CSE106: Discrete Mathematics

Spring 2024

PROJECT REPORT

Adjacency Matrix of Directed Graph

Course Code: CSE106

Course Title: Discrete Mathematics

Section: 12

Group Number: 9

Group Name: Boolean Autocrats

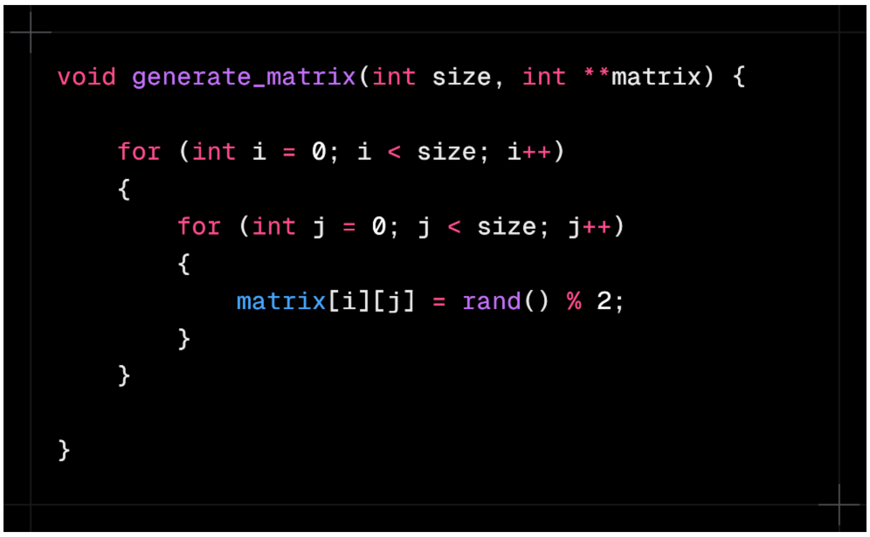
Submitted By:

Student ID	Student Name	Contribution Percentage
2023-3-60-186	Md. Hasib Ali Siam	33%
2023-3-60-125	Mosabbir Ahmed	34%
2023-3-60-539	Sadia Khan Farhin	33%

Introduction:

Using a C program, the **adjacency matrix** of a **directed graph** consisting of only 0 and 1, was randomly generated. Then the sum of in-degrees and out-degrees were calculated and it was shown that the sums were equal. Moreover, the computational time of completing the previous task as a function of the number of vertices was calculated and the collected data was used in Excel to make a graph chart. The data was collected by performing multiple trials and calculating the average computational time. Finally, from the equation found in the Excel graph and the theoretical calculation, the time complexity was determined and compared.

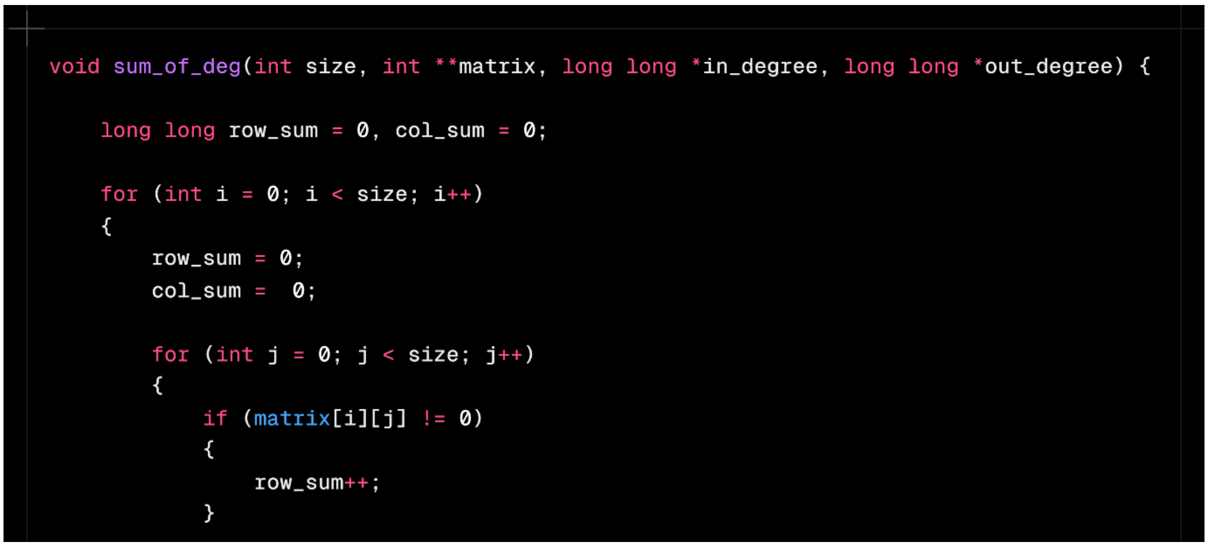
Random Matrix Generating Function:



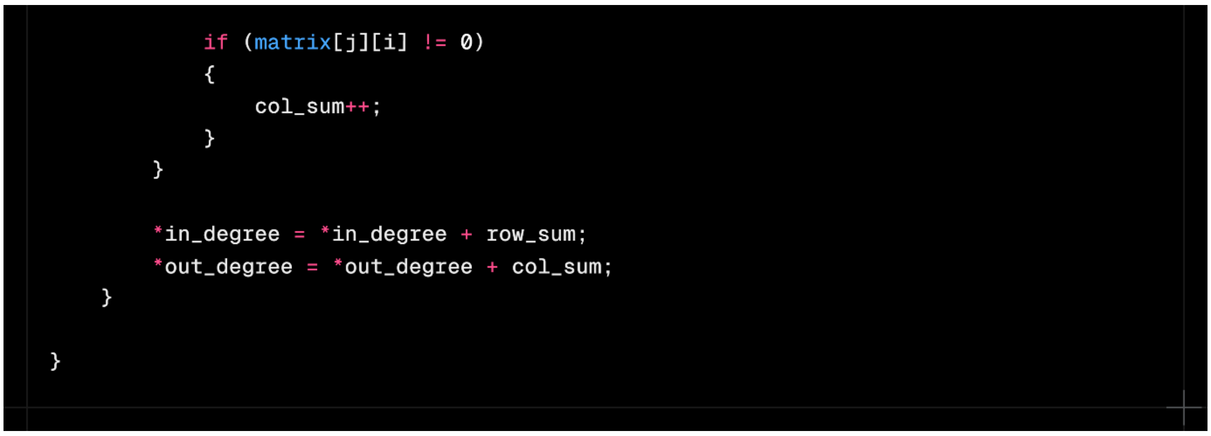
```c
void generate_matrix(int size, int **matrix) {

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            matrix[i][j] = rand() % 2;
        }
    }

}
```

Two function parameters are used, one for the number of vertices which is **size** and one for the 2D array for the adjacency matrix named **matrix**, for the 2D array, double pointer is being used for dynamic memory allocation. Using two for loops, the rows and columns of the matrix are accessed and by using the **rand( )** function from **stdlib.h**, a random number between 0 and 1 is being generated. Thus, the whole matrix of (size x size) is randomly generated.

Sum of Degrees Calculating Function:



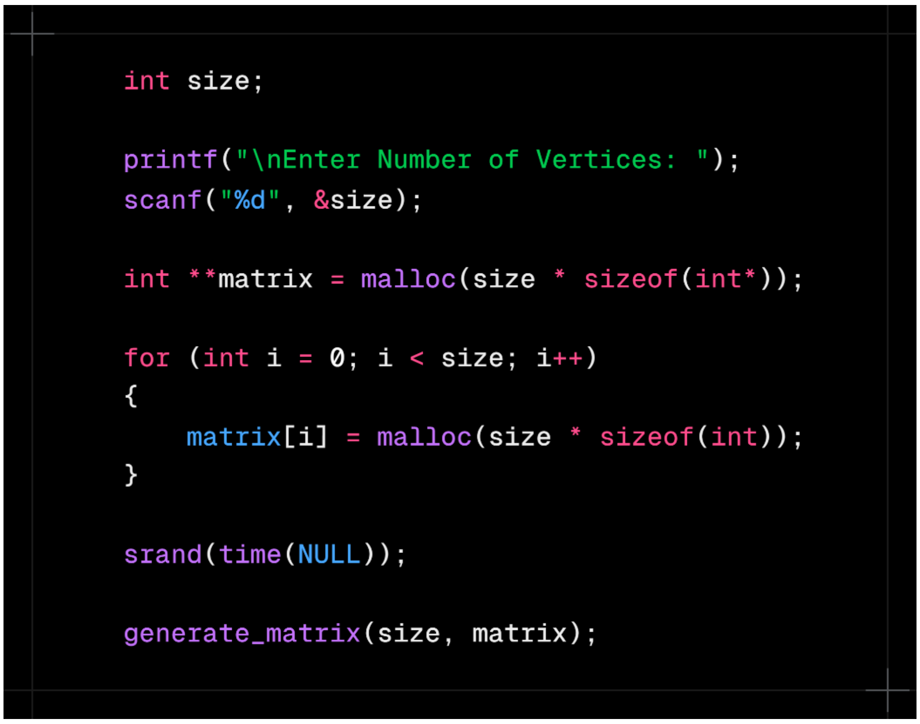
```c
void sum_of_deg(int size, int **matrix, long long *in_degree, long long *out_degree) {

    long long row_sum = 0, col_sum = 0;

    for (int i = 0; i < size; i++)
    {
        row_sum = 0;
        col_sum =  0;

        for (int j = 0; j < size; j++)
        {
            if (matrix[i][j] != 0)
            {
                row_sum++;
            }
```



```
            if (matrix[j][i] != 0)
            {
                col_sum++;
            }
        }

        *in_degree = *in_degree + row_sum;
        *out_degree = *out_degree + col_sum;
    }

}
```

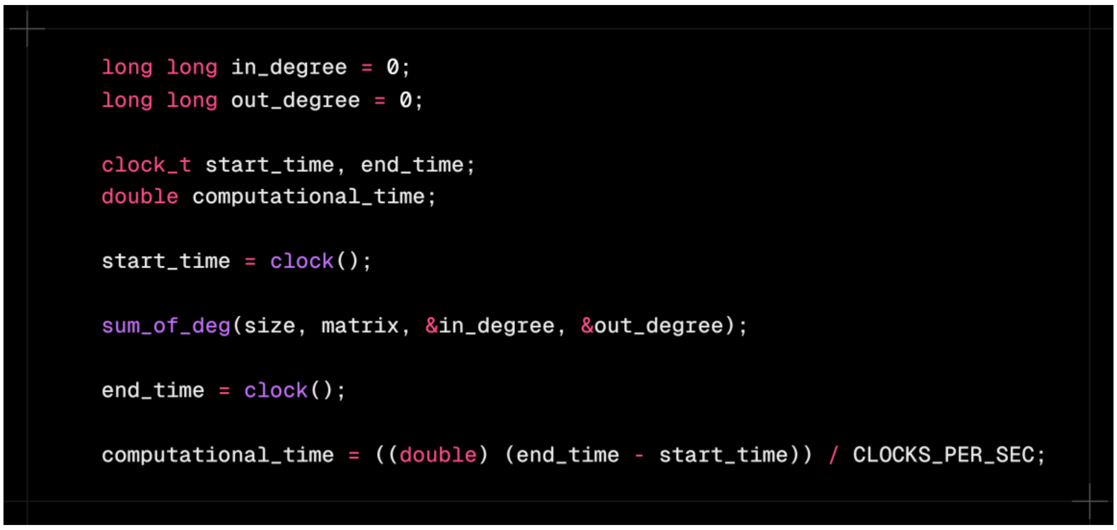
In this function, four parameters are used, for number of vertices, for 2D adjacency matrix using double pointer, for storing sum of in-degrees and for sum of out-degrees. For the sum, **long long** data type is used since the data can be really big depending the size of the matrix. Two variables are declared with initial value of 0 for storing the sum of degrees of one row and one column. Two for loops are used to access the rows and columns of the matrix, at the start of each iteration of the outer loop, degree sum of row and column is set to 0. For every non-zero element of every row and column, the **row_sum** and **col_sum** is incremented by 1. At the end of each iteration of the outer loop, the sums are added to the function arguments using pointers since we cannot return more than one value.

Main Function:



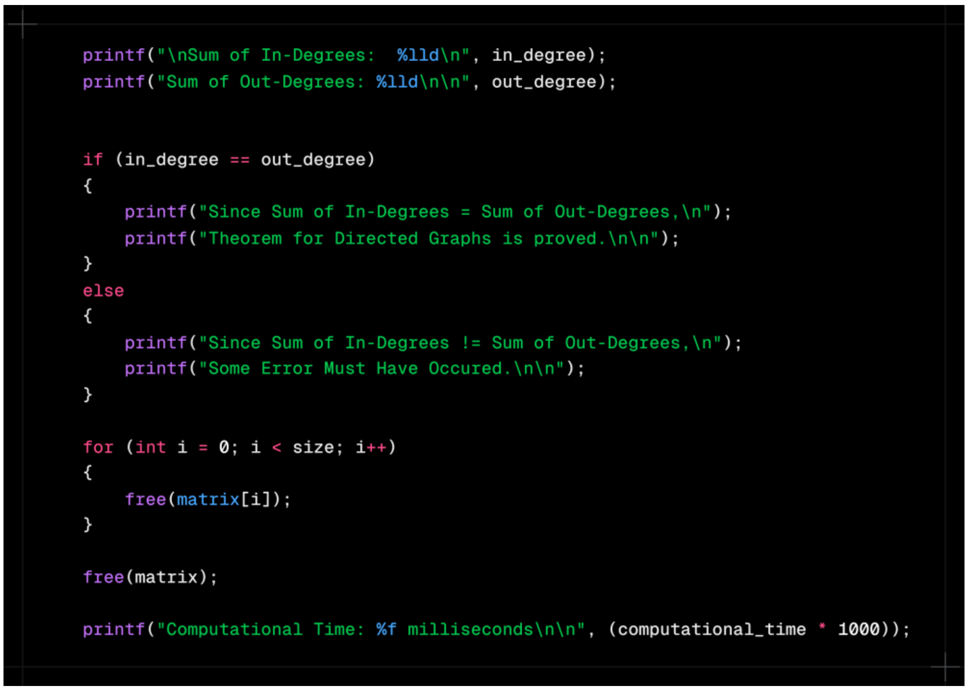
```
int size;

printf("\nEnter Number of Vertices: ");
scanf("%d", &size);

int **matrix = malloc(size * sizeof(int*));

for (int i = 0; i < size; i++)
{
    matrix[i] = malloc(size * sizeof(int));
}

srand(time(NULL));

generate_matrix(size, matrix);
```

First of all, the number of vertices is declared as **size** for taking input from the user using **scanf( )**, it is for making the program dynamic rather than static. Then using double pointer, a 2D array named **matrix** is declared, and using **malloc( )** function from **stdlib.h**, memory for the 2D array is dynamically stored during runtime. Dynamic memory allocation is used because of the large size of data, which can cause problem in case of static memory allocation during compile time. Before randomly generating the matrix with the **generate_matrix( )** function, the random number generator is being seeded using the **srand( )** and **time( )** functions from **stdlib.h** and **time.h** respectively.



```c
long long in_degree = 0;
long long out_degree = 0;

clock_t start_time, end_time;
double computational_time;

start_time = clock();

sum_of_deg(size, matrix, &in_degree, &out_degree);

end_time = clock();

computational_time = ((double) (end_time - start_time)) / CLOCKS_PER_SEC;
```

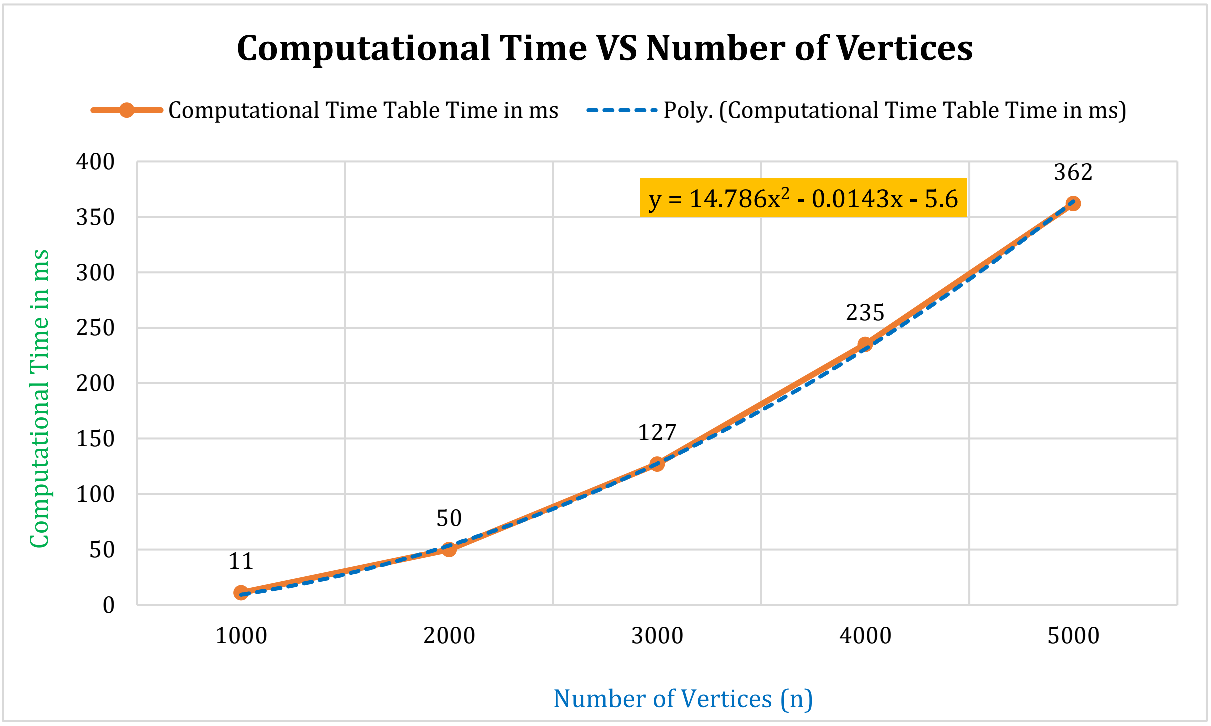
After That, two **long long** variables are declared with value 0 to store the sum of in-degrees and out-degrees. Two variables of data type **clock_t** are declared to store the processor time in **ticks**, one to use before the sum calculating function and one to use immediately after. A **double** variable is declared to store the computational time in seconds. Then the **sum_of_deg( )** function is used for calculating the sums. Before and after this function, the **start_time** and **end_time** variable are used using the **clock( )** function from **time.h** to store the processor time in **ticks**. The computational time is calculated in seconds by dividing the difference between the end and starting time in **ticks** by the constant from **time.h**, **CLOCKS_PER_SEC**, which is the number of **ticks per second**.



```c
printf("\nSum of In-Degrees:  %lld\n", in_degree);
printf("Sum of Out-Degrees: %lld\n\n", out_degree);


if (in_degree == out_degree)
{
    printf("Since Sum of In-Degrees = Sum of Out-Degrees,\n");
    printf("Theorem for Directed Graphs is proved.\n\n");
}
else
{
    printf("Since Sum of In-Degrees != Sum of Out-Degrees,\n");
    printf("Some Error Must Have Occured.\n\n");
}

for (int i = 0; i < size; i++)
{
    free(matrix[i]);
}

free(matrix);

printf("Computational Time: %f milliseconds\n\n", (computational_time * 1000));
```

Finally, The results are printed using **printf( )**, along with the statement that the theorem of directed graphs is proved. Then, the memory allocated for the 2D matrix during runtime is freed using the **free( )** function from **stdlib.h**. At the end, the computational time is printed in milliseconds by multiplying the **computational_time** variable with 1000.

Computational Time VS Number of Vertices Graph:



The data for computational time was collected by performing multiple trials and calculating the average time for each of the 5 values of **(n)**. From the trendline option, polynomial of order 2 was found to be suitable for the graph. A polynomial equation was also determined for further use.

Time Complexity Analysis:

From the above graph, we found the equation $y = 14.786x^2 - 0.0143x - 5.6$, where $x = n\ (Number\ of\ Vertices)$. So, the expression stands as $an^2 + bn + c$, where a, b and c are constants. Since the dominant term here is n^2, the time complexity is $\mathbf{O(n^2)}$.

Now, theoretically, in the sum of in-degrees and out-degrees function, if the size of the matrix is $\boldsymbol{n}$, the inner loop iterates $\boldsymbol{n}$ times, in each iteration there are two comparisons which involve constants, therefore these do not contribute to the time complexity. To exit the inner loop, there is an additional comparison when $\boldsymbol{j}$ becomes greater than equal $\boldsymbol{n}$. So, the inner loop contributes $(\boldsymbol{n} + \mathbf{1})$. The outer loop also iterates $\boldsymbol{n}$ times, and there is an additional comparison when $\boldsymbol{i}$ becomes greater than equal $\boldsymbol{n}$ to exit the outer loop. But, since the exit of the outer loop occurs after the total completions of the inner loop, the expression becomes $\boldsymbol{n}(\boldsymbol{n} + \mathbf{1}) + \mathbf{1} = \boldsymbol{n}^{\mathbf{2}} + \boldsymbol{n} + \mathbf{1}$. As a result, considering the dominant term $\boldsymbol{n}^{\mathbf{2}}$, the time complexity stands as $\mathbf{O(n^2)}$.Since the both of the time complexities are the same, we can conclude that the calculations were done correctly.

Conclusion:

The C program has been tested properly and it fulfills all the requirements, the graph was generated carefully, and the calculated and theoretical time complexities were proved to be equal. Therefore, our project is declared to be complete.