
Ah shit, here we go again.

MINI PROJECT:

Directed graph by adjacency matrix

Presented To:

Sadia Nur Amin
Lecturer,
Dept. Of Computer Science & Engineering(CSE)
East West University(EWU)

Presented By:

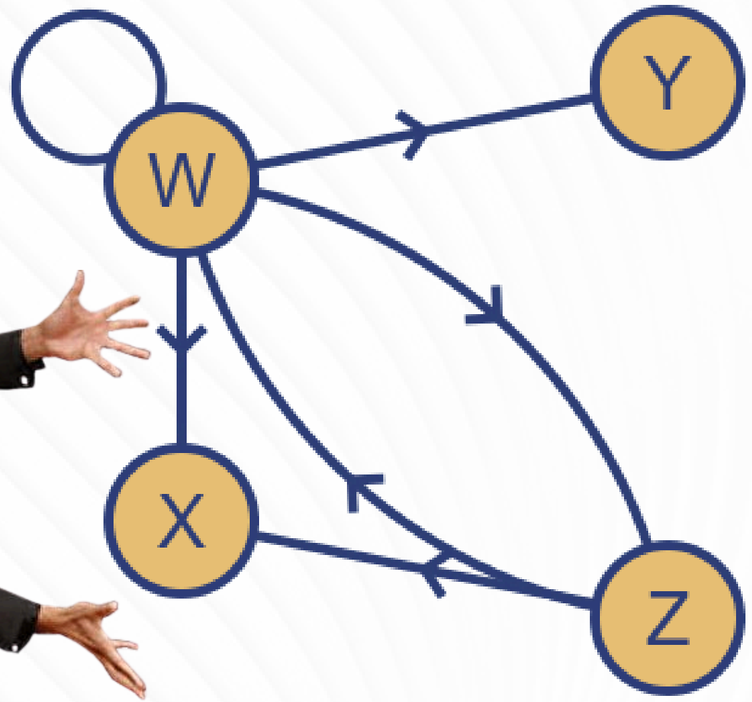
- Mehrab Morshed Marjan (2025-1-60-142)
- Roushna Akter Nowrin (2025-2-60-006)
- Salman Farchi (2025-1-60-052)
- Mitro Khan (2025-1-60-434)

Welcome to our Presentation!!

Randomly generate a Directed Graph represented by Adjacency Matrix

What do you know about Adjacency Matrix???

- An adjacency matrix is a tabular way to represent a graph.
- It shows the connections between different nodes.
- Each row and each column corresponds to a node in the graph.
- If there is a connection between two nodes, the value at their intersecting row and column is 1.
- If there is no connection, the value is 0.
- This matrix makes it easier and convenient to see connections in a graph

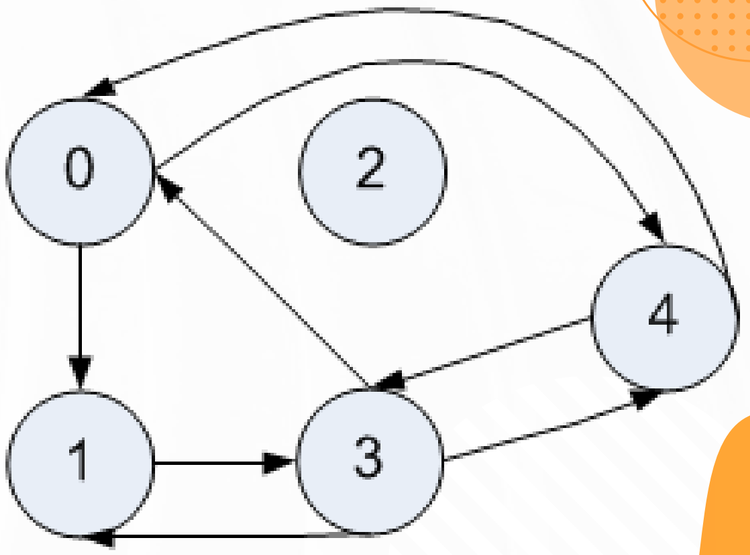


Directed graph with loop

	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0

What is a Directed Graph?

- A directed graph is a type of graph where the connections between the nodes have a defined direction.
- This means that if a node is linked to another, the connection only works one way, unless there is another arrow pointing in the opposite direction.
- The lines that link the nodes are referred to as directed edges or arcs, and they are represented with arrows to indicate the direction.
- In a directed graph, movement from one node to another is only possible in the direction of the arrow. To reverse direction, there must be an arrow pointing the other way



• Glimpse Of The Code

*CSE106 Project Odd.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int adj[5000][5000];
int main() {
    int n;
    int i, j;
    long long in_degree = 0, out_degree = 0;
    double timeMeasure;
    clock_t start, finish;

    printf("Enter number of vertices : ");
    scanf("%d", &n);

    if (n > 5000 || n <= 0) {
        printf("Invalid input! n must be between 1 and 5000.\n");
        return 0;
    }

    srand(time(NULL));

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            adj[i][j] = rand() % 4;
        }
    }

    start = clock();

    for (i = 0; i < n; i++) {
```

• Glimpse Of The Code

*CSE106 Project Odd.c

```
26          }
27      }
28
29      start = clock();
30
31      for (i = 0; i < n; i++) {
32          for (j = 0; j < n; j++) {
33              if (adj[i][j] > 0) {
34                  out_degree++;
35                  in_degree++;
36              }
37          }
38      }
39
40      finish = clock();
41
42      printf("Sum of In-degree is = %lld\n", in_degree);
43      printf("Sum of Out-degree is = %lld\n"  long long main::in_degree
44
45      if (in_degree == out_degree)
46          printf("Sum of in-degree and sum of out-degree are same\n");
47      else
48          printf("Sum of in-degree and sum of out-degree are not same\n");
49
50      timeMeasure = (double)(finish - start) / CLOCKS_PER_SEC * 1000.0;
51      printf("Degree computation time: %.3f ms\n", timeMeasure);
52
53      return 0;
54  }
55
```

• Output

```
Enter number of vertices : 1000
Sum of In-degree is = 749809
Sum of Out-degree is = 749809
Sum of in-degree and sum of out-degree are same
Degree computation time: 6.000 ms
```

```
Enter number of vertices : 2000
Sum of In-degree is = 2999766
Sum of Out-degree is = 2999766
Sum of in-degree and sum of out-degree are same
Degree computation time: 13.000 ms
```

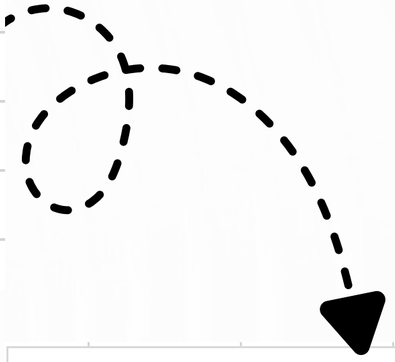
```
Enter number of vertices : 3000
Sum of In-degree is = 6750091
Sum of Out-degree is = 6750091
Sum of in-degree and sum of out-degree are same
Degree computation time: 31.000 ms
```

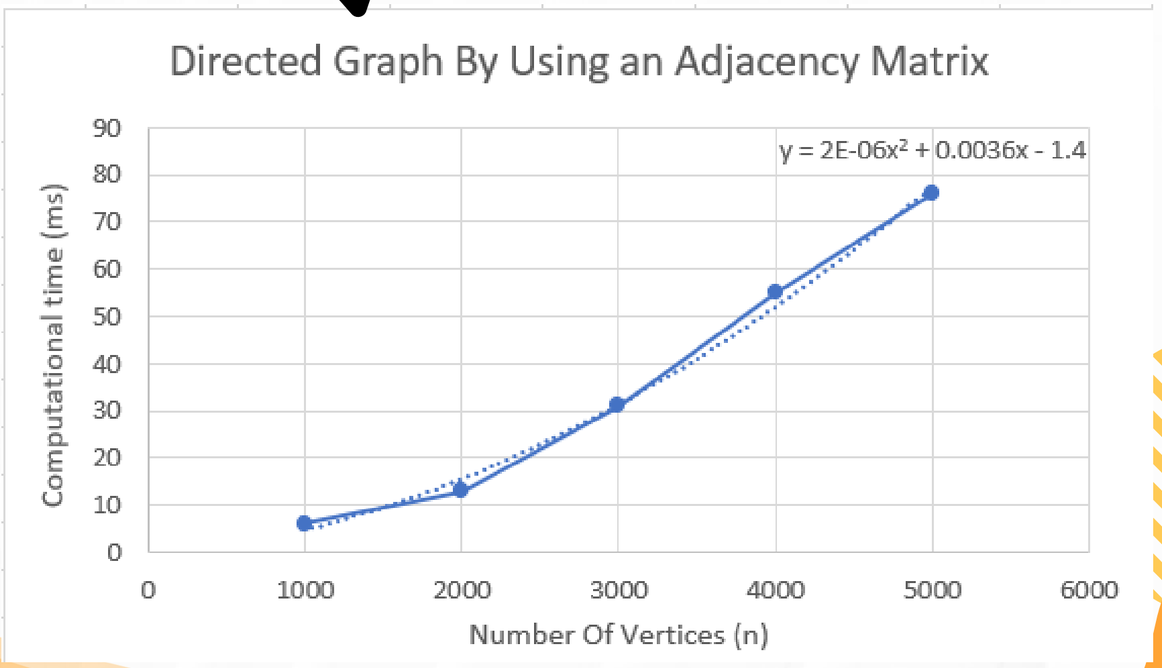
```
Enter number of vertices : 4000
Sum of In-degree is = 12000037
Sum of Out-degree is = 12000037
Sum of in-degree and sum of out-degree are same
Degree computation time: 55.000 ms
```

```
Enter number of vertices : 5000
Sum of In-degree is = 18750045
Sum of Out-degree is = 18750045
Sum of in-degree and sum of out-degree are same
Degree computation time: 76.000 ms
```

• Graph

Number of Vertices (n)	Computational Time (ms)
1000	6
2000	13
3000	31
4000	55
5000	76





Graph :The graph shows the relationship between calculation time (ms) and the variable n, indicating that as n increases, the calculation time steadily grows.

Time Complexity Determination (Practical)

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

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

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

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

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

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

$2\times10^{-6}n^2+0.0036n-1.4 \leq n^2$.

$f(n)=n^2$ (dominates as n grows).

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

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

Time Complexity Determination (Theoretical)

Given the equation:

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

Let:

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

Expanding:

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

Here:

- The quadratic term is $2\times10^{-6}n^2$.
- The linear term is 0.0036n
- The constant term is -1.4

Now, ignore the constant and linear terms because as nnn grows large, the quadratic term will dominate.

Thus, we have:

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

Since the dominant term is $2\times10^{-6}x^2$,we conclude:

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

• Comparison :

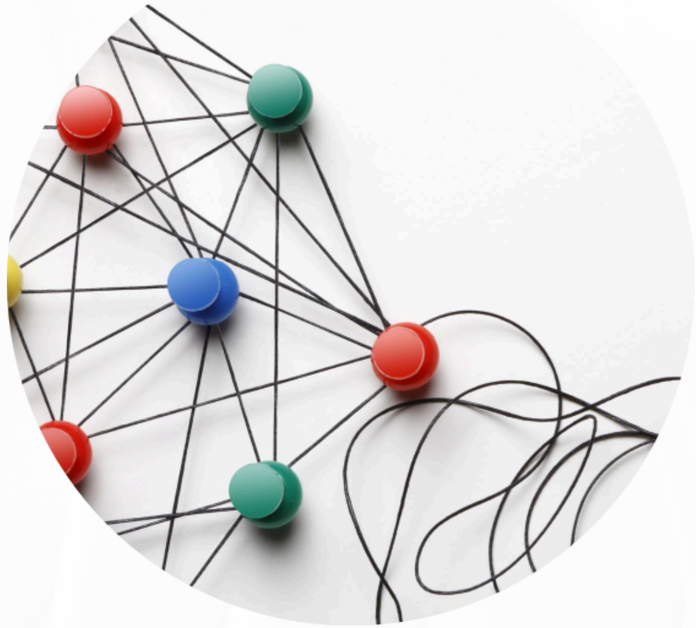
By analyzing the Computational Time vs Number of Vertices graph's equation, we determined the practical time complexity of our program, which is $O(n^2)$.

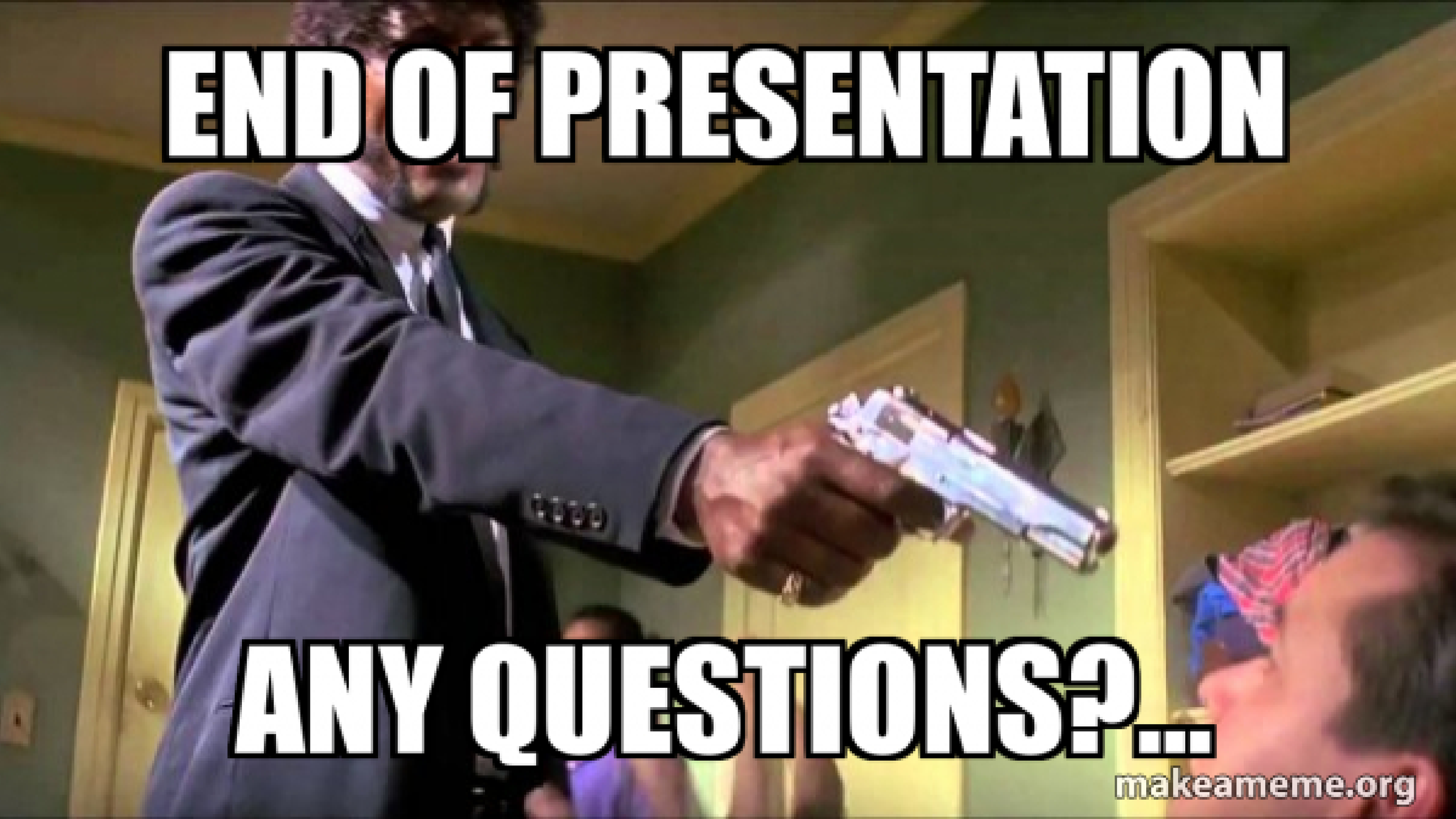
In addition, we also evaluated the time complexity theoretically, and the result was the same — the time complexity of the program is also $O(n^2)$.

Although the specific functions differed slightly in each case, both approaches ultimately lead to the same time complexity for the program: $O(n^2)$.

• Conclusions :

- The adjacency matrix offers a straightforward method to represent directed graphs.
- It is simple to compute the in-degree and out-degree of vertices.
- The equality of in-degree and out-degree has been verified..
- Both theoretical and practical analyses confirm the $O(n^2)$ time complexity
- While it is useful for graph analysis, it can be memory-intensive for larger graphs.




END OF PRESENTATION
ANY QUESTIONS?...
makeameme.org