****

**CSE 106**

**Discrete Mathematics**

**[Spring 2024]**

**Section: 09**

**Submitted to:**

Md Al-Imran

Lecturer

Department of Computer Science and Engineering

East West University

**Project Title**: Randomly generate a directed graph represented by adjacency matrix

**Submitted by:**

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| **2019-1-60-013** | **Md. Abir Hasan Khan** |
| **2023-1-60-267** | **Sams Bin Shahadat** |
| **2023-3-60-368** | **Nusrat Jahan Joba** |

**Scenery of the Project:**

**Introduction**

This project involves developing a program to generate random directed graphs represented by an adjacency matrix. The program ensures that in-degrees equal out-degrees and measures computational time in nanoseconds. We conducted empirical testing across various vertex numbers and plotted computational times against vertices in Excel. Comparing empirical and theoretical time complexities using Big O notation validated our findings, enhancing our understanding of program efficiency.

**Functionality**

The code performs the following tasks:

1. Initializes the number of vertices inputted by the user.
2. Generates a random directed graph using an adjacency matrix.
3. Calculates the in-degree and out-degree for each vertex in the graph.
4. Measures the computational time for graph generation and degree calculation.
5. Outputs the total in-degree, total out-degree, and computational time

**Key Enhancements**

1. **Macro for Maximum Vertices (MAX\_VERTICES):**
   * This macro ensures scalability and manageability by defining the maximum number of vertices allowed in the graph. It enables easy adjustment of graph size without modifying the code structure.
2. **Seeding the Random Number Generator:**
   * The ``srand(time(NULL))`` function seeds the random number generator, ensuring different random sequences on each program run. This prevents the generation of the same graph repeatedly.

**Components**

**Initialization and Input:** Prompt the user to input the number of vertices and validate it against the maximum allowed vertices.

**Graph Generation:** Generate a random adjacency matrix representing the directed graph. Ensure consistency in vertex degrees.

**Degree Calculation:** Calculate the in-degree and out-degree for each vertex using the adjacency matrix.

**Timing and Output:** Measure computational time in nanoseconds using clock functions. Output total in-degree, total out-degree, and computational time.

**Formulation of f(n)**

The total comparisons is the function here which is,

(n²+2n+1) + (3n²+2n+1)

=4n²+4n+2

So, the function is f(n)= 4n²+4n+2

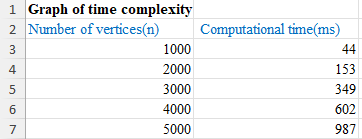
Now, we need to find time complexity by using this function’s big-O.

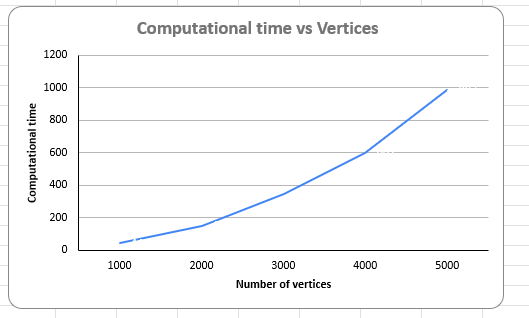
Finding Big-O from this function for time complexity:

4n²+4n+2< 4n²+4n² + n² [When, n>2]

=9n² 2 So, C=9, K=2 Hence, the time complexity is 4n²+4n+2= O(n²)

**Computational time vs Vertices**





**Time complexity from polynomial equation**

y=43.5x²-27.5x+31

y(x)= 43.5x²-27.5x+31

= 43.5x²-27.5x+31 <= 43.5x²-27.5x²+31x²

= 47x²

When k=1, c=47, g(x) = n²

y(x)= 43.5x²-27.5x+31

y(n)= O(n²)

**Compare of graph and source code time complexity**

* The time complexity found from the graph is, y(n) = O(n²)
* Time complexity calculated from the source code is, f (n) = O(n²)
  + We can see that the time complexity calculated from the graph is
  + same as the time complexity calculated from the source code.

**Analysis**

**Empirical Time Complexity:** We tested the program with different numbers of vertices and measured the computational time. By plotting the computational time against the number of vertices in Microsoft Excel, we conducted a graphical analysis to understand the empirical time complexity.

**Theoretical Time Complexity:** We compared the empirical results with theoretical predictions based on Big O notation to validate the program's performance.

**Conclusion**

Our program effectively generates random directed graphs with consistent vertex degrees and provides insights into computational time. The empirical analysis aligns with theoretical predictions, validating our approach and findings.