Course Code: CSE 205
Course Title: Algorithms Theory

KSA – 1 : Assignment

Assignment No. : 01

Student Name : Jahidul Islam

Student ID : 221002504

Section : 221-D7

Date of Assignment : 6/12/2023

Date of Submission : 24/12/2023

Submitted to : **Dr. Faiz Al Faisal,** Assistant Professor & Associate Chairperson, CSE, Green University of Bangladesh

---

To be Filled by the Teacher.

Received Date :

Late Submission :

Obtained Mark :

Remarks :

Signature:

Department of Computer Science Engineering
Green University of Bangladesh

1. TITLE:

Assignment
CSE 205: Algorithms
Submission Date- 24/12/2023 (5:00PM) (**Strict**)
(Coding ethics will be checked (**Careful**))

1. In this assignment, your first task will be the grid addition.
2. Starting with any number on the top row, make your way to the bottom, adding the numbers as you go.
3. Each time you go to the next row, you can move straight down, or one place to the left or right.
4. What is the highest total you can make? What is the lowest?
5. It must be a complete report with the proper explanation of your program and output. Hence, writing the program only will not be enough. (It's the part of your understandability).
6. If I find the similarity with any others and copied from Internet, your report will be rejected and scored as 0. (A viva will be taken if necessary)
7. This assignment carries 5 points from your total grade.
8. Share the program and the report in my Google class. Report must be in printed version (must show the output of your code). And send me only the .c extension file. [If I can't find the C file, you will be scored to zero]

1	5	1	5	1	5
3	3	2	3	3	4
2	3	4	4	3	2
2	2	3	2	2	4
2	2	4	3	4	2
4	4	4	4	2	3

9. Sample Input / output-

```
Please enter the desire grid dimension: 6 6
Please enter the desired input:
```

1	5	1	5	1	5
3	3	2	3	3	4
2	3	4	4	3	2
2	2	3	2	2	4
2	2	4	3	4	2
4	4	4	4	2	3

```
Answer:
        Highest: 24
        Lowest: 12
Thank you!!!
```

2. PROBLEM STATEMENT:

The grid addition problem involves finding the highest and lowest totals while traversing a grid from the top row to the bottom row. The goal is to determine the maximum and minimum sums achievable by moving down to adjacent cells in the subsequent rows. Each move can be either straight down or one place to the left or right.

3. OBJECTIVES/AIM:

Objective 1: Develop a dynamic programming solution for the Grid problem.

Objective 2: Provide flexibility for users to input different coin denominations. By changing the array of coins.

Objective 4: Output the highest and lowest totals

4. PROCEDURE:

The grid addition problem is solved using a dynamic programming approach in Java.

4.1. Initialization:

```
grid = {
        {1, 5, 1, 5, 1, 5},
        {3, 3, 2, 3, 3, 4},
        {2, 3, 4, 4, 3, 2},
        {2, 2, 3, 2, 2, 4},
        {2, 2, 4, 3, 4, 2},
        {4, 4, 4, 4, 2, 3}
};
```

4.2. Dynamic Programming - Fill the Matrices:

- Implement the dynamic programming approach to find the highest and lowest totals:
  - Utilize two separate functions, **findMaxTotal** and **findMinTotal**, to calculate the maximum and minimum totals, respectively.
  - For each function:
    - Create a copy of the original grid to prevent modification of the input.
    - Start from the second-to-last row and iterate upwards, updating each element based on the maximum or minimum sum of the available options (straight down, left, or right).
    - Calculate the final totals considering the possibilities at each step.

4.3. User-Friendly Input:

- For simplicity, the program currently uses predefined Grid. Users can customize this array as needed.

4.4. Output:

- Finally, the program Print the highest and lowest totals: to the console.

```java
System.out.println("The highest total is: " + maxTotal);
System.out.println("The lowest total is: " + minTotal);
```

5. IMPLEMENTATION:

CODE IN JAVA:

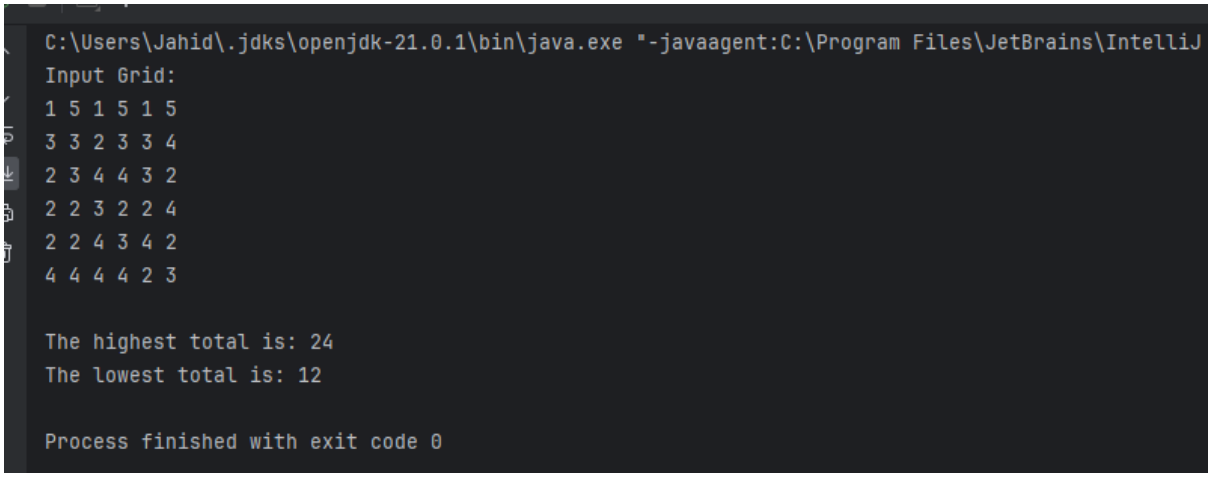
```java
public class GridAddition {

    public static void main(String[] args) {
        int[][] grid = {
                {1, 5, 1, 5, 1, 5},
                {3, 3, 2, 3, 3, 4},
                {2, 3, 4, 4, 3, 2},
                {2, 2, 3, 2, 2, 4},
                {2, 2, 4, 3, 4, 2},
                {4, 4, 4, 4, 2, 3}
        };

        int maxTotal = findMaxTotal(grid);
        int minTotal = findMinTotal(grid);

        System.out.println("The highest total is: " + maxTotal);
        System.out.println("The lowest total is: " + minTotal);
    }
```

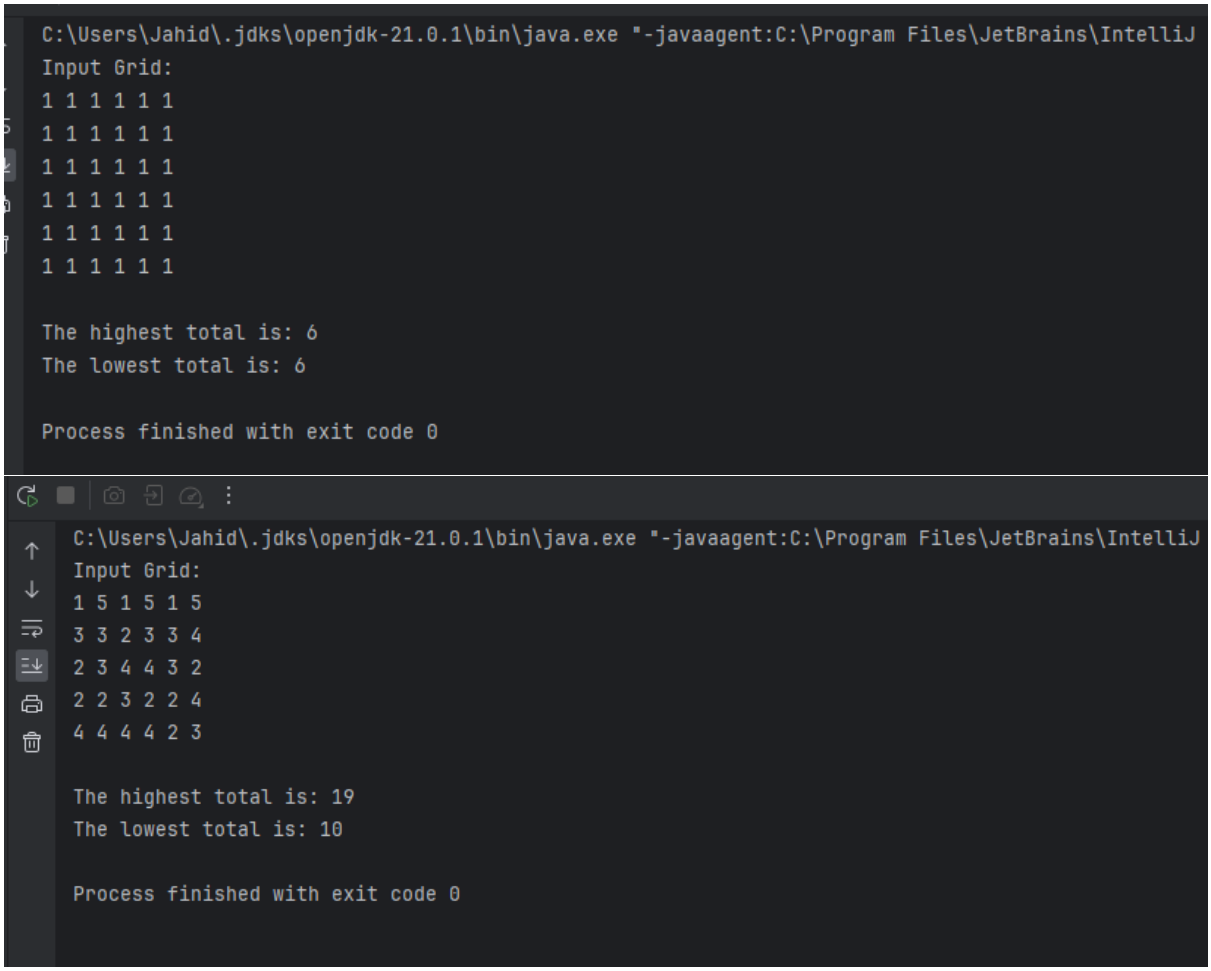
```java
public static int findMaxTotal(int[][] grid) {
    int rows = grid.length;
    int cols = grid[0].length;

    // Copy the original grid to avoid modifying the input grid
    int[][] dp = new int[rows][cols];
    for (int i = 0; i < rows; i++) {
        System.arraycopy(grid[i], 0, dp[i], 0, cols);
    }

    // Start from the second-to-last row and update each element with the maximum sum
    for (int i = rows - 2; i >= 0; i--) {
        for (int j = 0; j < cols; j++) {
            int left = dp[i + 1][Math.max(0, j - 1)];
            int down = dp[i + 1][j];
            int right = dp[i + 1][Math.min(cols - 1, j + 1)];

            dp[i][j] += Math.max(left, Math.max(down, right));
        }
    }

    // Find the maximum total in the top row
    int maxTotal = Integer.MIN_VALUE;
    for (int j = 0; j < cols; j++) {
        maxTotal = Math.max(maxTotal, dp[0][j]);
    }

    return maxTotal;
}
```

```java
    public static int findMinTotal(int[][] grid) {
        int rows = grid.length;
        int cols = grid[0].length;

        // Copy the original grid to avoid modifying the input grid
        int[][] dp = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            System.arraycopy(grid[i], 0, dp[i], 0, cols);
        }

        // Start from the second-to-last row and update each element with the minimum sum
        for (int i = rows - 2; i >= 0; i--) {
            for (int j = 0; j < cols; j++) {
                int left = dp[i + 1][Math.max(0, j - 1)];
                int down = dp[i + 1][j];
                int right = dp[i + 1][Math.min(cols - 1, j + 1)];

                dp[i][j] += Math.min(left, Math.min(down, right));
            }
        }

        // Find the minimum total in the top row
        int minTotal = Integer.MAX_VALUE;
        for (int j = 0; j < cols; j++) {
            minTotal = Math.min(minTotal, dp[0][j]);
        }

        return minTotal;
    }
}
```

6. TEST RESULT / OUTPUT:

- Objective 1:





7. DISCUSSION AND ANALYSIS:

6.1. Dynamic Programming Solution:

The implementation employs a dynamic programming approach to efficiently calculate the Max and Min value. This ensures optimal time complexity by avoiding redundant calculations and storing intermediate results in the dp array.

6.2. User Input Flexibility:

The program offers flexibility by allowing users to input the target amount and customize the coin denominations. This makes the solution adaptable to various scenarios where different denominations may be required.

6.3. Efficiency:

The algorithm's efficiency is notable, particularly for larger target amounts, as the dynamic programming technique optimizes the computation by breaking down the problem into smaller subproblems and reusing solutions.

6.4. Scalability:

The solution is scalable, and additional problems like solving Maze, finding path in maze can be easily incorporated.

8. CONCLUSION:

- The procedure involves initializing the input, applying a dynamic programming approach through separate functions, displaying the results, and providing insights into the time complexity.
- The program successfully addresses the grid addition problem, yielding the highest and lowest totals attainable through the defined rules.

This procedure outlines the steps taken to execute the program, emphasizing clarity in input handling, dynamic programming, and results presentation.