Al Imam Mohammad ibn Saud Islamic University
College of Computer and Information Sciences

Computer Science Department

Course Title:	Design and Analysis of Algorithms
Course Code:	CS216
Instructor:	Dr. Dhouha Ben Noureddine
project:	Path minimization
Due date:	Sunday (2024-10-20)
Semester:	1st Semester, 2024
Marks:	15 points

Student ID	Student Name	Section
444008062	Remaz Nawaf Alotaibi	371
442013809	Shaden Fahad Almudbal	
443020299	Jawza Fahad Alanazi	
444006296	Sara Abduallah Moubrad	

Table of Contents

Graph Class:

The pseudocodes for the Graph class:

```
Function Graph(V):
  Input: V (Number of vertices)
  Initialize adjList as an empty list of lists with size V
  For i from 0 to V-1:
    Append an empty list to adjList
Function addEdge(u, v, weight):
  Input: u (source vertex), v (destination vertex), weight (edge weight)
  Add [v, weight] to the adjacency list of vertex u
Function bellmanFord(source, destination):
  Input: source (starting vertex), destination (target vertex)
  Initialize dist as an array of size V, filled with infinity (MAX_VALUE)
  Set dist[source] to 0 (distance to source is 0)

  // Relax all edges V-1 times
  For i from 1 to V-1:
    For each vertex u from 0 to V-1:
      For each neighbor [v, weight] of u in adjList:
        If dist[u] is not infinity and dist[u] + weight < dist[v]:
          Set dist[v] to dist[u] + weight

  // Check for negative weight cycles
  For each vertex u from 0 to V-1:
    For each neighbor [v, weight] of u in adjList:
      If dist[u] is not infinity and dist[u] + weight < dist[v]:
        Return MIN_VALUE (negative cycle detected)

  Return dist[destination] (shortest distance from source to destination)
```

A brute-force algorithm

1) Show the designed algorithms using the three algorithms and write their pseudocodes

```
Function bruitForce():
  Start timer
  Call dfs(0, 0, matrix[0][0])
  End timer
  Print minimum path cost and execution time
Function dfs(x, y, currentSum):
  If (x, y) is the bottom-right corner of the matrix:
    Update minSum with the smaller of minSum and currentSum
    Return
  Mark (x, y) as visited
  For each direction (up, down, left, right):
    Calculate newX and newY
    If newX and newY are valid and not visited:
      Call dfs(newX, newY, currentSum + matrix[newX][newY])
    Unmark (x, y) as visited
Function isValid(x, y):
  Return true if (x, y) is within bounds and not visited
```

2) Analyze the designed algorithms and compute an asymptotic upper bound using Big-Oh notation. Conduct a theoretical analysis of the time complexity of all designed algorithms. Compare and explain their time complexity.

Brute Force (DFS-based)
The brute force algorithm uses Depth-First Search (DFS) to explore all possible routes from the top-left to the bottom-right corner of the grid.

Time Complexity:

- DFS examines every potential path in the grid.
- In an **(N X N)** grid, each cell can be explored in up to 4 directions (up, down, left, and right).
- Since DFS checks all paths, and the number of paths increases exponentially, the worst-case time complexity is **O(4^N^2).**
- For each of the **N^2** cells, DFS explores up to 4 branches, leading to an exponential increase in recursive calls.

3) Test the implemented algorithms and show results with different inputs.

```
4
295 934 535 340
931 176 475 635
-594 350 -932 826
664 237 357 869
1276
PS C:\Users\dell> ^C
PS C:\Users\dell>
PS C:\Users\dell>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\dell\AppData\Local\Temp\vscodesws_239bf\jdt_ws\jdt.ls-java-project\bin' 'project216_2024.Project216_2024'
5
32 101 66 -13 19
11 -14 48 158 7
101 114 175 12 34
89 125 330 21 141
100 33 112 41 26
290
PS C:\Users\dell>
```

A graph-based algorithm

1) Show the designed algorithms using the three algorithms and write their pseudocodes

```
Function GraphBasedAlgorithm():
  Start timer
  Call MinBellman()
  End timer
  Print execution time
==========================
Function MinBellman():
  Initialize rows and cols as N
  Initialize graph with V vertices (V = rows * cols)
  For i = 0 to rows-1:
    For j = 0 to cols-1:
      node = i * cols + j
      Add edges to adjacent nodes (up, down, left, right) in graph
      Add edge weights based on matrix values
  source = 0 (top-left corner)
  destination = rows * cols - 1 (bottom-right corner)
  Call graph.bellmanFord(source, destination)
  Print minimum path sum
```

2) Analyze the designed algorithms and compute an asymptotic upper bound using Big-Oh notation. Conduct a theoretical analysis of the time complexity of all designed algorithms. Compare and explain their time complexity.

Graph-Based Algorithm (Bellman-Ford)

This algorithm transforms the matrix into a graph and applies the Bellman-Ford algorithm to find the shortest path from the top-left to the bottom-right corner.

Time Complexity:

- **Graph Construction**: The matrix is represented as a graph with V = N X N vertices.
- Each vertex can have up to 4 edges (one for each possible direction), making the number of edges **O(4×N^2) =O(N^2)**
- **Bellman-Ford Algorithm**: The algorithm relaxes all edges **V – 1 time**
- Relaxing all edges takes **O(E)** where ***E*** is the number of edges.
- In this case, **E=O(N^2)** and **V=O(N^2)** because each node can have up to 4 edges.
- As a result, the total time complexity of Bellman-Ford is **O(V×E) =O(N^4)**

3) Test the implemented algorithms and show results with different inputs.

```
PS C:\Users\dell>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\dell\AppData\Local\Temp\vscodesws_239bf\jdt_ws\jdt.ls-java-project\bin' 'project216_2024.Project216_2024'
3
84 71 90
68 35 98
41 89 19
295
PS C:\Users\dell>
```

A dynamic-programming based algorithm.

1) Show the designed algorithms using the three algorithms and write their pseudocodes.

```
Function DyanamicAlgorithm():
  Start timer
  Call DynamicminPathSum(matrix)
  End timer
  Print minimum path sum and execution time
Function DynamicminPathSum(matrix):
  Initialize dp table with same dimensions as matrix
    dp[0][0] = matrix[0][0] (starting point)
    // Fill first row
  For j = 1 to cols-1:
    dp[0][j] = dp[0][j-1] + matrix[0][j]
  // Fill first column
  For i = 1 to rows-1:
    dp[i][0] = dp[i-1][0] + matrix[i][0]
   // Fill the rest of dp table
  For i = 1 to rows-1:
    For j = 1 to cols-1:
      dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + matrix[i][j]
   Return dp[rows-1][cols-1] (minimum path sum to the bottom-right corner)
```

2) Analyze the designed algorithms and compute an asymptotic upper bound using Big-Oh notation. Conduct a theoretical analysis of the time complexity of all designed algorithms. Compare and explain their time complexity.

Dynamic Programming (DP-based)

This algorithm uses Dynamic Programming (DP) to compute the minimum path sum by filling a table of size **N×N** times

Time Complexity:

- The algorithm fills a DP table by iterating through each cell in the **N×N** once.
- For each cell **(i,j)** it computes the minimum cost from the top or left cell in **O(1)** time
- Since there are **N^2** cells in the grid, the time complexity is **O(N^2)**

3) Test the implemented algorithms and show results with different inputs.