FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

DEPARTMENT TECHNOLOGY AND COMMUNICATION NETWORK

SEMESTER 2 2023/2024

CSC4202-5: DESIGN AND ANALYSIS OF ALGORITHMS

LECTURER NAME:

DR. NUR ARZILAWATI BINTI MD YUNUS

GROUP PROJECT

STUDENT NAME	MATRIC NUMBER
HASRINAH BINTI KURONG	213110
ANIS NADIRA BINTI NOOR MAISAM	210423
MOONNA BINTI ZULKAFLY	212137

Link to your online portfolio

https://sites.google.com/student.upm.edu.my/portfolio-project-csc4202/home

a) Original Scenario: Landslide Risk Reduction in a Hilly Terrain

Geographical Setting

The scenario is set in a region characterized by steep slopes, hilly terrain, and unstable geological formations, making it prone to landslides. The region is home to several communities, with infrastructure such as roads, homes, and agricultural lands scattered throughout the area.

Type of Disaster

The region faces the constant threat of landslides, which can be triggered by heavy rainfall, seismic activity, or human activities such as deforestation and construction. Landslides in this area can result in the sudden movement of rock, soil, and debris down slopes, causing extensive damage to infrastructure, homes, and natural habitats.

Damage Impact

Landslides have the potential to cause significant damage, including the loss of life, displacement of communities, and disruption of transportation networks. They can also lead to the destruction of homes, agricultural lands, and natural ecosystems, further exacerbating the impact on local communities and the environment.

Importance of Automated Algorithm Design (AAD)

AAD plays a crucial role in landslide risk reduction by optimizing landslide monitoring, early warning systems, land use planning and emergency response measures, It also plays a key role in the automated identification of evacuation routes in the event of a landslide. AAD can analyze geological data, rainfall patterns, terrain mapping, and land use changes to identify areas at risk and develop effective strategies for landslide mitigation, preparedness and evacuation.

Goal

The goal of the scenario is to develop a comprehensive landslide risk reduction plan that minimizes the occurrence and impact of landslides, protects vulnerable populations, and ensures the sustainable development of the region. The plan should aim to enhance the resilience of the region to landslides and promote safe and sustainable development practices in hilly terrains. One of the important components of the landslide risk reduction plan is the identification of the least risky evacuation paths for landslide victims, ensuring their safety during the emergency situations.

Expected Output

The expected output of the scenario includes a path that identifies the least risky evacuation route, which is a critical component of the landslide risk reduction plans. The plan should be designed to efficiently determine optimal evacuation routes, enhancing the region's resilience to landslides and promote safe evacuation practices.

b) Explain why finding an optimal solution for this scenario is important.

Finding the best way to reduce the risk of landslides in hilly areas is really important because for several reasons:

1. Save lives

   Landslides can be very dangerous, especially in areas with dense populations and highways. Finding the best solution is crucial for predicting where landslides might occur and keeping everyone safe. Preventive measures such as early warning systems, slope stabilization, and land use planning can significantly reduce the risks associated with landslides. An optimal plan minimizes these risks by identifying vulnerable areas, implementing early warning systems, and ensuring communities are prepared to evacuate.

2. Minimizing Damage and Economic Loss

   Landslides can destroy infrastructure, homes, and agricultural land, causing significant economic hardship. An optimal plan prioritizes cost-effective mitigation strategies like slope stabilization and land-use regulations, reducing the potential for damage during landslides.

3. Ensuring Sustainable Development

   By finding the optimal solution, sustainable development practices can be promoted in hilly terrains. This includes identifying safe areas for construction, implementing land use regulations, and ensuring that infrastructure is resilient to landslides, and leading to sustainable development in the region.

4. Enhancing Resilience

   Developing an optimal solution enhances the region's resilience to landslides. This includes identifying and implementing measures to mitigate landslide risks, such as slope stabilization, early warning systems and effective evacuation routes, which can help communities recover quickly from landslides and reduce the impact of future events.

c) Review the suitability of sorting, DAC, DP, greedy and graph algorithms as a solution paradigm for the chosen problem by stating their strengths and weaknesses.

1. Sorting Algorithm

   Sorting offers the advantage of efficiently organizing and retrieving data, which is crucial for prioritizing areas for landslide mitigation based on risk levels. By arranging geographical data and historical landslide occurrences, authorities can quickly identify high-risk zones and allocate resources accordingly. However, sorting alone is insufficient for addressing the complexity of landslide risk reduction, as it fails to consider the spatial relationships and dynamic factors that influence landslide behavior. Thus, while useful for initial data organization, sorting must be complemented by more sophisticated methods for a comprehensive solution. To conclude, sorting is beneficial for data organization but lacks the depth to address complex risk factors.

2. Divide and Conquer (DAC) Algorithm

   The divide and conquer approach excels in breaking down complex problems into smaller, more manageable subproblems. This makes it suitable for dividing the hilly terrain into distinct regions for focused landslide risk assessment and mitigation planning. Additionally, DAC's suitability for parallel processing allows for concurrent

risk assessments across multiple areas, enhancing efficiency. However, landslide risks are often interconnected, and actions in one area can affect neighboring regions. DAC may struggle to effectively manage these interdependencies, necessitating integration with other methods to fully address the interconnected nature of landslide risks. To conclude, divide and conquer aids in regional assessment but struggles with interdependencies.

3. Dynamic Programming (DP) Algorithm

   Dynamic programming is highly effective for optimizing resource allocation in landslide prevention, taking into account factors like cost-effectiveness and impact. By efficiently handling overlapping subproblems through memoization, DP can identify repeated risk patterns and develop robust mitigation strategies. Nevertheless, DP can be computationally intensive, posing scalability challenges for large regions or complex terrains. Moreover, it requires a deep understanding of the problem to effectively decompose it into subproblems. While powerful for resource optimization, DP may need additional support to manage large-scale implementations. Dynamic programming excels in resource optimization but faces scalability issues.

4. Greedy Algorithm

   Greedy algorithm is straightforward to implement and provide quick, near-optimal solutions, making it useful for immediate prioritization of landslide mitigation efforts. It can rapidly identify high-risk areas and initiate quick responses, which is crucial in emergency situations. However, greedy algorithm often make decisions based on local optima, which may not lead to the best global outcome. This limitation can result in suboptimal solutions when considering the broader context of landslide risk reduction. Therefore, while valuable for rapid initial actions, greedy algorithms should be used in conjunction with other strategies for long-term planning. Greedy algorithms offer quick solutions but may not be globally optimal.

5. Graph Algorithm

   Graph algorithm is essential for modeling transportation networks and evacuation routes, which are critical in landslide emergency response. It provides efficient solutions for shortest path problems, ensuring effective evacuation planning and coordination with emergency services. However, implementing a graph algorithm

requires detailed data and careful consideration of the graph's structure and properties. Additionally, the complexity of dynamic and spatially variable risks in landslide-prone regions can pose significant challenges.

d) Design the algorithm to solve the problem and explain the idea of your algorithm paradigm by emphasising which part needs recurrence and the function for the optimization.

Dynamic Programming Overview:

Dynamic Programming is an optimization technique that solves problems by breaking them down into simpler subproblems and storing the results of these subproblems to avoid redundant calculations. It is particularly useful for problems with overlapping subproblems and optimal substructure.

Recurrence Relation and Optimization Function:

Recurrence Relation: Defines how the solution to a larger problem is built from the solutions to smaller subproblems.

Optimization Function: Ensures that the best (optimal) solution is selected by comparing the outcomes of different choices.

Algorithm Design:

1. Terrain Data Collection:
    - Collect the terrain data, including elevation, slope, and geological information.
    - Preprocess the data to create a grid representation of the terrain.
2. Landslide Hazard Mapping:
    - Identify areas at risk of landslides based on the terrain data and historical landslide events.
    - Create a landslide hazard map that highlights high-risk areas.
3. Slope Stabilization Measures:
    - Implement measures such as retaining walls, slope reinforcement, and reforestation to stabilize slopes and reduce the risk of landslides.
    - Optimize the placement of these measures based on the landslide hazard map.

4. Land Use Zoning Regulations:
    - Implement regulations that restrict construction in high-risk areas and promote sustainable land use practices.
    - Ensure that land use planning is integrated with the landslide hazard map.
5. Community Awareness and Preparedness Programs:
    - Educate communities about the risks of landslides and provide training on emergency response measures.
    - Develop evacuation procedures and coordinate with emergency services.
6. Emergency Response Protocols:
    - Establish protocols for responding to landslide events, including evacuation procedures and coordination with emergency services.

The Algorithm Paradigm

Problem Breakdown:

- The problem is broken down into smaller subproblems, where each subproblem is to find the minimum risk path to a specific cell (i, j) in the grid.

Recurrence Relation:

- The recurrence in the algorithm will be in the dynamic programming step, where the minimum risk path is calculated for each cell in the terrain grid. The recurrence relation will be:

$$dp[i][j] = \min(dp[i-1][j] + terrain[i][j], dp[i][j-1] + terrain[i][j])$$

- This recurrence relation calculates the minimum risk path for each cell by considering the minimum risk paths from the top and left cells.

Optimization Function:

- The optimization function minimises the risk by considering the minimum risk from the possible previous cells.

- The function for optimization will be the minimum risk path, which is the path that minimizes the risk of landslides. This function will be calculated using a dynamic programming approach.

Pseudocode:

1. Start
2. Initialize rows and cols from terrain dimensions
3. Create a 2D array dp for dynamic programming
4. Initialize dp with maximum values
5. Set dp[0][0] as the starting point of the terrain grid
6. Iterate through each cell of the terrain grid
    - Update dp[i][j] with the minimum risk to reach the current cell
7. Return dp[rows-1][cols-1] as the minimum risk path
8. End

e) Define the algorithm specification.

The dynamic programming algorithm specification involves several key components that ensure efficient and optimal problem-solving. The first step is to break down a complex problem into smaller, manageable subproblems. This decomposition is crucial because it allows the algorithm to focus on solving each subproblem independently, which is essential for efficient computation.

Problem Statement: Given a hilly terrain characterized by steep slopes and unstable geological formations, develop an algorithm to find the least risky evacuation path for landslide victims. The goal is to minimize the risk of casualties and damage to infrastructure during a landslide event.

Input:

- A 2D array terrain representing the terrain, where terrain[i][j] represents the risk value at cell (i, j) of the terrain.
- The terrain is a grid of size n x m, where n is the number of rows and m is the number of columns.
- The risk value at each cell represents the likelihood of a landslide occurring at that location, with higher values indicating higher risk.

Output:

- The minimum risk path from the top-left corner of the terrain to the bottom-right corner, representing the least risky evacuation path for landslide victims.
- The total risk value along the minimum risk path.

Constraints:

- The terrain grid is non-empty, with n > 0 and m > 0.
- Each cell in the terrain grid contains a non-negative integer representing the risk value, i.e., 0 <= terrain[i][j] <= 100.

Dynamic Programming Approach:

- **Initialization**: Create a 2D array dp of size n x m to store the minimum risk at each cell. Initialize all values in dp to Integer.MAX_VALUE except for dp[0][0], which is initialized to terrain[0][0].
- **Recurrence Relation**: For each cell (i, j) in the terrain grid, update dp[i][j] with the minimum risk value to reach that cell from the top-left corner. The minimum risk value to reach cell (i, j) is the minimum of the risks from the cell above (i-1, j) and the cell to the left (i, j-1), plus the risk value at cell (i, j). This can be expressed as: dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + terrain[i][j]
- **Optimal Substructure**: The minimum risk path to reach cell (i, j) depends on the minimum risk paths to reach the cells (i-1, j) and (i, j-1).

- **Final Result**: The minimum risk path to reach the bottom-right corner (n-1, m-1) is stored in dp[n-1][m-1]. Return this value as the total risk along the minimum risk path.

Time Complexity: The time complexity of the dynamic programming approach is O(nm), where n is the number of rows and m is the number of columns in the terrain grid. This is because we iterate through each cell of the terrain grid once to calculate the minimum risk path.

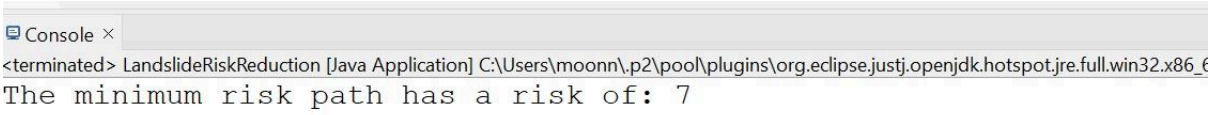
f) Develop a program Java language.

```java
package algo;
import java.util.Arrays;

public class LandslideRiskReduction {

    // Function to find the minimum risk path
    public static int findMinRisk(int[][] terrain) {
        int rows = terrain.length;
        int cols = terrain[0].length;

        // Create a dp array to store the minimum risk at each cell
        int[][] dp = new int[rows][cols];

        // Initialize the dp array with maximum possible values
        for (int[] row : dp) {
            Arrays.fill(row, Integer.MAX_VALUE);
        }

        // Start from the top-left corner
        dp[0][0] = terrain[0][0];

        // Fill the dp array with the minimum risk values
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (i > 0) {
                    dp[i][j] = Math.min(dp[i][j], dp[i-1][j] + terrain[i][j]);
                }
                if (j > 0) {
                    dp[i][j] = Math.min(dp[i][j], dp[i][j-1] + terrain[i][j]);
                }
            }
        }

        // Return the minimum risk to reach the bottom-right corner
        return dp[rows-1][cols-1];
    }

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

        // Find and print the minimum risk
        int minRisk = findMinRisk(terrain);
        System.out.println("The minimum risk path has a risk of: " + minRisk);
    }
}
```

```
import java.util.Arrays;

public class LandslideRiskReduction {

    // Function to find the minimum risk path
    public static int findMinRisk(int[][] terrain) {
        int rows = terrain.length;
        int cols = terrain[0].length;

        // Create a dp array to store the minimum risk at each cell
        int[][] dp = new int[rows][cols];

        // Initialize the dp array with maximum possible values
        for (int[] row : dp) {
            Arrays.fill(row, Integer.MAX_VALUE);
        }

        // Start from the top-left corner
        dp[0][0] = terrain[0][0];

        // Fill the dp array with the minimum risk values
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (i > 0) {
                    dp[i][j] = Math.min(dp[i][j], dp[i-1][j] + terrain[i][j]);
                }
                if (j > 0) {
                    dp[i][j] = Math.min(dp[i][j], dp[i][j-1] + terrain[i][j]);
                }
            }
        }

        // Return the minimum risk to reach the bottom-right corner
        return dp[rows-1][cols-1];
```

```
    }

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

        // Find and print the minimum risk
        int minRisk = findMinRisk(terrain);
        System.out.println("The minimum risk path has a risk of: " + minRisk);
    }
}
```

The Output :



g) Provide an analysis of the **algorithm's correctness** as well as **time complexity** (best, average and worst time) by using asymptotic notation.

Algorithm's Correctness Analysis

The algorithm implements a dynamic programming approach to find the minimum risk path in a 2D grid.

1. Initialization

   The `dp` array is initialized to store the minimum risk at each cell. The initial cell `dp[0][0]` is set to the risk value of the starting cell `terrain[0][0]`, ensuring the starting point is correctly represented.

2. Filling the DP Array

The algorithm iterates over each cell in the grid and updates the `dp` array with the minimum risk value by considering the minimum risk from the top cell (`i-1, j`) and the left cell (`i, j-1`). This ensures that each cell in the `dp` array holds the minimum risk required to reach that cell.

3. Final Value
   The value in `dp[rows-1][cols-1]` represents the minimum risk to reach the bottom-right corner of the grid from the top-left corner, which is the desired output.

The algorithm correctly computes the minimum risk path by ensuring that at each step, the minimum cumulative risk is propagated through the grid. The correctness is maintained by the principle of dynamic programming, where the optimal solution to the problem is built from the optimal solutions to its subproblems. The algorithm ensures that the minimum risk path is found, which is crucial for safe evacuation during a landslide. This directly aligns with the goal of minimizing the impact of landslides by finding the safest routes for evacuation. By calculating the cumulative risk at each cell, the algorithm effectively maps out the risk landscape, helping to identify the safest paths in a landslide-prone area.

Time Complexity Analysis

The time complexity of the algorithm can be analyzed as follows:

1. Initialization
   Initializing the `dp` array takes `O(rows * cols)` time since we need to fill the entire array with `Integer.MAX_VALUE`.

2. Filling the DP Array
   The nested loops iterate over each cell in the grid. Each cell is visited once, and updating the `dp` array takes constant time `O(1)` per cell. Therefore, the time complexity for filling the `dp` array is `O(rows * cols)`.

Since both the initialization and the main computation steps have the same time complexity, the overall time complexity of the algorithm is **O(rows * cols)**. With a time complexity of O(rows * cols), the algorithm efficiently handles large terrains, ensuring that

evacuation paths can be computed quickly even in real-time scenarios. This is critical in emergency situations where rapid decision-making is necessary.

Best Case, Average Case, and Worst Case Analysis

As each cell in the grid represents a terrain segment with an associated risk value, for this algorithm, the time complexity remains the same across all cases because the algorithm processes each cell in the grid exactly once regardless of the input values.

- **Best Case**

  The best case occurs when the terrain values are such that the minimum risk path is straightforward, either going directly right or down, but the algorithm still needs to compute the risk values for all cells. Time complexity is **O(rows * cols)**. Example terrain grid:

```
int[ ][ ] terrain = {
    {1, 1, 1},
    {1, 1, 1},
    {1, 1, 1}
};
```

- **Average Case**

  For an average terrain configuration, the algorithm processes each cell once. If the terrain grid has varying risk values, neither extremely high nor extremely low, the algorithm needs to explore various paths to find the one with the least cumulative risk by processing each cell exactly once. Time complexity is **O(rows * cols)**. Example terrain grid:

```
int[ ][ ] terrain = {
    {1, 3, 1},
    {2, 5, 1},
    {4, 2, 1}
};
```

- **Worst Case**

Even in the worst-case scenario, where the risk values are high and require maximum computation, the algorithm still processes each cell once to find the minimum cumulative risk. Time complexity is **O(rows * cols)**. Example of terrain grid:

```
int[ ][ ] terrain = {
    {5, 8, 9},
    {7, 10, 6},
    {12, 11, 4}
};
```

Space Complexity Analysis

The space complexity of the algorithm is dominated by the space required for the `dp` array, which is **O(rows * cols)**. This space is used to store the minimum risk values for each cell in the grid.

The `dp` array is a two-dimensional array that stores the minimum risk values for each cell in the grid. It has dimensions `rows` x `cols`, where `rows` is the number of rows in the grid and `cols` is the number of columns. Each cell in the `dp` array corresponds to a cell in the terrain grid, and it stores the minimum cumulative risk value to reach that cell from the starting point `(0, 0)`. Since the `dp` array stores a value for each cell in the grid, its total size is proportional to the number of cells in the grid, which is `rows` x `cols`.

In the context of finding the evacuation path with the least risk in a terrain grid prone to landslides, the space complexity analysis is crucial for understanding the algorithm's memory requirements. The `dp` array's space complexity directly relates to the grid size and determines the amount of memory needed to store the minimum risk values for each cell. As the algorithm processes each cell in the grid and computes the minimum risk path, the `dp` array's space usage grows with the grid size. This space is essential for the algorithm to store intermediate results and ensure that the optimal evacuation path can be traced back from the destination cell to the starting point.

h) Develop an online portfolio (using google sites or github or google colab or any suitable tool)

PSEUDOCODE :

Algorithm MinimizeLandslideRisk

Input: 2D array terrain representing the terrain grid with risk levels

Output: Minimum risk to reach the bottom-right corner

1. Define the Terrain Grid:
   Let terrain be a 2D array of size rows x cols

2. Initialise the DP Table:
   Create a 2D array dp of the same size as terrain
   For i from 0 to rows-1
      For j from 0 to cols-1
         dp[i][j] = Infinity // Initialize all cells with a large value

3. Base Case:
   dp[0][0] = terrain[0][0]

4. Recurrence Relation:
   For i from 0 to rows-1
      For j from 0 to cols-1
         If i > 0
            dp[i][j] = min(dp[i][j], dp[i-1][j] + terrain[i][j])
         If j > 0
            dp[i][j] = min(dp[i][j], dp[i][j-1] + terrain[i][j])

5. Boundary Conditions:
   For i = 0 and j = 0
      Continue // Already handled in base case

6. Final Solution:
   Return dp[rows-1][cols-1]

End Algorithm