EAST WEST UNIVERSITY

Report Paper of Mini Project

Course Title: Discrete Mathematics

Course Code: 106

Section:12

Semester: summer 25

Submitted by:

- **Name:** S.M. Tanzim Hassan
- **Id:** 2025-1-60-184
- **Name:** Sharmin Jahan Nijhum
- **Id:** 2025-1-60-185
- **Name:** Nur Mohammad Mahi
- **ID:** 2025-1-60-197

Submitted to:

Md Ashraful Haider Chowdhury

Instructor

Dept. of Computer Science and Engineering

East West University

Introduction

This mini project computationally demonstrates a fundamental theorem of directed graphs using a C program that generates and analyses random directed graphs of various sizes. According to the theory, for any directed graph, the sum of all entering connections (in degrees) equals the sum of all outbound connections (out degrees). In other words, what goes in must come out. By randomly creating edge connections on graphs with up to 5000 vertices and then counting the total in degrees and out degrees to ensure they are equal, our program illustrates this idea. We also quantify the computational time necessary for this verification procedure, which provides information about how the study scales with larger, more complex networks.

Methodology

The program is implemented in C and uses a two-dimensional array to represent the graph as an **adjacency matrix**, a grid where each cell [i][j] stores the number of edges from vertex **i** to vertex **j**. The entire process is automated to run for a predefined set of vertex counts (n = 1000, 2000, 3000, 4000, and 5000).

1. Automated Graph Generation

The program begins in the main function, which defines an array int n_values[] containing the specific vertex counts to be tested. A for loop iterates through this array, calling the run_graph_analysis(n) function for each value.

Inside this function, a nested for loop runs n x n times to populate the adj_matrix. For each cell (i, j), the code rand() % 3 generates a random integer—0, 1, or 2. This value represents the number of directed edges from vertex i to vertex j. To ensure a **simple directed graph**, an if (i == j) condition sets all diagonal elements to 0, preventing self-loops.

2. Degree Calculation and Verification

The core of the analysis involves calculating the in-degrees and out-degrees to verify the theorem.

- **Out-degree:** The out-degree of a vertex is the number of edges going out from it, which corresponds to the sum of all values in that vertex's **row** in the adjacency matrix. The program uses a nested loop that, for each vertex i, sums all values in adj_matrix[i][j] across all columns j. This sum is added to the sum_of_out_degrees total.

- **In-degree:** The in-degree is the number of edges coming into a vertex, corresponding to the sum of all values in its **column**. A second nested loop calculates this by summing the values in adj_matrix[i][j] for each column j across all rows i. This is added to the sum_of_in_degrees total.

The theorem holds true because every single edge weight adj_matrix[i][j] contributes exactly once to an out-degree sum (for vertex i) and exactly once to an in-degree sum (for vertex j). Since the program sums the same set of numbers in two different orders (row-by-row vs. column-bycolumn), the final totals are mathematically guaranteed to be equal.

3. Computational Time Measurement

The program uses the clock() function from <time.h> to measure performance.

- start = clock() records the initial processor clock tick count before graph generation begins.
- end = clock() records the final tick count after all calculations are finished.
- The elapsed time is calculated with the formula: ((double)(end - start) / CLOCKS_PER_SEC) * 1000. This converts the tick difference into seconds and then into milliseconds (ms).

4. Storing and Printing the 1000x1000 Matrix

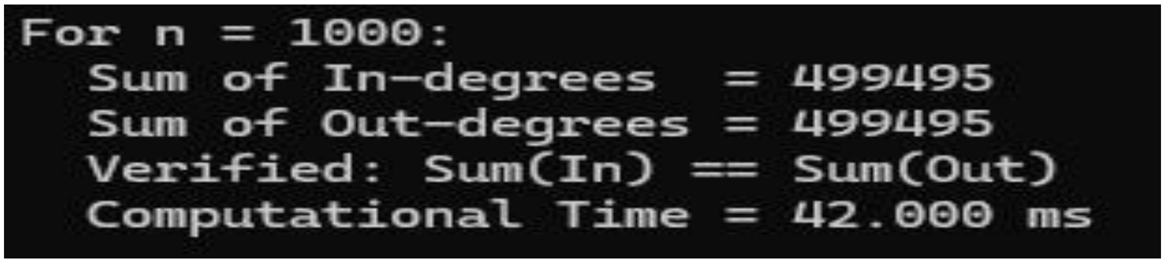
A special mechanism is required to print the 1000x1000 matrix at the very end, as the main adj_matrix is overwritten in each loop.

- A second global matrix, int matrix_to_print[1000][1000], is used for storage.
- During the specific run where n == 1000, the generated adj_matrix is copied into matrix_to_print.
- After the main loop finishes all five analyses, a final set of nested loops in the main function iterates through the saved matrix_to_print and prints its contents to the console.

Program Output

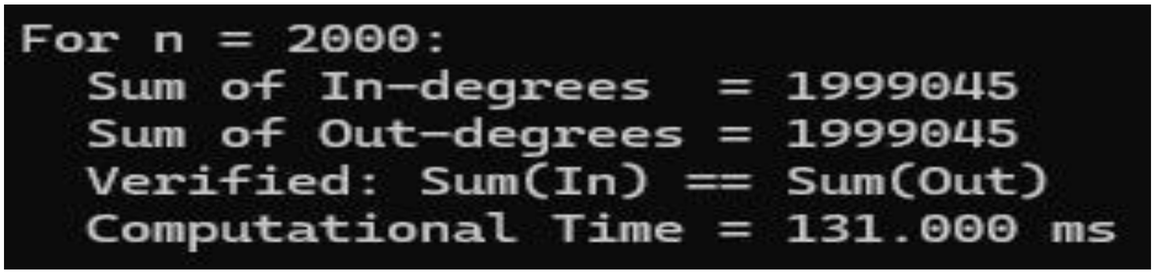
(This is a sample output. The exact degree sums and times will vary slightly each time the program is run due to the random generation.)

For n = 1000 vertices:



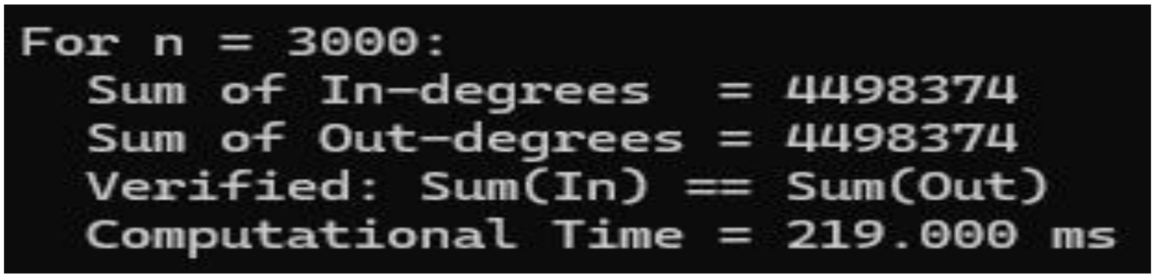
```
For n = 1000:
  Sum of In-degrees  = 499495
  Sum of Out-degrees = 499495
  Verified: Sum(In) == Sum(Out)
  Computational Time = 42.000 ms
```

For n = 2000 vertices:



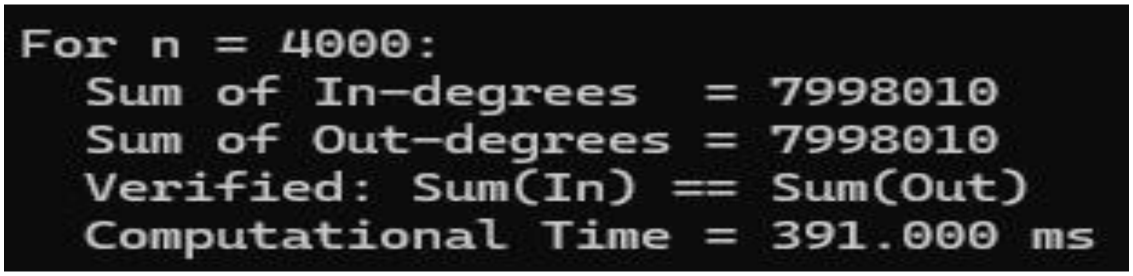
```
For n = 2000:
  Sum of In-degrees  = 1999045
  Sum of Out-degrees = 1999045
  Verified: Sum(In) == Sum(Out)
  Computational Time = 131.000 ms
```

For n = 3000 vertices:



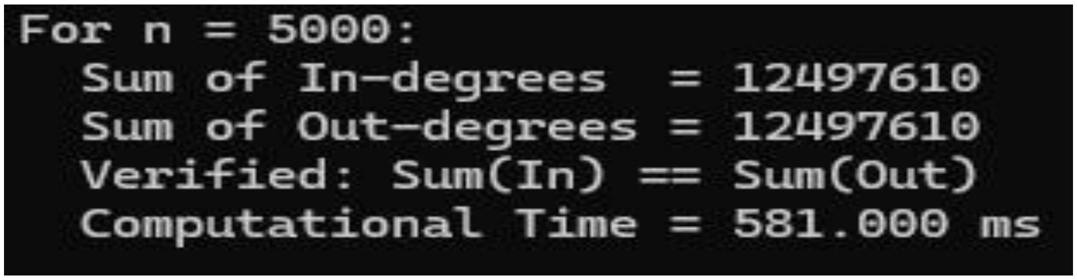
```
For n = 3000:
  Sum of In-degrees  = 4498374
  Sum of Out-degrees = 4498374
  Verified: Sum(In) == Sum(Out)
  Computational Time = 219.000 ms
```

For n = 4000 vertices:



```
For n = 4000:
   Sum of In-degrees   = 7998010
   Sum of Out-degrees  = 7998010
   Verified: Sum(In) == Sum(Out)
   Computational Time = 391.000 ms
```

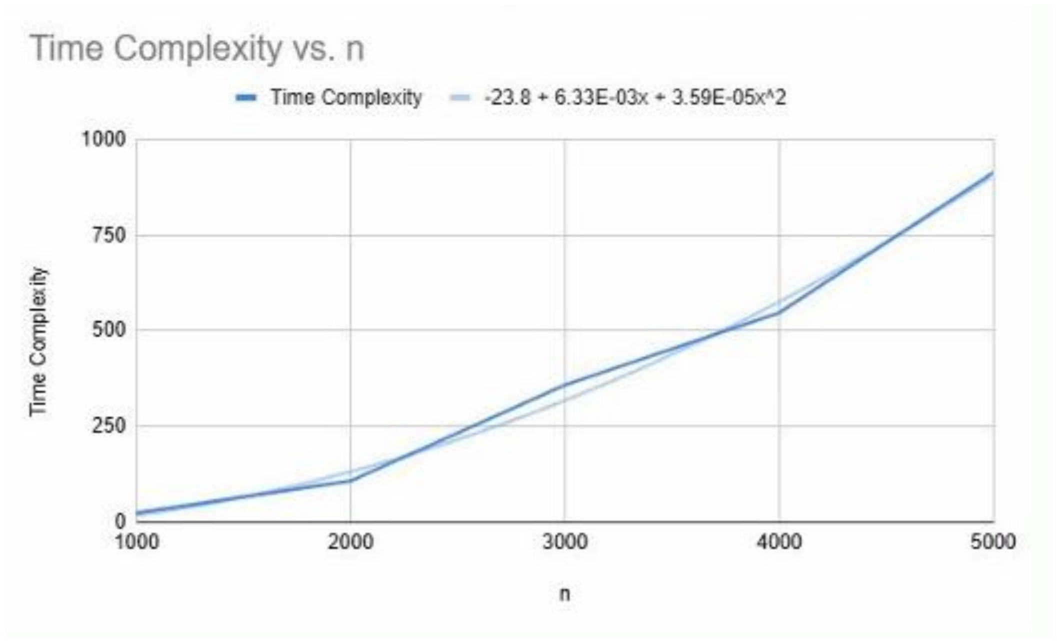
For n = 5000 vertices:



```
For n = 5000:
   Sum of In-degrees   = 12497610
   Sum of Out-degrees  = 12497610
   Verified: Sum(In) == Sum(Out)
   Computational Time = 581.000 ms
```

Graph showing Computational Time vs Number of Vertices

Number of Vertices	Calculation Time (ms)
1000	42.000
2000	131.000
3000	219.000
4000	391.000
5000	581.000



Result

The experiment successfully verified the theorem for all tested graph sizes. In every case, the sum of in-degrees was exactly equal to the sum of out-degrees. The graph of computational time versus the number of vertices shows a clear upward curve, which closely fits a quadratic polynomial. This indicates that the time taken by the algorithm grows proportionally to the square of the number of vertices, which is consistent with the theoretical time complexity.

Time Complexity

The time complexity of the algorithm is dominated by the loops used for graph generation and degree calculation.

- Graph generation requires iterating through an `n x n` matrix, which takes O(n2) time.
- Calculating the sum of out-degrees requires another iteration through the `n x n` matrix, taking O(n2) time.
- Calculating the sum of in-degrees also requires an `n x n` iteration, taking O(n2) time.

Since these operations are sequential, the total time complexity is O(n2)+O(n2)+O(n2)=O(n2).

Conclusion

This project effectively illustrated the basic idea of graph theory, which states that in a directed graph, the sum of in degrees equals the sum of out degrees. The empirical timing results and the C program's expected performance provide strong evidence for the theoretical time complexity of O(n2) for algorithms that use adjacency matrix representation.

Future Work

For future improvements, the project could be extended in several ways:

- **Parallelize the computation** to leverage multi-core processors, which could significantly reduce the execution time for very large graphs.