COP3503 – Spring 2025
Programming Assignment 4

Treasure Hunt – Finding the Safest Path

Assignment Description:

Welcome, adventurer! You are an explorer searching for a hidden treasure deep within the mystical Cursed Gridlands**.** The land is divided into a grid of dangerous zones, each with a risk level (a positive number). Your goal is to navigate from the top-left corner (0,0) to the bottom-right corner (n-1, m-1) while minimizing the total risk you accumulate along the way.

However, moving through the grid comes with rules:

- You can only move right or down at any step.
- Each cell contains a risk value, and stepping on it adds to your total risk.
- Your mission is to find the safest path to reach the treasure with the lowest possible total risk.

Your task will involve writing three different approaches to solving the problem:

1. Recursive Backtracking (Brute Force)
2. Memoization (Top-Down Dynamic Programming)
3. Tabulation (Bottom-Up Dynamic Programming)

Example Walkthrough:

Imagine you have the following Risk Map Grid (3x3):

```
3  2  5

1  9  8

4  6  2
```

Possible Paths & Their Risks:

1. Path 1: (3 → 2 → 5 → 8 → 2) → Total risk = 20
2. Path 2: (3 → 2 → 9 → 8 → 2) → Total risk = 24
3. Path 3: (3 → 2 → 9 → 6 → 2) → Total risk = 22
4. Path 4: (3 → 1 → 9 → 8 → 2) → Total risk = 23
5. Optimal Path: (3 → 1 → 4 → 6 → 2) → Total risk = 16

Your goal is to implement **three** methods that compute the optimal risk level for the treasure hunt.

Assignment Requirements

You must create a **TreasureHunt** class that implements three different solutions to compute the lowest-risk path.

1. Recursive Approach (`findMinRiskRecursive`)
    - Solve the problem using a brute force recursive approach (without optimization).
    - The function should compute all possible paths and return the one with the minimum risk.
    - Function prototype: `public int findMinRiskRecursive(int[][] grid, int row, int col)`
    - Time Complexity: O(2^(n+m)) where n and m are the input matrix dimensions.
2. Memoization Approach (`findMinRiskMemoization`)
    - Extend the recursive approach but use a 2D array cache to store previously computed results.
    - Function prototype: `public int findMinRiskMemoization(int[][] grid, int row, int col, int[][] memo)`
    - Time Complexity: O(n × m) where n and m are the input matrix dimensions.
3. Tabulation Approach (`findMinRiskTabulation`)
    - Solve the problem bottom-up using a DP table to store solutions for subproblems.
    - Start from `(0,0)` and iteratively compute the best risk values for each cell.
    - Function prototype: `public int findMinRiskTabulation(int[][] grid)`
    - Time Complexity: O(n × m) where n and m are the input matrix dimensions.
4. Make all methods public and class attribute private.
5. You may create additional helper methods and attributes if needed as long as they are implemented and called in your solution file and NOT called from the driver file. Adding extra methods to call in the driver file will not match to what the graders will use evaluate your code. This will result in a low score with no change to be applied!
6. If your code does not compile or is stuck in an infinite loop, you will receive no credit on any of the rubric categories except for code style/comments, assignment directions, and code submission/header.

Sample Input and Output:

Input:

```
3  2  5

1  9  8

4  6  2
```

Output:

```
Minimum Risk Path (Recursive): 16

Minimum Risk Path (Memoization): 16

Minimum Risk Path (Tabulation): 16
```

Using the Driver Script and Testing Your Code:

- You must not include any packages in your Java file.
- Your functions should match the required function signatures in `TreasureHunt.java` so that `TreasureHuntDriver.java` can call them.
- The driver script (`TreasureHuntDriver.java`) contains five predefined test cases and will execute your functions to validate their correctness.
- You should create your own test cases to ensure your implementation is correct, as the grader will be testing your code on different test cases.

Using the Python Script for Automated Testing

To ensure your program produces the correct output, use the provided Python script (`test_treasure_hunt.py`). This script will automatically compile and execute your Java program and compare its output to the expected output stored in `output.txt`.

Steps to Run the Python Script:

1. Place `test_treasure_hunt.py` in the same directory as your Java files.
2. Ensure the expected output file (`output.txt`) is in the same directory.
3. Run the script using:
    - `python3 test_treasure_hunt.py`
4. The script will:
    - Compile your Java files
    - Run your Java program (`TreasureHuntDriver`)
    - Compare the program output with `output.txt`
    - Display whether your output matches the expected output

Submission Instructions

1. Submit `TreasureHunt.java` to Webcourses/Canvas (No need to submit the driver file).
2. Ensure all methods match the function prototypes provided.
3. Code should compile successfully in Eustis (Failure to compile will result in a 0).
4. Follow proper coding style and documentation.