

**CSE106 Discrete Mathematics**

**Section: 01**

**Report of Mini Project**

**Directed graph represented by adjacency matrix by using C program**

**Submitted by:** Group 01

|  |  |
| --- | --- |
| Student ID | Student Name |
| 2022-3-60-123 | Md. Nura Alam Naim |
| 2022-3-60-127 | Tanvir Rahman |
| 2022-3-60-182 | Md. Mahamudur Rahman Maharaz |

**Submitted To:**

Dr. Md Mozammel Huq Azad Khan

Professor

Department of Computer Science & Engineering, EWU

**Date of Submission:**

21.05.2023

Project Overview:  
This project aims to create a directed graph and analyze its properties using a C program. The main steps are:

Random graph generation:  
We developed a C program that generates a randomly directed graph represented by an adjacency matrix. The graph has a fixed number of vertices initially set to 1000. Each entry in the adjacency matrix represents an edge between two vertices, and its value indicates the presence or absence of the edge.

Closing calculation:  
Determined the degree of on and off for each vertex in the generated graph. A vertex's indegree is the number of input edges and the output degree is the number of output edges. Checked that the sum of the on degrees of all vertices equals the sum of the off degrees. Measured the computation time required for these step-in milliseconds (MS).

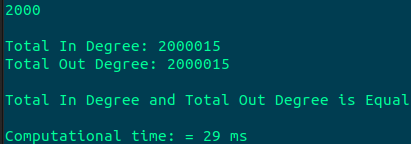
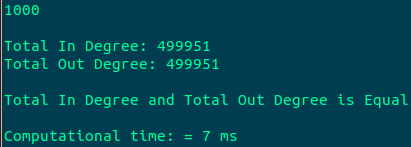
Scalability test:  
Repeated steps 1 and 2 for different values of n, specifically 2000, 3000, 4000, and 5000. This gave an idea of how the program will perform as the size of the chart increases.

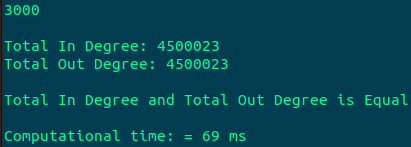
Time complexity analysis:  
Using a Microsoft Excel worksheet, created a line chart representing computation time vs n. Added a trendline using the Polynomial option to determine the ordinal value that best fits the graph. Graph the equation that gives an estimate of the program's time complexity as a function of n.

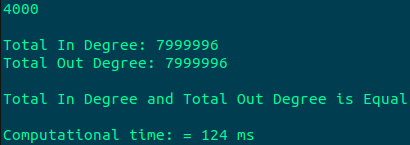
Theoretical analysis:  
We theoretically analyzed the time complexity of programs based on algorithms and implementations. Compared the theoretical time complexity with the time complexity derived from the trend line equation in step 4 to assess the accuracy of the analysis. Following these steps, the project aims to generate a random directed graph, analyze its properties, and determine the program's time complexity with respect to the input size.

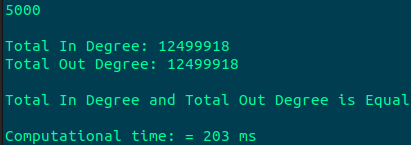
Output of the Program:

Here is the output of the program for different vertexes where n = 1000, 2000,3000,4000,5000.  
In every example the in degrees and the out degrees are equal.



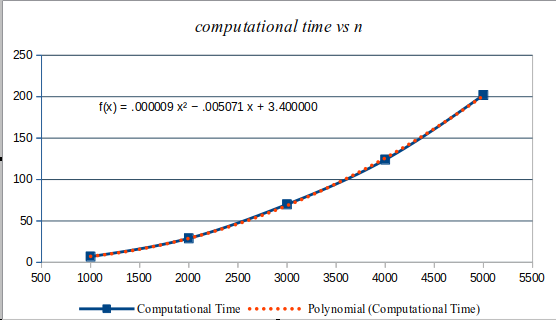






**Graph Showing Computational time vs the number of vertices:**

|  |  |
| --- | --- |
| Number of vertexes (n) | Computational Time in MS |
| 1000 | 7 |
| 2000 | 29 |
| 3000 | 69 |
| 4000 | 124 |
| 5000 | 203 |



Equation:

Theoretical time complexity:

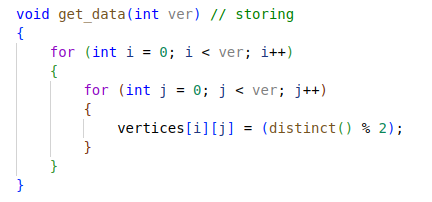
Let, n be the vertex of our matrix. So, here we are considering n as ver variable for our benefit. In the main Function we have called getData() function which is creating the matrix with random value which is generated by the distinct function. and then called calc\_Degree() function, which is calculating the in and out degree.

|  |  |
| --- | --- |
| Statement | Time Complexity |
| **{**  **{**    **}**  **}** | The inner &outer for loops will be executed n times;  For each loop one comparison will be done.  One more comparison will be needed to exit the both *for* loops. Therefore, exactly comparisons are used.  Hence, the average-case time complexity is |
| **{**  **{**      **}**  **}** | The inner &outer for loops will be executed n times;  For each loop one comparison will be made.  One more comparison will be needed to exit the both *for* loops. Therefore, exactly  comparisons are used.  Hence, the average-case time complexity is |
| The average-time Complexity of the algorithm is: |  |

Comparison: From the graph, we get the average-time complexity as a function of n, , and theoretically from Big-theta notation we get the time complexity, f(n) = So, both the time complexity from the same program is the same.

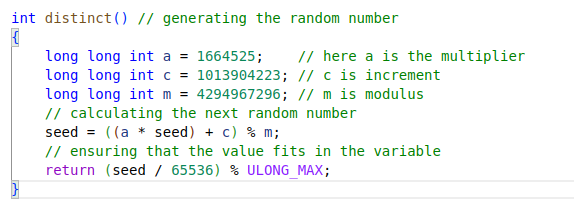
Description of Functions we used in the program:

* get\_data Function:



This function is creating the random Matrix. Here it is generating the random value with a distinct function.

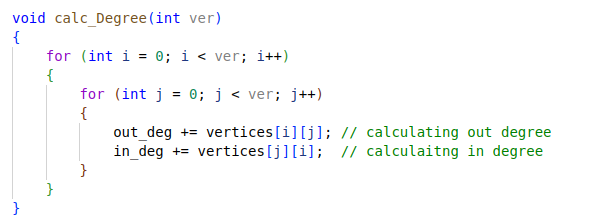
* Distinct Function:



Here it is generating pseudorandom numbers in the linear congruential method where,  
a is the multiplier, c is increment and m is modulo. The values of a, m and c are taken as commonly used constants in the linear congruential generator (LCG) algorithm for generating pseudo-random numbers. These values have been chosen based on their properties.

Here, the value of seed is updated within the distinct() function. The updated seed value is then used to generate the next random number by dividing it by 65536 and taking the modulus of ULONG\_MAX (the maximum value for an unsigned long integer).   
This operation ensures that the generated random number fits within the range of ULONG\_MAX

* calc\_degree Function:



This function is calculating the in and out degree from the randomly generated matrix. Here we are calculating the in degree by adding value column wise and calculating the out degree by row wise. After completing the nested for loop, we will get the in degrees and out degrees.