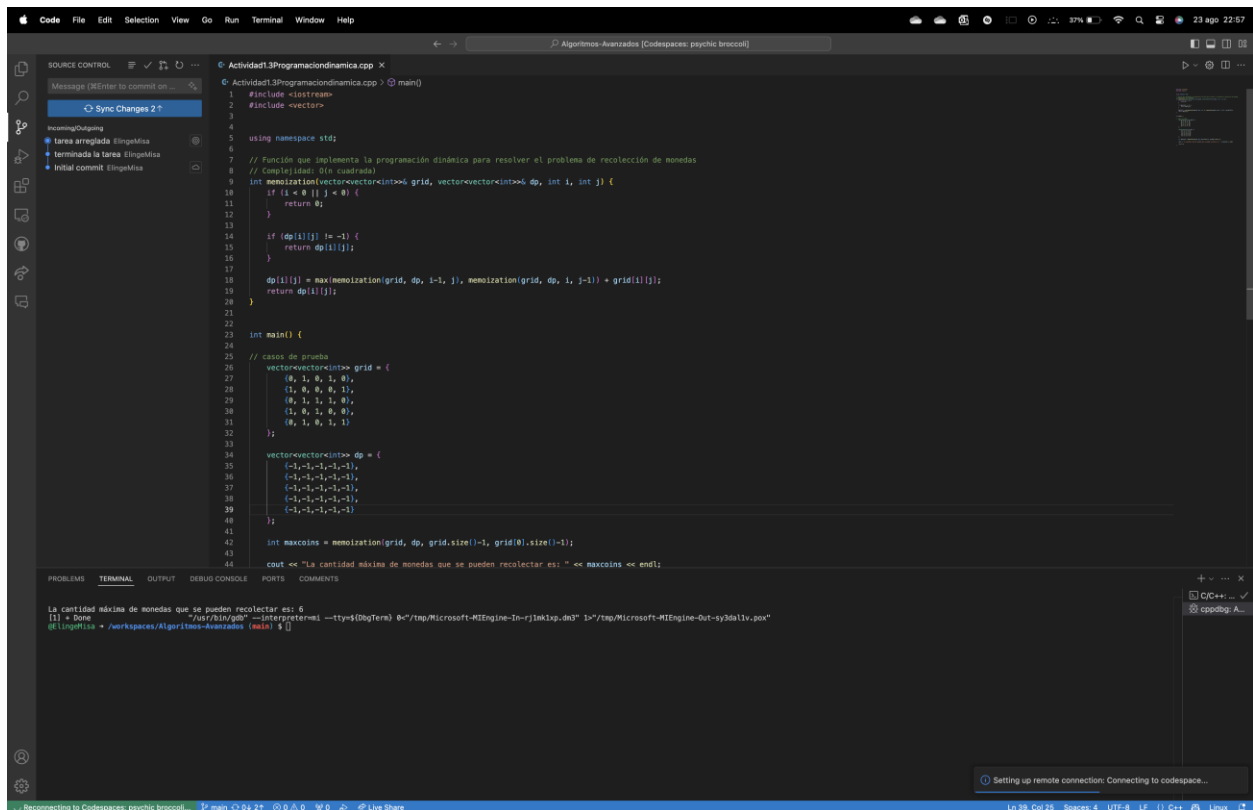


Actividad 1.3 Programación dinámica

Victor Misael Escalante Alvarado

A01741176

Screenshot del código



```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 // Función que implementa la programación dinámica para resolver el problema de recolección de monedas
7 // Complejidad: O(n*m)
8 int memorization(vector<vector<int>>& grid, vector<vector<int>>& dp, int i, int j) {
9     if (i < 0 || j < 0) {
10         return 0;
11     }
12     if (dp[i][j] != -1) {
13         return dp[i][j];
14     }
15     dp[i][j] = max(memorization(grid, dp, i-1, j), memorization(grid, dp, i, j-1)) + grid[i][j];
16     return dp[i][j];
17 }
18
19 int main() {
20     // casos de prueba
21     vector<vector<int>> grid = {
22         {0, 1, 0, 1, 0},
23         {1, 0, 0, 0, 1},
24         {0, 1, 1, 1, 0},
25         {1, 0, 1, 0, 0},
26         {0, 1, 0, 1, 1}
27     };
28     vector<vector<int>> dp = {
29         {-1, -1, -1, -1, -1},
30         {-1, -1, -1, -1, -1},
31         {-1, -1, -1, -1, -1},
32         {-1, -1, -1, -1, -1},
33         {-1, -1, -1, -1, -1}
34     };
35     int maxcoins = memorization(grid, dp, grid.size()-1, grid[0].size()-1);
36     cout << "La cantidad máxima de monedas que se pueden recolectar es: " << maxcoins << endl;
37 }
```

Analiza la relación de recurrencia del problema. ¿Porque funciona? Explícalo en tus propios términos.

Este algoritmo se basa en la relación de recurrencia de que mientras no se haya llegado al punto de inicio (0,0), seguiremos comparando las posibilidades de los 2 movimientos posibles arriba o a la izquierda, puesto que tomamos como punto de partida el final del recorrido y, junto con la memorización de las celdas pasadas, nos estamos ahorrando el recalcular de rutas, haciendo del problema algo más tratable.

Enlace a codespace de github :

<https://prod.liveshare.vsengsaas.visualstudio.com/join?421D5F4AB8397FA376C6131D0D42A35C97AF>

