BLG372E Analysis of Algorithms

PROJECT 1

Ozan Arkan Can

040090573

Instructor: Zehra Çataltepe

Teaching Assistant: Meryem Uzun-Per

Submission Date: 02.04.2012

CRN: 22534

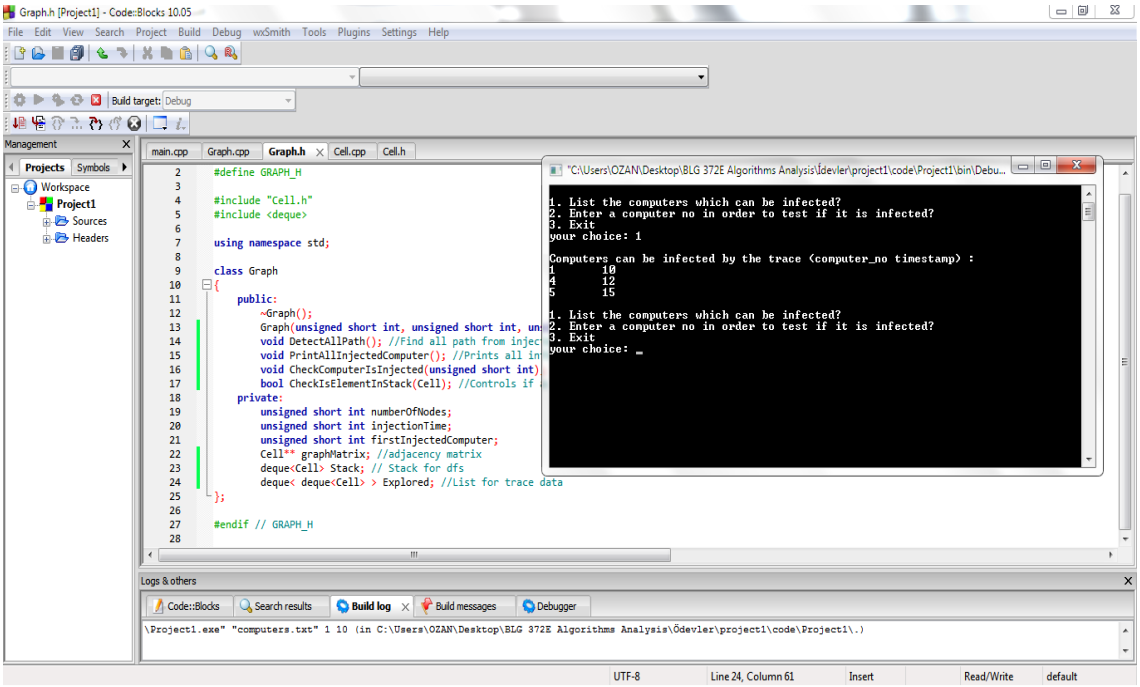
Introduction

In this project, it is aim that designing an algorithm for graph traversal by scenario that given project document. According to the document, there are several computers and computer C_i and C_j communicate at time t_k. One of computers is infected by virus at time t_x and the virus spreads when computers communicate. The purpose of project is finding all infected computers and controlling whether a computer has infected.

Development and Operating Environments

MS Windows

The Code::Blocks IDE has been used to write source code, compile and run the code.



UNIX

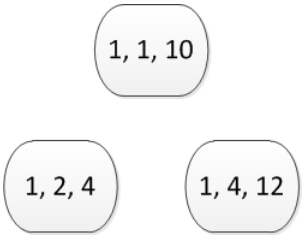
The source code has been also copied to Unix, then compiled and tested with GNU C++ Compiler. The following commands have been used;

To Compile:
▷ g++ main.cpp Graph.h Graph.cpp Cell.h Cell.cpp -o main

To Run:
▷ .\main "computers.txt" 1 10

Node Representation

In this project nodes are represent by Cell class. The node has three data. Incoming node no("source" variable), Destination node no("destination" variable) and communication time ("timestamp" variable). All nodes source and destination values are different except node that for first injected computer. Source and destination values of that node are same. Here some example:



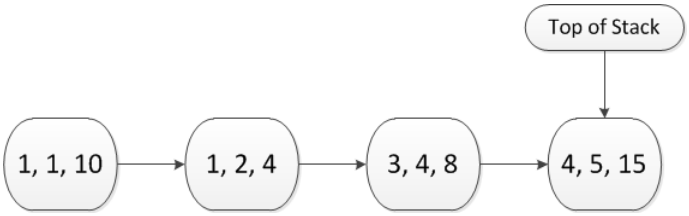
Graph Representation

In this project adjacency matrix is chosen for representing graph. The reason why adjacency matrix is chosen that in depth first search algorithm neighbours of a computer is controlled and it is $O(1)$ time with adjacency matrix. In adjacency matrix row is source, column is destination and if value of matrix[row][source] equals one , it means there is a directed edge between node(row) and node(column). In this project elements of adjacency matrix is Cell which represents node. Matrix is represented as a pointer of pointer to Cell in code and in the constructor gets memory place for that pointer dynamically. After creating graph, adjacency matrix is shown like that:

$$\begin{bmatrix} (1,1,timestamp) & (1,2,timestamp) & (1,3,timestamp) & \dots \\ (2,1,timestamp) & (2,2,timestamp) & (2,3,timestamp) & \dots \\ (3,1,timestamp) & (2,3,timestamp) & (3,3,timestamp) & \dots \\ \vdots & & & \end{bmatrix}$$

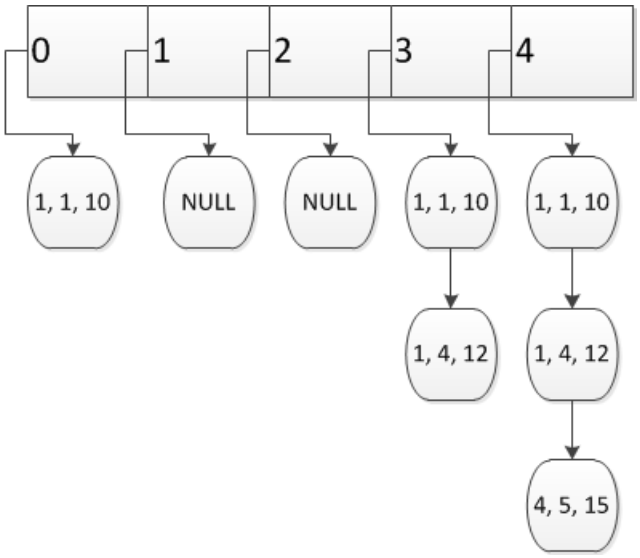
Stack Representation

STL library is used for stack data structure in the code. In STL library, there are severeal data structure and deque is chosen for this project. Almost all of these data structures have push-back(), push-front(), pop-back(), pop-front() methods, but all of them have not operator[ ]. deque structure has it and this property is used several time in graph traversing algorithm. Nodes of stack ara also represented by Cell objects.



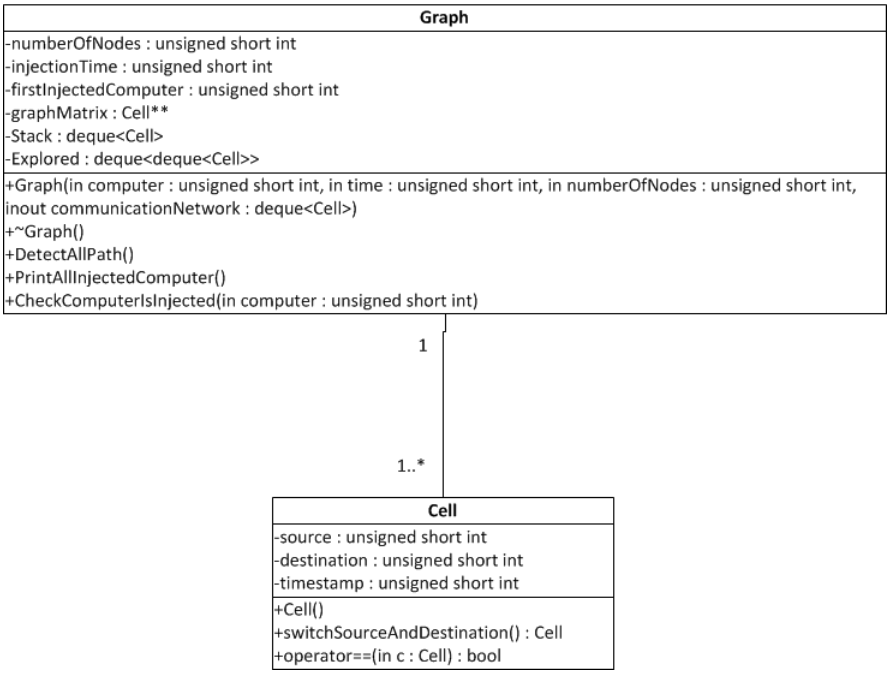
Trace Data Representation

One of project requirements is showing a trace information about infected computer that its number given by user. All infected nodes have trace data and Explored variable is used for hold this data. Explored variable is deque of deque of Cell. That means Explored has a list of Cell for all node. These lists are updated dynamically in dept first search subroutine.



UML Class Diagram

Program UML Class Diagram is like following chart:



Analysis of Algorithm

In this chapter program flow is presented step by step and analyzed all functions. There are some abbreviation;
n : number of nodes
m : number of edges

Getting communication network :

Computers communication information is taken from out source that name of file is given by user. File structure is like that:

source destination timestamp

getCommunicationInfo method is used for this process in main source code. In this function each data is taken part by part. One Cell object is created for each triple (source, destination, timestamp). And, it is added to list if timestamp is greater or equals first injection time. Method also calculates number of nodes by finding node that its number is maximum. Method is processed until end of file. There are m triples so function running time is $O(m)$.

```
deque<Cell> getCommunicationInfo(char* fileName, unsigned short int& numberOfNodes, unsigned short int firstInjectionTime)
{
    ifstream inputFile(fileName);
    Cell* newData;
    deque<Cell> list;

    if(!inputFile.is_open())
        throw "File could not be opened.";

    while(!inputFile.eof())
    {
        newData = new Cell();
        inputFile >> newData->source >> newData->destination >> newData->timestamp;

        if(newData->timestamp >= firstInjectionTime)//Eliminate communication between two computers that they communicate before first injec
            list.push_back(*newData);

        if(newData->source > numberOfNodes)
            numberOfNodes = newData->source;
        if(newData->destination > numberOfNodes)
            numberOfNodes = newData->destination;
    }

    return list;
}
```

Creating graph :

Graph is created in the constructor of Graph Class. This constructor takes number of first injected computer, first injection time, number of nodes and communication info as parameters. Adjacency matrix and list of trace data are initialized in constructor. Adjacency matrix is filled by communication info as directed graph. Number of elements in communication info list is less or equal than number of edges, so running time of constructor is $O(m)$.

```
Graph::Graph(unsigned short int computer, unsigned short int time, unsigned short int numberOfNodes, deque<Cell>& communicationNetwork)
{
    this->numberOfNodes = numberOfNodes; // set number of nodes
    injectionTime = time; // set first injection time
    firstInjectedComputer = computer; // set first infected computer
    Cell current, switched, firstNode;

    //Get memory place for graphMatrix pointer nxn.
    graphMatrix = new Cell*[numberOfNodes];
    for(int i = 0; i < numberOfNodes; i++)
        graphMatrix[i] = new Cell[numberOfNodes];

    //Create list of trace data
    for(int i = 0; i < numberOfNodes; i++)
    {
        deque<Cell> d;
        Explored.push_back(d);
    }

   //Create and add first injected computer to adjacency matrix
   firstNode.destination = firstInjectedComputer;
   firstNode.source = firstInjectedComputer;
   firstNode.timestamp = injectionTime;
   graphMatrix[firstInjectedComputer - 1][firstInjectedComputer - 1] = firstNode;

   //Iterate over network data
   for(unsigned int i = 0; i < communicationNetwork.size(); i++)
   {
       current = communicationNetwork[i];

       //Add an edge both node
       graphMatrix[current.source - 1][current.destination - 1] = current;
       switched = current.switchSourceAndDestination();
       graphMatrix[switched.source - 1][switched.destination - 1] = switched;
   }
}
```

Detecting all injected computers with their trace data

To detect all injected computers and their trace path depth first search algoritgm is used. DFS algorithm modified for project. Here is pseudocode for modified dfs algorithm:

1: Initialize S to be a stack with node that for first injected element
2: Initialize a Cell previous
3: **While** S is not empty
4: Take a node u from S
5: **if** trace data of u is empty
6: **if** previous timestamp does not equal zero
7: Assign trace data of source of u to trace data of destination of u
8: **end if**
9: Add u to end of trace data of u
10: For each node v that is neighbour of u
11: **if** timestamp of v is greater or equals than timestamp of u
12: Add v to the stack
13: **end if**
14: **end if**
15: **else**
16: **if** destination of u is reacheable earlier with ex path
17: Assign recent path as ex path and add u end of path
18: **end if**
19: **end else**
20: **end while**

When we consider given algorithm, it can be seen that while loop at line 3 iterates at most m times, because algorithm traverses all edge. Loop at line 10 iterates at most n - 1 times, because a node can be neighbour of all other nodes, but there is not an edge itself. thus, total running time of finding all infected computers is $O(m * n)$. $m > n - 1 \rightarrow O(m^2)$.

```
void Graph::DetectAllPath()
{
    Cell firstNode, current, previous;

    firstNode.destination = firstInjectedComputer;
    firstNode.source = firstInjectedComputer;
    firstNode.timestamp = injectionTime;

    Stack.push_back(firstNode);

    while(Stack.size() != 0)
    {
        current = Stack.back();
        Stack.pop_back();

        if(Explored[current.destination - 1].size() == 0)
        {
            if(previous.timestamp != 0)//For first node
                Explored[current.destination - 1] = Explored[current.source - 1];
            Explored[current.destination - 1].push_back(current);

            for(int i = 0; i < numberOfNodes; i++)
            {
                if(current.destination - 1 != i && current.source - 1 != i && graphMatrix[current.destination - 1][i].timestamp != 0)//Add neig
                {
                    Cell neighbour = graphMatrix[current.destination - 1][i];
                    Cell search = neighbour.switchSourceAndDestination();

                    if(neighbour.timestamp >= current.timestamp)
                        Stack.push_back(neighbour);
                }
            }
        }
        else
        {
            Cell forComparison = Explored[current.destination - 1][Explored[current.destination - 1].size() - 1];

            if(current.timestamp < forComparison.timestamp)
            {
                Explored[current.destination - 1] = Explored[current.source - 1];
                Explored[current.destination - 1].push_back(current);
            }
        }
        previous = current;
    }
}
```

Determining if a computer is infected and Printing all infected computers :

Program can print all infected computers easily after detected all path that reach an infected computer. Trace data of nodes are in the Explored array, and just looking if trace data list is empty or not to determine whether a computer is infected. Iterating all trace data of nodes, it can be shown user that all infected computers. Running time of determining if a computer is infected is $O(1)$, because program just controls it and running time of printing trace data is $O(m)$. Running time of printing all infected nodes is $O(n)$, because there are at most n nodes that is proper infected computers.

```
void Graph::CheckComputerIsInjected(unsigned short int computer)
{
    if(Explored[computer - 1].size() == 0)
        cout << "No" << endl;
    else
    {
        cout << "Yes" << endl;
        cout << "The trace (computer_no timestamp):" << endl;;
        for(unsigned short int i = 0; i < Explored[computer - 1].size(); i++)
        {
            Cell current = Explored[computer - 1][i];
            cout << current.destination << "\t" << current.timestamp << endl;
        }
    }
}
void Graph::PrintAllInjectedComputer()
{
    cout << "\nComputers can be infected by the trace (computer_no timestamp) :" << endl;
    for(unsigned short int i = 0; i < numberOfNodes; i++)
    {

        if(Explored[i].size() != 0)
        {
            Cell c = Explored[i][Explored[i].size() - 1];
            cout << c.destination << "\t" << c.timestamp << endl;
        }
    }
}
```

Total Running Time of Program :

Program executes DetectAllPath method only one time and it is slowest method. This reason running time of program is $O(m^2)$. If we consider that user can call the PrintAllInjectedComputer and CheckComputerIsInjected method repeatedly, and these methods run in linear time.

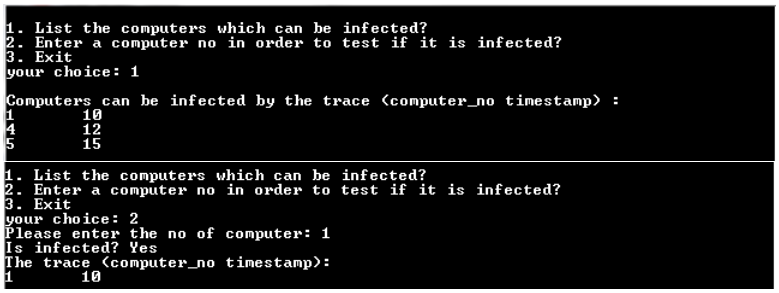
Testing

Program has tested with following two scenario:

Scenario 1:
First infected computer: 1
First infection time: 10
Communication Info:

1	2	4
2	4	8
3	4	8
1	4	12
2	3	14
4	5	15

Screen shots about execution of program:



```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 2
Is infected? No
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 3
Is infected? No
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 4
Is infected? Yes
The trace (computer_no timestamp):
1       10
4       12
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 5
Is infected? Yes
The trace (computer_no timestamp):
1       10
4       12
5       15
```

Scenario 2:
First infected computer: 2
First infection time: 6
Communication Info:

2 5 4
2 3 5
1 3 7
4 2 8
1 7 9
4 1 9
1 6 10
4 6 12
5 4 14
6 5 16

Screen shots about execution of program:

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 1

Computers can be infected by the trace (computer_no timestamp) :
1       9
2       6
4       8
5       14
6       10
7       9
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 1
Is infected? Yes
The trace (computer_no timestamp):
2       6
4       8
1       9
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 2
Is infected? Yes
The trace (computer_no timestamp):
2       6
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 3
Is infected? No
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 4
Is infected? Yes
The trace (computer_no timestamp):
2       6
4       8
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 5
Is infected? Yes
The trace (computer_no timestamp):
2       6
4       8
5       14
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 6
Is infected? Yes
The trace (computer_no timestamp):
2       6
4       8
1       9
6       10
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 2
Please enter the no of computer: 7
Is infected? Yes
The trace (computer_no timestamp):
2       6
4       8
1       9
7       9
```

```
1. List the computers which can be infected?
2. Enter a computer no in order to test if it is infected?
3. Exit
your choice: 3
Program was terminated.
```

CONCLUSION

In this project,

- Graph theory and some greedy graph algorithms are investigated and learned.
- To experience analyzing algorithm.
- To experience devoloping efficent algorithm for a problem.
- To experience modifying known algorithm for a problem.
- It was observed that efficiency of well-chosen data structure on running time of algorithm.