EAST WEST UNIVERSITY

Course title: DISCRETE MATHEMATICS

Course code: CSE106

Section: 10

Semester: Fall'25

Report Paper of Mini Project

Submitted by: Mehrab Morshed Marjan

ID: 2025-1-60-142

Roushna Akter Nowrin

ID: 2025-2-60-006

Salman Farchi

ID: 2025-1-60-052

Mitro Khan

ID:2025-1-60-434

Submitted to:

Instructor

Sadia Nur Amin

Dept. of Computer Science and Engineering

East West University

Introduction:

This project focuses on representing **directed graphs** using an adjacency matrix. A directed graph consists of a set of vertices connected by edges, where each edge has a specific direction from one vertex to another. This differs from an undirected graph, in which edges do not have any direction.

The graph is stored using an **adjacency matrix**. The adjacency matrix is a two dimensional table where rows correspond to outgoing edges from a vertex and columns correspond to incoming edges to a vertex. The element **`adj[i][j]`** represents the number of edges from vertex i to vertex j. This representation makes it easy to verify connections between vertices and supports both multiple edges and self-loops.

In this project, we:

1. Generate random directed graphs.
2. Count the total number of edges and compute the in-degrees and out-degrees of vertices.
3. Measure the computational time required.
4. Repeat the experiment for graphs of different sizes.
5. Present the results using Excel-based visualizations.
6. Compare the experimental results with theoretical time complexity.

The objective is to understand both the **practical performance** and the **theoretical complexity** of graph algorithms when adjacency matrices are used.

Methodology:

1. Graph Generation

1.1 Random Directed Graph Creation

- **Objective:** To create random directed graphs with different numbers of vertices (n=1000, 2000, 3000, 4000, 5000).
- **Approach:**
- A C program is used to generate the adjacency matrix.
- The adjacency matrix is declared as a 2D array of size $n \times n$.
- Each element `adj[i][j]` is filled with a random number between 0 and x, where x is the maximum number of edges allowed between two vertices.
- Because the graph is **directed**, we do not make the matrix symmetric. That means `adj[i][j]` and `adj[j][i]` can be different.
- Self-loops are possible since a vertex can have an edge to itself.

2. Edge and Degree Calculation

2.1 Counting Edges

- **Objective:** To count the total number of directed edges in the graph.
- **Approach:**
- Each nonzero entry `adj[i][j]` represents an edge from vertex i to vertex j.
- The total number of edges is the sum of all entries of the adjacency matrix.

2.2 Calculating Vertex Degrees

- **Objective:** To calculate in-degree and out-degree of each vertex.
- **Approach:**
- **Out-degree of vertex *i*:** Sum of all entries in row *i*. This shows how many edges leave vertex *i*.
- **In-degree of vertex *i*:** Sum of all entries in column *i*. This shows how many edges come into vertex *i*.

- Total in-degree and total out-degree for the graph are calculated by summing across all vertices.

2.3 Computational Time Measurement

- **Objective**: To measure the time taken to generate the adjacency matrix and calculate degrees.
- **Approach:**
- Use `clock()` from the `<time.h>` library in C.
- Measure the time before and after graph creation and degree calculation.
- Convert the result into milliseconds.
- Printing time is excluded so that only computational work is measured

3. Repetition for Different Values of (n)

- **Objective:** To test the program for different input sizes for (n = 1000, 2000, 3000, 4000, and 5000).
- **Approach:**
- The program is run with different values of ***n*** **(1000, 2000, 3000, 4000, 5000).**
- For each run, the total in-degree, total out-degree, verification of the theorem, and computational time are recorded.

Table of Results:

Number of vertices (n)	Computational time (ms)
1000	6.000
2000	13.000
3000	31.000
4000	55.000
5000	76.000

4. Data Visualization

4.1 Excel Graph Creation

- **Objective:** To visualize how computational time increases with the number of vertices(x).
- **Approach:**
- The experimental data is entered into Microsoft Excel.
- A line graph is created with ***n*** (number of vertices) on the x-axis and computational time on the y-axis.
- A polynomial trendline is added to the graph.
- The equation of the trendline is displayed to help with complexity analysis.

5. Theoretical Time Complexity Analysis

- **Objective**: To analyze the time complexity of the algorithm.
- **Approach:**
- **Graph Generation:** Filling the adjacency matrix requires n^2 operations → O(n^2).
- **Degree Calculation:** Each row and column must be summed → **O(n^2).**
- **Overall Complexity:** Adding both steps still gives **O(n^2).**

Thus, theoretically, the program has quadratic complexity.

6. Comparison of Empirical and Theoretical Complexity

- **Objective:** To compare the time complexity derived from the trendline equation with the theoretical complexity.
- **Approach:**
  - Analyze the polynomial equation obtained from the trendline in Excel and compare it's leading term with the theoretical complexity **(O(n^2)).**

Degrees and time complexity:

- ## Time Complexity Determination (Practical)

From the Excel trendline, the experimental growth fits the equation:

$y = 2E\text{-}06x^2+0.0036x-1.4$

$y=2\times10^{-6}x^2+0.0036x-1.4$ **Let**

$f(x) = 2\times10^{-6}x^2+0.0036x-1.4$

$f(n) = 2\times10^{-6}n^2+0.0036n-1.4$

Then $f(n) = O(g(n))$

$f(n) \leq C\times g(n)$

$2\times10^{-6}n^2+0.0036n-1.4 \leq C\times n^2$ where, $[n^2 > n$ **for** $n \geq 0]$

$2\times10^{-6}n^2+0.0036n-1.4 \leq n^2$ $f(n)$

$= O(n^2)$

This shows that the practical result from experiments matches quadratic complexity.

- ## Time Complexity Determination (Theoretical)

We can also calculate the exact number of operations:

$f(n)=(n+1)(n+1)+ (n+1)(n+1)+1$

Expanding:

$f(n)=(n+1)(n+1)+ (n+1)(n+1)+1$

$=(n^2+n+n+1)+(n^2+n+n+1)+1$

$f(n) = 2n^2+4n+3= O(n^2)$

Since the dominating term is $2n^2$**:**

$f(n)= O(n^2)$

Program Output:

- **For n = 1000 vertices**

```
Enter number of vertices : 1000
Sum of In-degree is = 749959
Sum of Out-degree is = 749959
Sum of in-degree and sum of out-degree are same
Degree computation time: 4.000 ms

Process returned 0 (0x0)   execution time : 3.864 s
Press any key to continue.
```

- **For n = 2000 vertices**

```
Enter number of vertices : 2000
Sum of In-degree is = 2999973
Sum of Out-degree is = 2999973
Sum of in-degree and sum of out-degree are same
Degree computation time: 13.000 ms

Process returned 0 (0x0)   execution time : 3.057 s
Press any key to continue.
```

- **For n = 3000 vertices**

```
Enter number of vertices : 3000
Sum of In-degree is = 6749861
Sum of Out-degree is = 6749861
Sum of in-degree and sum of out-degree are same
Degree computation time: 27.000 ms

Process returned 0 (0x0)   execution time : 4.042 s
Press any key to continue.
```

- **For n = 4000 vertices**

```
Enter number of vertices : 4000
Sum of In-degree is = 12000018
Sum of Out-degree is = 12000018
Sum of in-degree and sum of out-degree are same
Degree computation time: 48.000 ms

Process returned 0 (0x0)   execution time : 3.957 s
Press any key to continue.
```

- **For n = 5000 vertices**

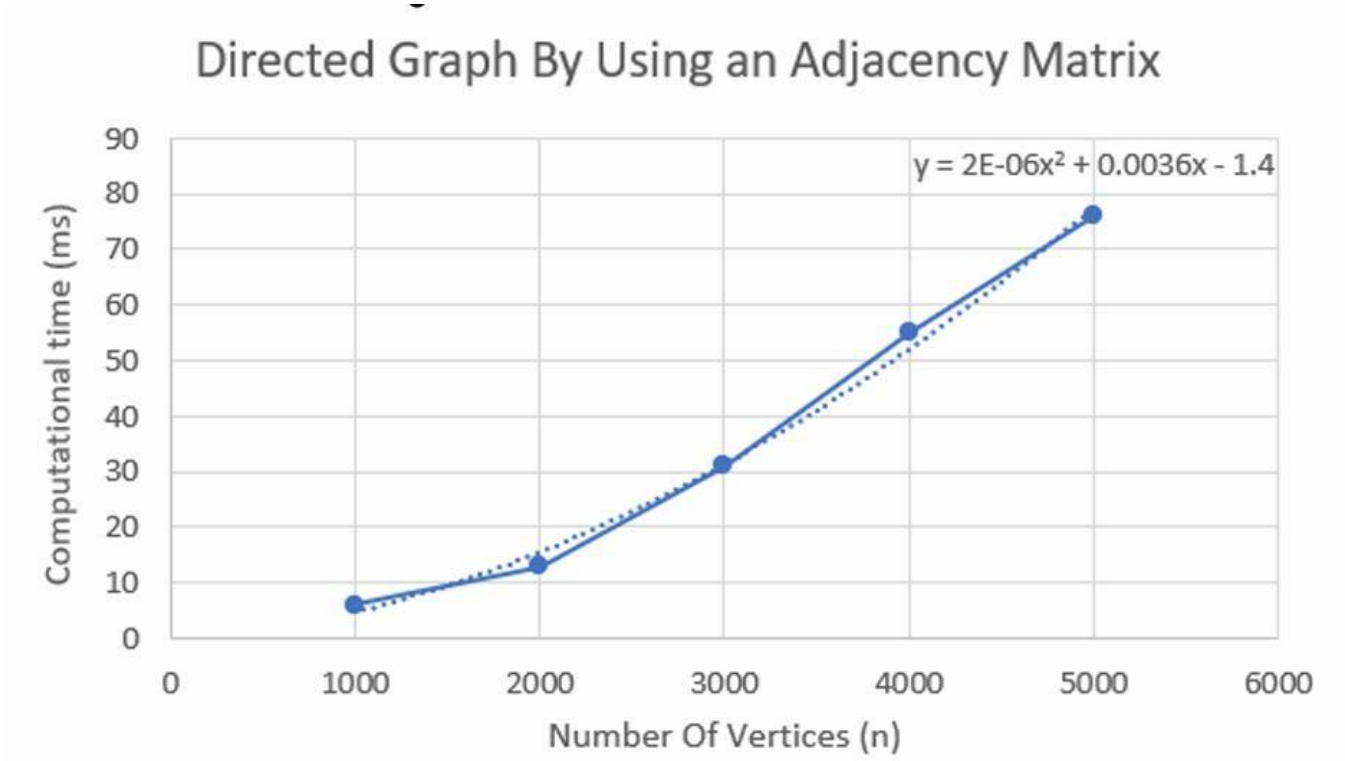
```
Enter number of vertices : 5000
Sum of In-degree is = 18750065
Sum of Out-degree is = 18750065
Sum of in-degree and sum of out-degree are same
Degree computation time: 73.000 ms

Process returned 0 (0x0)   execution time : 3.707 s
Press any key to continue.
```

-

Number of vertices (n)	Computational time (ms)
1000	6.000
2000	13.000
3000	27.000
4000	48.000
5000	73.000

Graph:

Graph showing computational time vs n:



Result: Both practical analysis (Excel trendline) and theoretical analysis confirm that the time complexity of the program is: **O*(n²)***

Time Complexity: O*(n²)*

Conclusion:

This project shows that adjacency matrices can be used effectively to represent **directed graphs**. The adjacency matrix makes it easy to calculate in-degrees and out-degrees, count edges, and check properties.

The results prove that the total in-degree is always equal to the total out-degree, which verifies the theorem for directed graphs.

The computational experiments for different graph sizes confirm that the time required grows quadratically with the number of vertices, which matches the theoretical expectation of $\mathbf{O}(n^2)$.

Thus, adjacency matrices are a clear and simple way to represent directed graphs, even though they require large amounts of memory for very big graphs.

Future Work:

1. **Weighted Directed Graphs**
    - Extend the project to include weighted directed graphs and implement shortest path algorithms like Dijkstra and Bellman-Ford.
2. **Efficient Storage**
    - For very large and sparse graphs, adjacency matrices waste memory. Future work could use adjacency lists or compressed storage to save space.
3. **Parallel Processing**
    - Speed up graph generation and degree calculation by using parallel programming or GPU-based methods.
4. **Dynamic Directed Graphs**
    - Modify the program to support graphs where vertices and edges can be added or removed in real time.
5. **Practical Applications**
    - Apply directed graph analysis to real-life cases such as traffic flow, web links, social media follower networks, or biological networks.

6. **Better Visualization**
    - Use software tools to draw directed graphs with arrows to show edge direction, making the results easier to understand.
7. **Advanced Algorithms**
    - Implement algorithms such as PageRank, strongly connected components, and topological sorting on the generated graphs.