 EAST WEST UNIVERSITY

**Course Code: CSE 106/205**

**Course DISCRETE MATHEMATICS**

**Section: 06**

***Mini project***

***on***

Directed Graph

***Submitted To:***

Dr. Mohammad Salah Uddin

Assistant Professor,

Dept. of CSE, East West University

***Submitted By:***

Sk Mohammad asem 2017-3-60-068

Ankon Nandi 2022-1-60-131

Kamrun Nahar 2022-1-60-301

**Date of Submission:** 06 September 2022

INTRODUCTION

**C** programming language is very fast in terms of execution as it does not have any additional processing overheads such as garbage collection or preventing memory leaks etc. Therefore, we can use C programming language for generating a directed graph by using adjacency matrix, sum of in degree and out degree, and time complexity without any garbage values.

**A** directed graph is a set of nodes connected by edges, where the edges have a direction associated with them. E.g. an arc (**x**, **y)** is considered to be directed from x to y, and the arc (**y**, **x**) is the inverted link; where y is a direct successor of x, and x is a direct predecessor of y.

The In - degree of vertex **v** is **deg^**(**+**)(**v**) is the number of edges. To find In- degree of a vertex, count the number of edges ends at the vertex .The Out-degree of vertex **v** is **deg**^(**-**)(**v**) is the number of edges .To find out degree of a vertex, count the number of edges starting from the vertex.

**A**n adjacency matrix is compact ways represent the finite graph. If a graph have **n** vertices, its adjacency matrix is an (**n** × **n)** matrix ;where each entry represents the number of edges from one vertex to another.

**I**n computer science, the time complexity is the computational complexity that describes the amount of computer time that takes to run an algorithm.

**Firstly**,we prepared a code by using C programming for randomly generate a directed graph which represented by an adjacency matrix with **n** = **1000** vertices. **Secondly,** it expressed the sum of **In-degree** and **Out-degree** which are equal. It also determined the computational time .**Thirdly,** by repeating the vertices of **n**= **2000,3000,4000,5000** we have solved the time complexity problem of the program. Then, there have a graph showing computational time **vs.** vertices. From the graph we found a time complexity. **Finally,** we theoretically determined of the computational time complexity of the program as the function of the vertices **n** for **1000, 2000, 3000, 4000, 5000.**Then we compared this time with the time complexity of the graph.

**Degree & Complexity :-** Consequently, we can say that for every different value of **n** we have found the respective In-degree and Out -degree of the code. From this, we can also say that we found in every number of vertices of same Out-degree and same In-degree. When we analysis the graph we observed that the computational timeincreased when the values of **n** also increased. Hence we can also say that the computational timedepends on the vertices n**.**

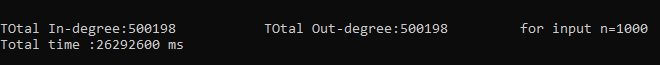
In conclusion**,** we can say that the highest time complexity is

**Big O**(**)**Therefore, the approximate complexity will be **Big O**(**)**.

Output:

* When n =1000

       Total In-Degree=500198 &  Total Out-degree=500198

  Computational time in ms =26292600ms 

* When n =2000

       Total In-Degree=1999879 &  Total Out-degree=1999879

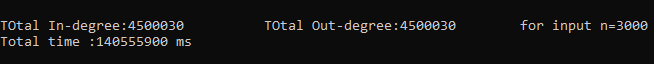
  Computational time in ms =62513800ms



* When n =3000

       Total In-Degree=4500030 &  Total Out-degree=4500030

  Computational time in ms =140555900ms



* When n =4000

       Total In-Degree=8000013 &  Total Out-degree=8000013

  Computational time in ms =234319900ms



* When n =5000

       Total In-Degree=12500068 &  Total Out-degree=12500068

  Computational time in ms =394738800ms



Time Complexity Of The Program

* To generate the matrix:

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

in\_arr[i]=0;

out\_arr[i]=0;

for(j=0;j<n;j++){

arr[i][j]=rand()%2;

This is a nested for loop. Here, the outer for loop executes n times and the inner for loop executes n+1 times because there is 1 comparison in the inner for loop.

So, f1=n\*(n+1)=n^2+n=O(n2).

* To print the matrix:

for(i=0;i<n1;i++){

for(j=0;j<n1;j++){

printf(“%d”,arr[i][[j]);

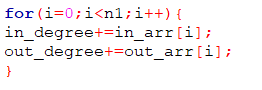
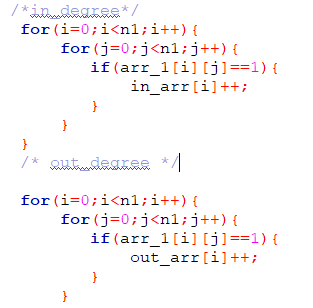
}

This is a nested for loop. Here, the outer for loop executes n times and the inner for loop executes n times because there is no comparison in the inner loop.

So, f2=n\*n=n^2=O(n2)

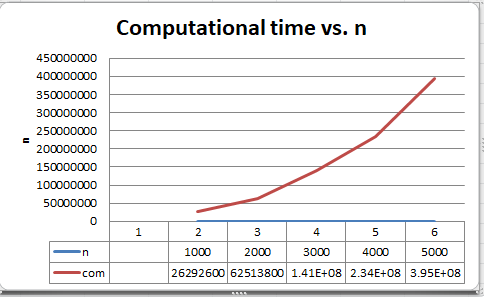
* To count the in-degrees and out-degrees: This is a nested for loop. Here, the outer for loop executes n times and the inner for loop executes (n+2) times because there is 2 if function which executes 1 times each.

So, f3=n\*(n+2)=n^2+2n=O(n2)



Graph

Here we can see the t=graph of computational time vs n. Where the value of n increased, the computational time as well increased.



Source Code:

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void main()

{

struct timespec starts,ends;

printf("---------------Directed\_graph----------------\n");

int n1,n2,n3,n4,n5;

printf("Enter the input: \n");

scanf("%d%d%d%d%d",&n1,&n2,&n3,&n4,&n5);

int i,j;

int \*arr\_1[n1],\*arr\_2[n2],\*arr\_3[n3],\*arr\_4[n4],\*arr\_5[n5];

int \*in\_arr,\*out\_arr;

int in\_degree=0,out\_degree=0;

long long time1=0,time2=0,time3=0,time4=0,time5=0;

in\_degree=0,out\_degree=0; /\* when n=1000 \*/

in\_arr = (int\*)malloc(n1 \* sizeof(int));

out\_arr = (int\*)malloc(n1 \* sizeof(int));

for(i=0; i<n1 ; i++){

arr\_1[i]=(int\*)malloc(n1 \* sizeof(int));

}

for(i=0;i<n1;i++){

in\_arr[i]=0;

out\_arr[i]=0;

for(j=0;j<n1;j++){

arr\_1[i][j]=rand()%2;

}

}

clock\_gettime(CLOCK\_REALTIME,&starts);

/\*in\_degree\*/

for(i=0;i<n1;i++){

for(j=0;j<n1;j++){

if(arr\_1[i][j]==1){

in\_arr[i]++;

}

}

}

/\* out\_degree \*/

for(i=0;i<n1;i++){

for(j=0;j<n1;j++){

if(arr\_1[i][j]==1){

out\_arr[i]++;

}

}

}

clock\_gettime(CLOCK\_REALTIME,&ends);

for(i=0;i<n1;i++){

in\_degree+=in\_arr[i];

out\_degree+=out\_arr[i];

}

/\* calculate total time \*/

time1=(ends.tv\_sec-starts.tv\_sec)\*1000+(ends.tv\_nsec-starts.tv\_nsec);

printf("\n\nTOtal In-degree:%6d \t\t TOtal Out-degree:%6d \t for input n=%d\n",in\_degree,out\_degree,n1 );

printf("Total time :%ld ms\t",time1);

in\_degree=0,out\_degree=0; /\* when n=2000 \*/

in\_arr = (int\*)malloc(n2 \* sizeof(int));

out\_arr = (int\*)malloc(n2 \* sizeof(int));

for(i=0; i<n2 ; i++){

arr\_2[i]=(int\*)malloc(n2 \* sizeof(int));

}

for(i=0;i<n2;i++){

in\_arr[i]=0;

out\_arr[i]=0;

for(j=0;j<n2;j++){

arr\_2[i][j]=rand()%2;

}

}

clock\_gettime(CLOCK\_REALTIME,&starts);

/\*in\_degree\*/

for(i=0;i<n2;i++){

for(j=0;j<n2;j++){

if(arr\_2[i][j]==1){

in\_arr[i]++;

}

}

}

/\* out\_arr \*/

for(i=0;i<n2;i++){

for(j=0;j<n2;j++){

if(arr\_2[i][j]==1){

out\_arr[i]++;

}

}

}

clock\_gettime(CLOCK\_REALTIME,&ends);

for(i=0;i<n2;i++){

in\_degree+=in\_arr[i];

out\_degree+=out\_arr[i];

}

/\* calculate total time \*/

time2=(ends.tv\_sec-starts.tv\_sec)\*1000+(ends.tv\_nsec-starts.tv\_nsec);

printf("\n\nTOtal In-degree:%6d \t TOtal Out-degree:%6d \t for input n=%d\n",in\_degree,out\_degree,n2 );

printf("Total time :%ld ms\t",time2);

in\_degree=0,out\_degree=0; /\* when n=3000 \*/

in\_arr = (int\*)malloc(n3 \* sizeof(int));

out\_arr = (int\*)malloc(n3 \* sizeof(int));

for(i=0; i<n3 ; i++){

arr\_3[i]=(int\*)malloc(n3 \* sizeof(int));

}

for(i=0;i<n3;i++){

in\_arr[i]=0;

out\_arr[i]=0;

for(j=0;j<n3;j++){

arr\_3[i][j]=rand()%2;

}

}

clock\_gettime(CLOCK\_REALTIME,&starts);

/\*in\_degree\*/

for(i=0;i<n3;i++){

for(j=0;j<n3;j++){

if(arr\_3[i][j]==1){

in\_arr[i]++;

}

}

}

/\* out\_arr \*/

for(i=0;i<n3;i++){

for(j=0;j<n3;j++){

if(arr\_3[i][j]==1){

out\_arr[i]++;

}

}

}

clock\_gettime(CLOCK\_REALTIME,&ends);

for(i=0;i<n3;i++){

in\_degree+=in\_arr[i];

out\_degree+=out\_arr[i];

}

/\* calculate total time \*/

time3=(ends.tv\_sec-starts.tv\_sec)\*1000+(ends.tv\_nsec-starts.tv\_nsec);

printf("\n\nTOtal In-degree:%6d \t TOtal Out-degree:%6d \t for input n=%d\n",in\_degree,out\_degree,n3 );

printf("Total time :%ld ms\t",time3);

in\_degree=0,out\_degree=0; /\* when n=4000 \*/

in\_arr = (int\*)malloc(n4 \* sizeof(int));

out\_arr = (int\*)malloc(n4 \* sizeof(int));

for(i=0; i<n4 ; i++){

arr\_4[i]=(int\*)malloc(n4 \* sizeof(int));

}

for(i=0;i<n4;i++){

in\_arr[i]=0;

out\_arr[i]=0;

for(j=0;j<n4;j++){

arr\_4[i][j]=rand()%2;

}

}

clock\_gettime(CLOCK\_REALTIME,&starts);

/\*in\_degree\*/

for(i=0;i<n4;i++){

for(j=0;j<n4;j++){

if(arr\_4[i][j]==1){

in\_arr[i]++;

}

}

}

/\* out\_arr \*/

for(i=0;i<n4;i++){

for(j=0;j<n4;j++){

if(arr\_4[i][j]==1){

out\_arr[i]++;

}

}

}

clock\_gettime(CLOCK\_REALTIME,&ends);

for(i=0;i<n4;i++){

in\_degree+=in\_arr[i];

out\_degree+=out\_arr[i];

}

/\* calculate total time \*/

time4=(ends.tv\_sec-starts.tv\_sec)\*1000+(ends.tv\_nsec-starts.tv\_nsec);

printf("\n\nTOtal In-degree:%6d \t TOtal Out-degree:%6d \t for input n=%d\n",in\_degree,out\_degree,n4 );

printf("Total time :%ld ms\t",time4);

in\_degree=0,out\_degree=0; /\* when n=5000 \*/

in\_arr = (int\*)malloc(n5 \* sizeof(int));

out\_arr = (int\*)malloc(n5 \* sizeof(int));

for(i=0; i<n5 ; i++){

arr\_5[i]=(int\*)malloc(n5 \* sizeof(int));

}

for(i=0;i<n5;i++){

in\_arr[i]=0;

out\_arr[i]=0;

for(j=0;j<n5;j++){

arr\_5[i][j]=rand()%2;

}

}

clock\_gettime(CLOCK\_REALTIME,&starts);

/\*in\_degree\*/

for(i=0;i<n5;i++){

for(j=0;j<n5;j++){

if(arr\_5[i][j]==1){

in\_arr[i]++;

}

}

}

/\* out\_arr \*/

for(i=0;i<n5;i++){

for(j=0;j<n5;j++){

if(arr\_5[i][j]==1){

out\_arr[i]++;

}

}

}

clock\_gettime(CLOCK\_REALTIME,&ends);

for(i=0;i<n5;i++){

in\_degree+=in\_arr[i];

out\_degree+=out\_arr[i];

}

/\* calculate total time \*/

time5=(ends.tv\_sec-starts.tv\_sec)\*1000+(ends.tv\_nsec-starts.tv\_nsec);

printf("\n\nTOtal In-degree:%6d \t TOtal Out-degree:%6d \t for input n=%d\n",in\_degree,out\_degree,n5 );

printf("Total time :%ld ms\t",time5)

}