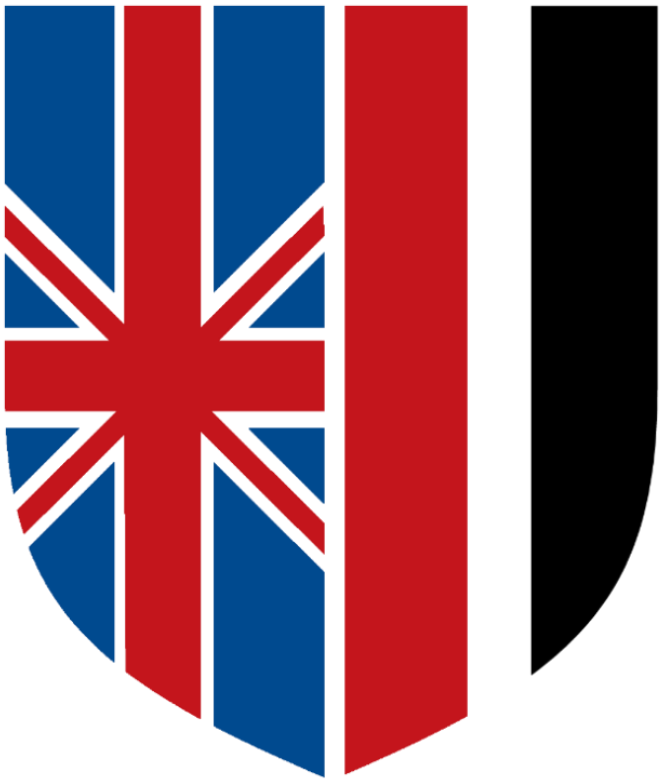


18CSCI01I

Analysis of Algorithms

Group Project Report

Group Members

Student Name	ID	Problem No.
Abdelrahman Essam	155295	Problem 1

Problem 1:

1. Pseudocode

matrix[ ][ ] (Input 0 → empty, 1 → coin, -1 → bomb) // for the first 2DArrays

New Path with coin numbers [ ][ ] (-1 → unvisited, c → coins count)

//even -> right, down

//odd -> left, down

int function → (matrix, new Path , coins so far, int x, int y)

start of the function

    if (0 > X > number of rows || if 0 > y >number of columns) → return 0

    if (grid [x] [y] = = -1) return 0 // there is a bomb

    if (new Path[x][y] != -1) → new Path[x][y] //the cell is unvisited

    if (row % 2 == 0) → step =1 //move right

    else → step = -1 //move left

    input result moving down = calc (matrix, dp, coins, x, y+1) //moving down

    input result right = calc(matrix, dp, coins, x+ step, y) //moving right or left

    mew Path [x][y] = max(result down, result right) //COMPARE STEPS

    return new Path[x][y];

end of function

2. Algorithm paradigm used:

The algorithm paradigm used to solve this program is: Dynamic Programming

What is dynamic programming ?

It's a way for approaching problem in which it memorises his work on subproblems that will be used again to avoid resolving them with a recursive relation between small and large solutions to reduce the complexity.

The algorithms saves the best path at each step(represented in number of coins) so the program doesn't have to recalculate or revisit cells, Moving from dead ends to the starting point which will contain the max number of coins (in case of using recursion) moreover, the program will have some restriction form moving that will be represented as conditions to archive the correct output.

3. Problem Complexity

At this problem the function recurse on the rows and columns for 2 directions Down and Right or Left depending on the row number.

However, the complexity should have been 2^(r*c) for using 2 calls at the function.

But by using dynamic programming and memorising the past solutions the complexity of this problem will be equal to **O(R*C)**

If it is a square matrix it could be represented as $O(n^2)$

}

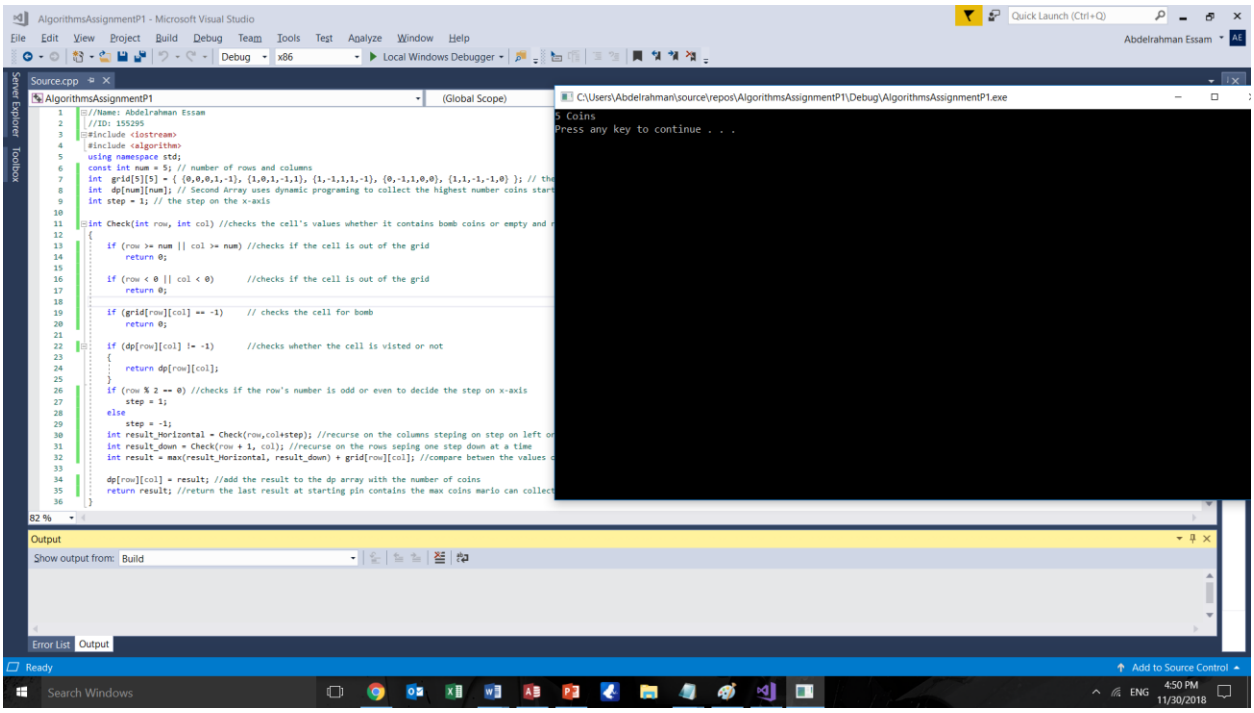
*If the problem will need the user to input the matrix cells values by him self a Dynamic 2darray or Vector will be needed in addition to nested for loop to fill it.

*The variable “num” can be replaced by 2 variables if the matrix isn’t square by r and c whom represents numbers of rows and number of columns at the initialisation and at the main in the nested loop filling the dp wit -1s.

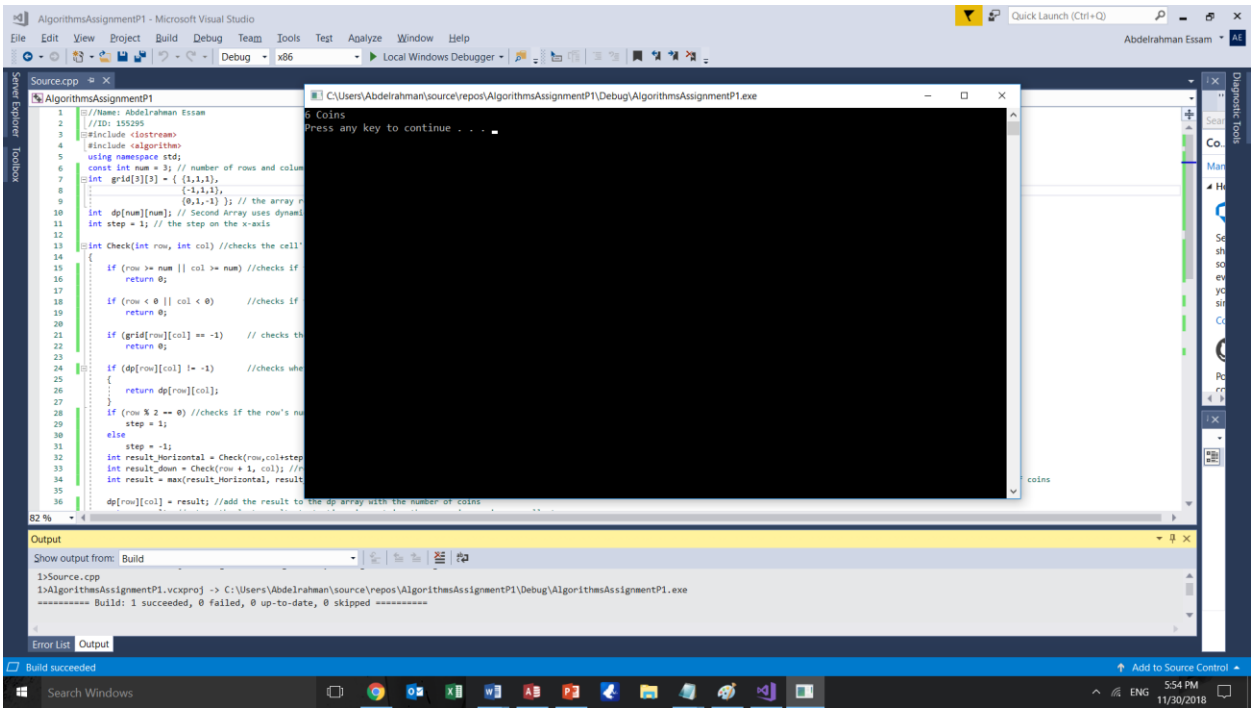
```
const int num = 3; // number of rows and columns
int  grid[3][3] = { {1,1,1},
                    {-1,1,1},
                    {0,1,-1} }; // the array repr
int  dp[num][num]; // Second Array uses dynamic p
```

```
for (int i = 0; i < num; i++) { //fillin
    for (int j = 0; j < num; j++) {
        dp[i][j] = -1;
    }
}
```

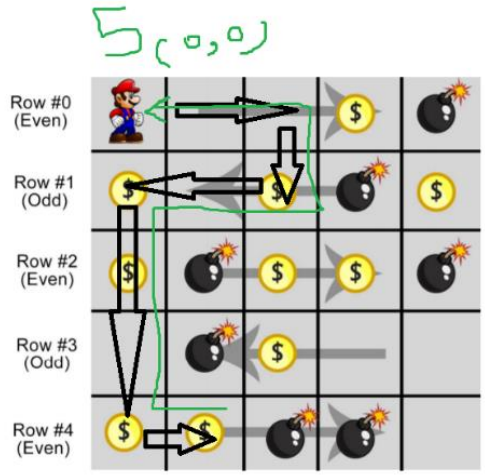
4.2 Screenshot of program actual run



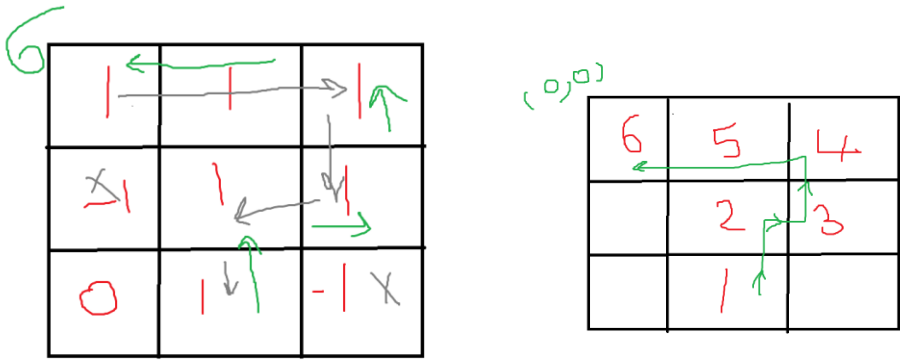
Another Test case for smaller Grid.



4. Code Explanation



As Shown in the figure above the program will start with the cell (0,0) from this cell the program will check recursively for the available way (if there is a bomb or not) the will compare between the available ways according to coins then save it's progress in a different 2D array (dp) which men that the cell will contain the max number of coins so far how ever a condition is set to determine whether it's an even or odd row to choose the correct step (left or right) recursively the program will add the pre calculated number back to the starting cell in which will contain the max number of coins that Mario can collect.



The figure on the left represent the matrix that Mario will traverse including the bombs(-1) coins (1) and empty cells(0).

The figure on the right is the dp matrix including all the values of the chosen path incrementing from 1 to 6 which is the start point (0,0)