I.T.U.
Faculty of Computer and Informatics
Computer Engineering

İSTANBUL TEKNİK ÜNİVERSİTESİ

1773

Lesson name: Analysis of Algorithms

Lesson Code: BLG 372E

Name Surname: Abdullah AYDEĞER

Number: 040090533

Instructor's Name: Zehra ÇATALTEPE

Assistant's Name: Meryem UZUN

Due Date: 02.04.2012

Table of Contents

My Approach of Problem

My approach of the given problem likes that;

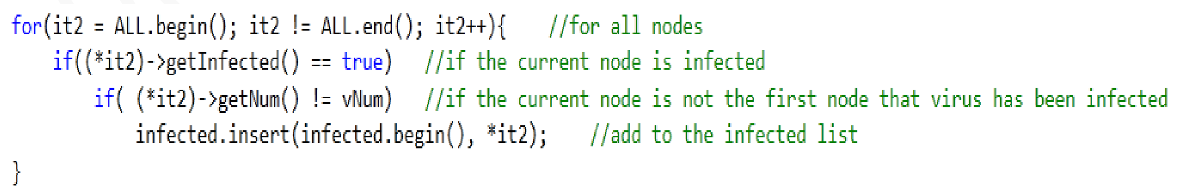
- Firstly, I read input file and creating 2 nodes for each line in the input file.
- Adding all of these creating nodes to the one list (ALL).
- If creating node has been added to ALL list before, then get its smaller times node and make these nodes connected with an directed edge from smaller time to bigger.
- After reading all of the input, then get the first virus infected computer number and time.
- By using the number and time make all of nodes are infected which can be reachable from the first virused node.
- Later asks to user "what do you want?" message on the screen and according to user answer, program can be able to list all of infected viruses or by using the given number determining given numbers' node is infected or not(if infected giving the path).

My Algorithms

Listing all of the Infecteds

My algorithm for listing all of infected computer likes that:

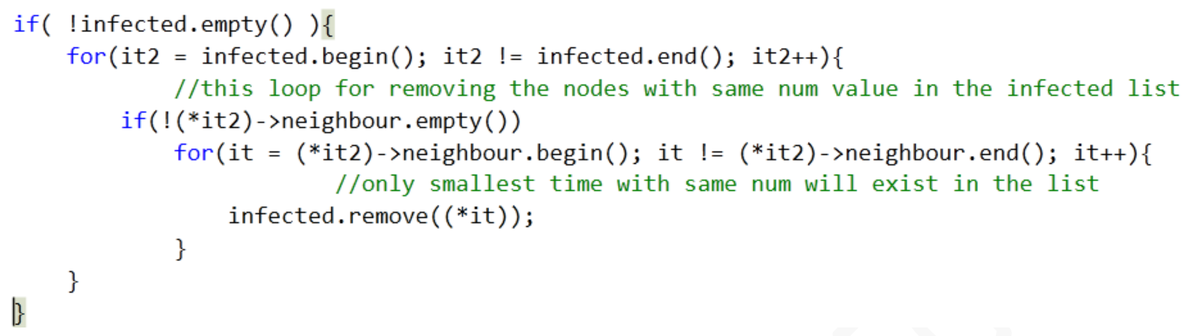
- For all (given program 'ALL') nodes if the computer is infected and that is not the first virused computer, add these nodes to the infecteds list.(as seen in figure 1)



```
for(it2 = ALL.begin(); it2 != ALL.end(); it2++){     //for all nodes
    if((*it2)->getInfected() == true)    //if the current node is infected
        if( (*it2)->getNum() != vNum)    //if the current node is not the first node that virus has been infected
            infected.insert(infected.begin(), *it2);     //add to the infected list
}
```

Figure 1. Adding all infecteds to the list

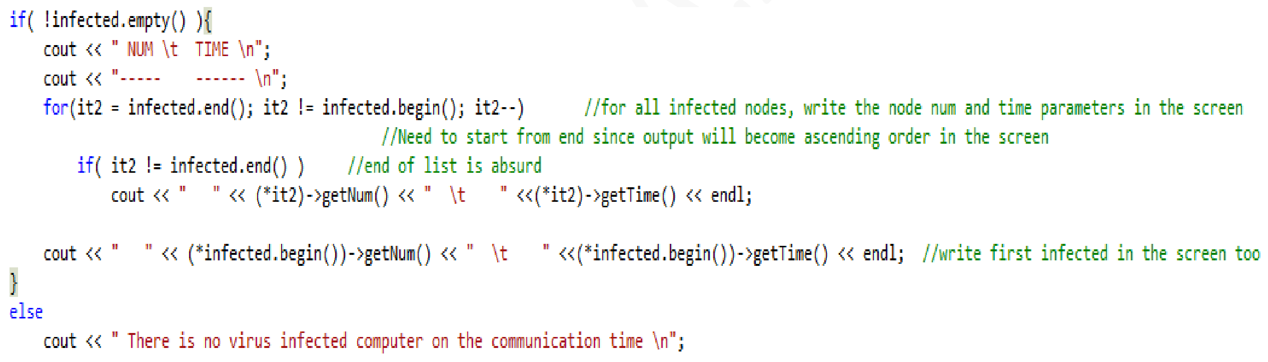
- For all computer in the infected list, if any node number takes place more than once, then delete the bigger number and only the smallest number node will exist more. (as seen in figure 2).



```
if( !infected.empty() ){
    for(it2 = infected.begin(); it2 != infected.end(); it2++){
            //this loop for removing the nodes with same num value in the infected list
        if(!(*it2)->neighbour.empty())
            for(it = (*it2)->neighbour.begin(); it != (*it2)->neighbour.end(); it++){
                        //only smallest time with same num will exist in the list
                infected.remove((*it));
            }
    }
}
```

Figure 2. Removing the same number nodes

- Later if infected list is not empty, then write their number and time on the screen. (as seen in figure3).



```
if( !infected.empty() ){
    cout << " NUM \t  TIME \n";
    cout << "-----    ------ \n";
    for(it2 = infected.end(); it2 != infected.begin(); it2--)       //for all infected nodes, write the node num and time parameters in the screen
                                              //Need to start from end since output will become ascending order in the screen
        if( it2 != infected.end() )     //end of list is absurd
            cout << "    " << (*it2)->getNum() << "  \t    " <<(*it2)->getTime() << endl;

    cout << "    " << (*infected.begin())->getNum() << "  \t    " <<(*infected.begin())->getTime() << endl;  //write first infected in the screen too
}
else
    cout << " There is no virus infected computer on the communication time \n";
```

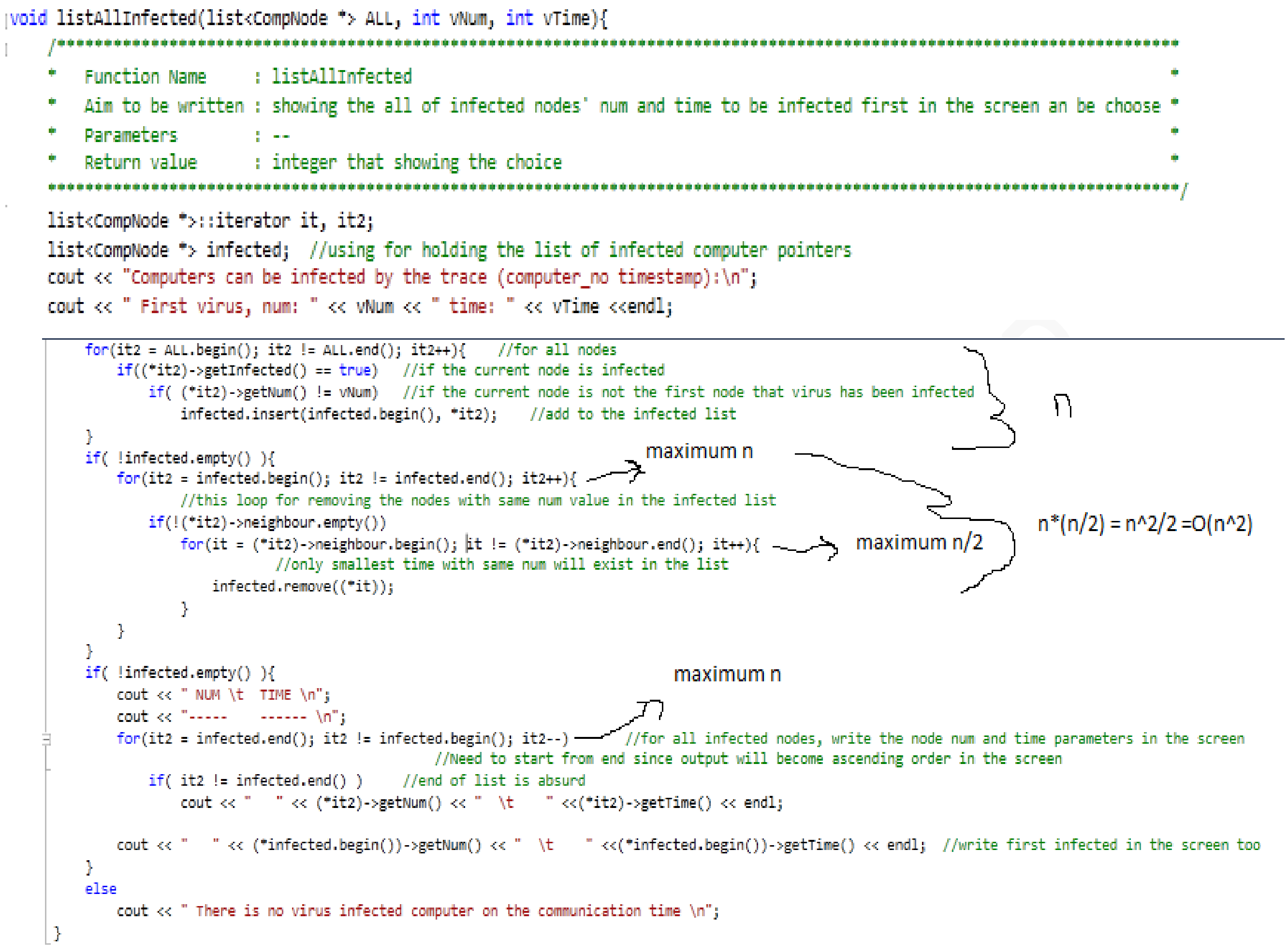
Figure 3. Showing infecteds on the screen

Time Complexity

I calculated the time complexity of listing of all infected viruses function in the photo so that it can be seen easily. The same function can be written with the time complexity in O(n) but I wanted to show nodes with the same number for once. Therefore, I required to spent more time and I got the time complexity of O(n^2).

n = shows the number of the all nodes

m = show the number of all edges



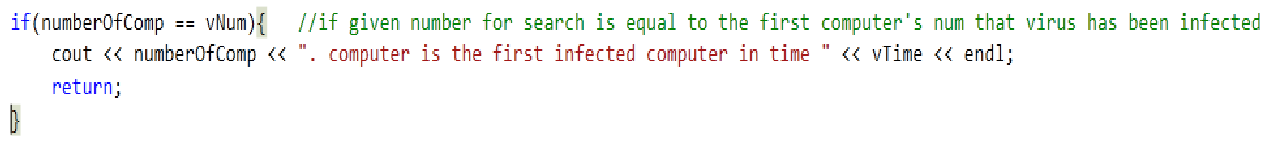
```
void listAllInfected(list<CompNode *> ALL, int vNum, int vTime){
    /*******************************************************************************************************
    *   Function Name     : listAllInfected                                                                *
    *   Aim to be written : showing the all of infected nodes' num and time to be infected first in the screen an be choose *
    *   Parameters        : --                                                                             *
    *   Return value      : integer that showing the choice                                                *
    *******************************************************************************************************/
    list<CompNode *>::iterator it, it2;
    list<CompNode *> infected;  //using for holding the list of infected computer pointers
    cout << "Computers can be infected by the trace (computer_no timestamp):\n";
    cout << " First virus, num: " << vNum << " time: " << vTime <<endl;

    for(it2 = ALL.begin(); it2 != ALL.end(); it2++){    //for all nodes
        if((*it2)->getInfected() == true)   //if the current node is infected
            if( (*it2)->getNum() != vNum)   //if the current node is not the first node that virus has been infected
                infected.insert(infected.begin(), *it2);    //add to the infected list
    }
    if( !infected.empty() ){
        for(it2 = infected.begin(); it2 != infected.end(); it2++){
                //this loop for removing the nodes with same num value in the infected list
            if(!(*it2)->neighbour.empty())
                for(it = (*it2)->neighbour.begin(); it != (*it2)->neighbour.end(); it++){
                        //only smallest time with same num will exist in the list
                    infected.remove((*it));
                }
        }
    }
    if( !infected.empty() ){
        cout << " NUM \t  TIME \n";
        cout << "-----    ------ \n";
        for(it2 = infected.end(); it2 != infected.begin(); it2--)   //for all infected nodes, write the node num and time parameters in the screen
                                    //Need to start from end since output will become ascending order in the screen
            if( it2 != infected.end() )     //end of list is absurd
                cout << "   " << (*it2)->getNum() << "  \t    " <<(*it2)->getTime() << endl;

        cout << "   " << (*infected.begin())->getNum() << "  \t    " <<(*infected.begin())->getTime() << endl;  //write first infected in the screen too
    }
    else
        cout << " There is no virus infected computer on the communication time \n";
}
```

Figure 4. Algorithm of listing all infecteds

Determining given computer has been infected or not

I follow some strategies for determining the node has been infected or not, and if it has been infected showing the path of the virus from the first computer to the searching node. I can explain my strategy as follow:

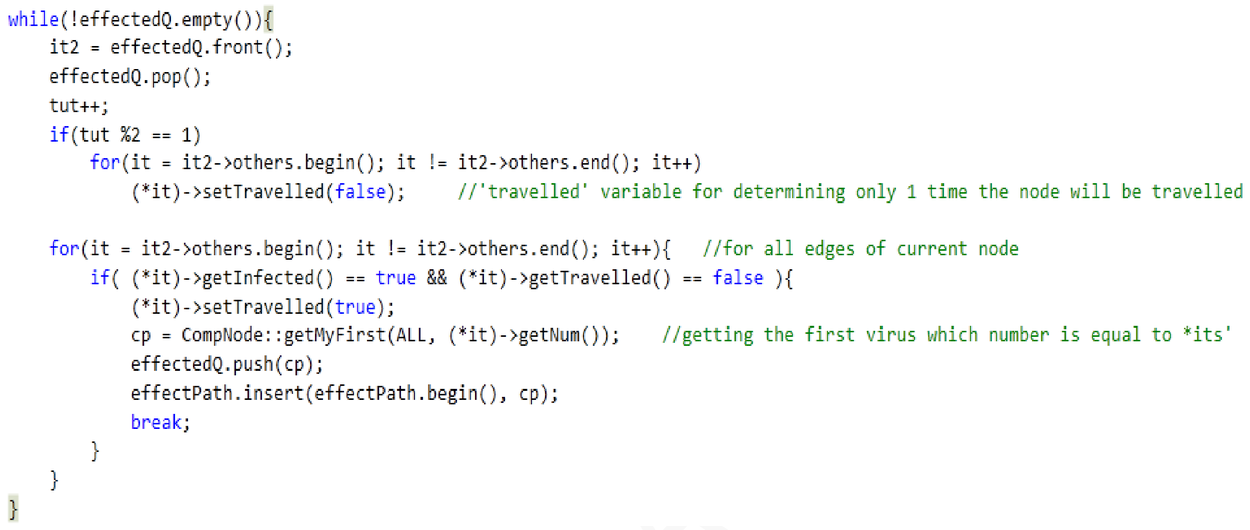
- Firstly, control the given number is equal to the first infected computer number, and if they are equal then show the number and time on the screen and return the main menu(as seen in figure5).



```
if(numberOfComp == vNum){   //if given number for search is equal to the first computer's num that virus has been infected
    cout << numberOfComp << ". computer is the first infected computer in time " << vTime << endl;
    return;
}
```

Figure 5. First virus found as infected

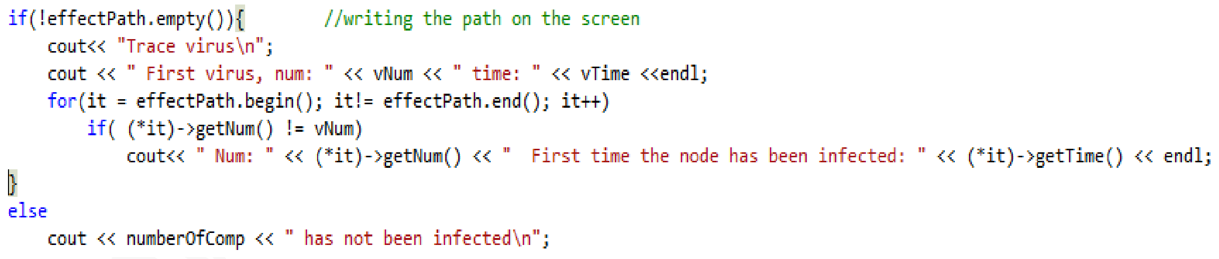
- If searching node is not the first infected computer, then try to find the path from the searching node to the first infected computer by using queue for this (as seen in figure 6). All used virus path inserting to the list (in program list name is 'effectPath').



```
while(!effectedQ.empty()){
    it2 = effectedQ.front();
    effectedQ.pop();
    tut++;
    if(tut %2 == 1)
        for(it = it2->others.begin(); it != it2->others.end(); it++)
            (*it)->setTravelled(false);     //'travelled' variable for determining only 1 time the node will be travelled

    for(it = it2->others.begin(); it != it2->others.end(); it++){   //for all edges of current node
        if( (*it)->getInfected() == true && (*it)->getTravelled() == false ){
            (*it)->setTravelled(true);
            cp = CompNode::getMyFirst(ALL, (*it)->getNum());    //getting the first virus which number is equal to *its'
            effectedQ.push(cp);
            effectPath.insert(effectPath.begin(), cp);
            break;
        }
    }
}
```

Figure 6. Finding the path of virus

- Later all nodes in the path list are showing on the screen (as seen in figure 6).

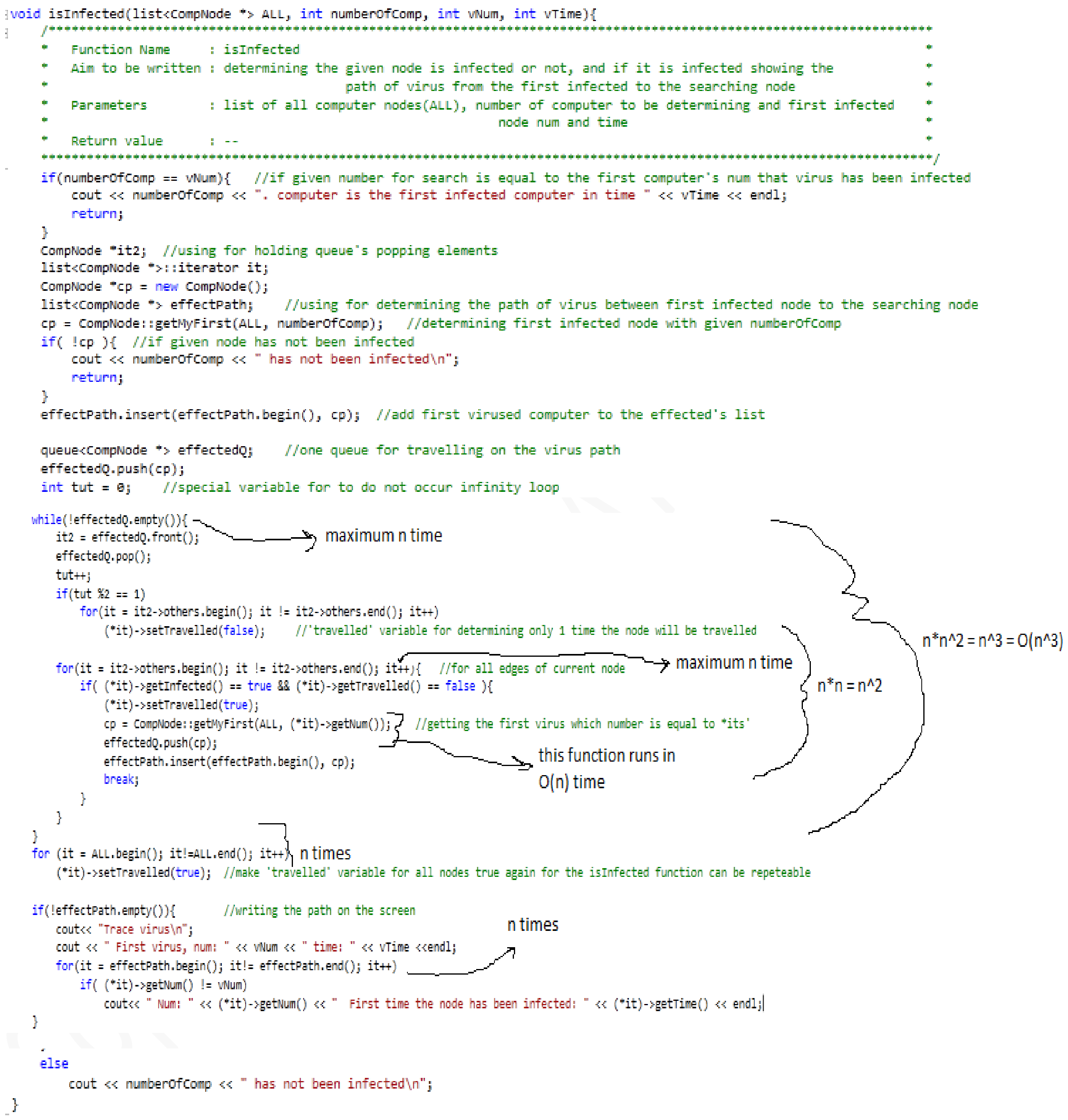


```
if(!effectPath.empty()){        //writing the path on the screen
    cout<< "Trace virus\n";
    cout << " First virus, num: " << vNum << " time: " << vTime <<endl;
    for(it = effectPath.begin(); it!= effectPath.end(); it++)
        if( (*it)->getNum() != vNum)
            cout<< " Num: " << (*it)->getNum() << "  First time the node has been infected: " << (*it)->getTime() << endl;
}
else
    cout << numberOfComp << " has not been infected\n";
```

Figure 7. Showing the path on the screen

Time Complexity

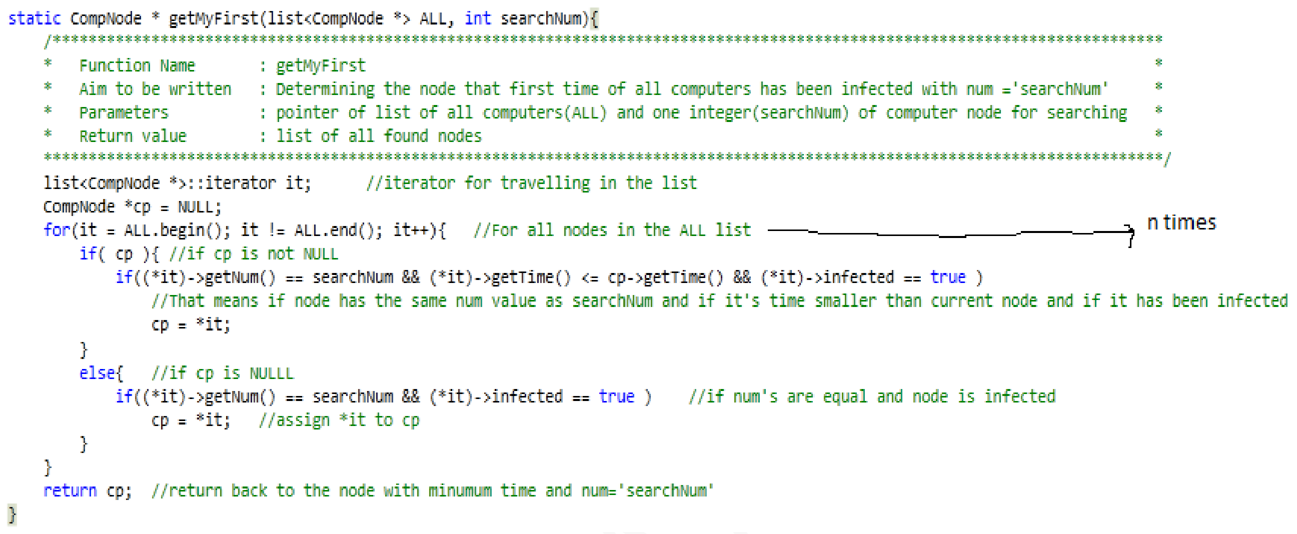
I have firstly one while loop for the queue is not empty. My queue can be maximum n size, therefore while loop runs for 'n' times. In addition one node can be maximum n/2 edge which means 'n' again for complexity calculation. In the for loop of edges, I determine the first node by which number is equal to the given number by using the function that name is getMyFirst. This function runs for 'n' time as seen in the figure 9.



```
void isInfected(list<CompNode *> ALL, int numberOfComp, int vNum, int vTime){
    /*****************************************************************************************************************
    *   Function Name     : isInfected                                                                              *
    *   Aim to be written : determining the given node is infected or not, and if it is infected showing the       *
    *                                         path of virus from the first infected to the searching node           *
    *   Parameters        : list of all computer nodes(ALL), number of computer to be determining and first infected *
    *                                                             node num and time                                 *
    *   Return value      : --                                                                                      *
    *****************************************************************************************************************/
    if(numberOfComp == vNum){   //if given number for search is equal to the first computer's num that virus has been infected
        cout << numberOfComp << ". computer is the first infected computer in time " << vTime << endl;
        return;
    }
    CompNode *it2;  //using for holding queue's popping elements
    list<CompNode *>::iterator it;
    CompNode *cp = new CompNode();
    list<CompNode *> effectPath;    //using for determining the path of virus between first infected node to the searching node
    cp = CompNode::getMyFirst(ALL, numberOfComp);   //determining first infected node with given numberOfComp
    if( !cp ){  //if given node has not been infected
        cout << numberOfComp << " has not been infected\n";
        return;
    }
    effectPath.insert(effectPath.begin(), cp);  //add first virused computer to the effected's list

    queue<CompNode *> effectedQ;    //one queue for travelling on the virus path
    effectedQ.push(cp);
    int tut = 0;    //special variable for to do not occur infinity loop

    while(!effectedQ.empty()){
        it2 = effectedQ.front();
        effectedQ.pop();
        tut++;
        if(tut %2 == 1)
            for(it = it2->others.begin(); it != it2->others.end(); it++)
                (*it)->setTravelled(false);    //'travelled' variable for determining only 1 time the node will be travelled

        for(it = it2->others.begin(); it != it2->others.end(); it++){   //for all edges of current node
            if( (*it)->getInfected() == true && (*it)->getTravelled() == false ){
                (*it)->setTravelled(true);
                cp = CompNode::getMyFirst(ALL, (*it)->getNum());   //getting the first virus which number is equal to *its'
                effectedQ.push(cp);
                effectPath.insert(effectPath.begin(), cp);
                break;
            }
        }
    }
    for (it = ALL.begin(); it!=ALL.end(); it++)
        (*it)->setTravelled(true);  //make 'travelled' variable for all nodes true again for the isInfected function can be repeteable

    if(!effectPath.empty()){        //writing the path on the screen
        cout<< "Trace virus\n";
        cout << " First virus, num: " << vNum << " time: " << vTime <<endl;
        for(it = effectPath.begin(); it!= effectPath.end(); it++)
            if( (*it)->getNum() != vNum)
                cout<< " Num: " << (*it)->getNum() << "  First time the node has been infected: " << (*it)->getTime() << endl;
    }

    else
        cout << numberOfComp << " has not been infected\n";
}
```

Figure 8. Algorithm of isInfected

My function for getting the first node which has the number as parameter of this function:



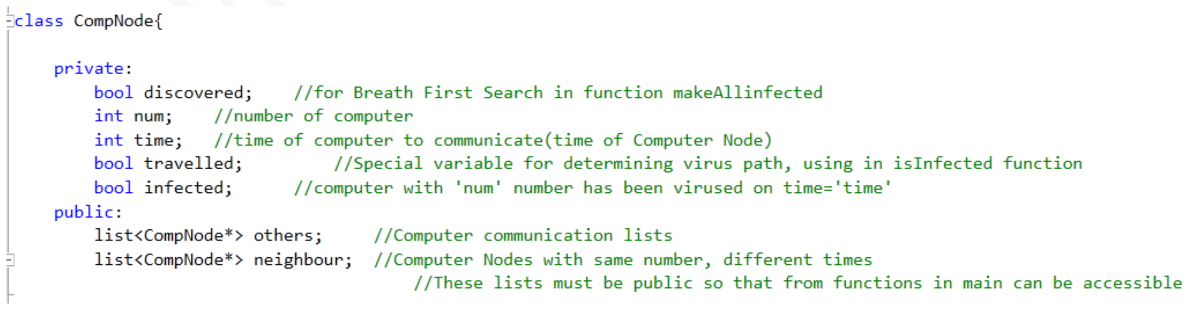
```
static CompNode * getMyFirst(list<CompNode *> ALL, int searchNum){
    /****************************************************************************************************************
    *   Function Name       : getMyFirst                                                                            *
    *   Aim to be written   : Determining the node that first time of all computers has been infected with num ='searchNum'    *
    *   Parameters          : pointer of list of all computers(ALL) and one integer(searchNum) of computer node for searching  *
    *   Return value        : list of all found nodes                                                               *
    ****************************************************************************************************************/
    list<CompNode *>::iterator it;      //iterator for travelling in the list
    CompNode *cp = NULL;
    for(it = ALL.begin(); it != ALL.end(); it++){   //For all nodes in the ALL list
        if( cp ){ //if cp is not NULL
            if((*it)->getNum() == searchNum && (*it)->getTime() <= cp->getTime() && (*it)->infected == true )
                //That means if node has the same num value as searchNum and if it's time smaller than current node and if it has been infected
                cp = *it;
        }
        else{   //if cp is NULLL
            if((*it)->getNum() == searchNum && (*it)->infected == true )    //if num's are equal and node is infected
                cp = *it;   //assign *it to cp
        }
    }
    return cp;  //return back to the node with minumum time and num='searchNum'
}
```

Figure 9. getMyFirst function

Data Structures

Class parameters

I have used one class for doing all necessary acts, which name is ‘CompNode’. My class parameters are seen in figure 10 with necessary explanations.



```
class CompNode{

    private:
        bool discovered;     //for Breath First Search in function makeAllinfected
        int num;     //number of computer
        int time;    //time of computer to communicate(time of Computer Node)
        bool travelled;          //Special variable for determining virus path, using in isInfected function
        bool infected;       //computer with 'num' number has been virused on time='time'
    public:
        list<CompNode*> others;      //Computer communication lists
        list<CompNode*> neighbour;   //Computer Nodes with same number, different times
                                         //These lists must be public so that from functions in main can be accessible
```

Figure 10. Class parameters

My class methods are seen in next figures with some explanations.

Methods:

```
int getNum();   //get method for 'num'
int getTime();  //get method for 'time'
bool getInfected(); //get method for 'infected'
bool getTravelled();    //get method for 'travelled'
void setTravelled(bool );   //set method for 'travelled'

CompNode();     //default constructor
CompNode(int, int); //constructor with number and time parameters
```

Static Functions

addToALL

```
static void addToALL(list<CompNode*> *ALL, CompNode *cp){
    /************************************************************************************
    *   Function Name       : addToALL                                                  *
    *   Aim to be written   : Adding given computer node to the list of all nodes       *
    *   Parameters          : pointer of list of all computers(ALL) and one pointer of a computer node  *
    *   Return value        : --                                                        *
    ************************************************************************************/
```

isExist

```
static list<CompNode *> isExist(list<CompNode*> ALL, int searchNum){
    /************************************************************************************
    *   Function Name       : isExist                                                   *
    *   Aim to be written   : Determining the given num(searchNum) has to be added to the list of all nodes before or not   *
    *   Parameters          : pointer of list of all computers(ALL) and one integer(searchNum) of computer node for searching  *
    *   Return value        : pointer list of all found nodes                           *
    ************************************************************************************/
```

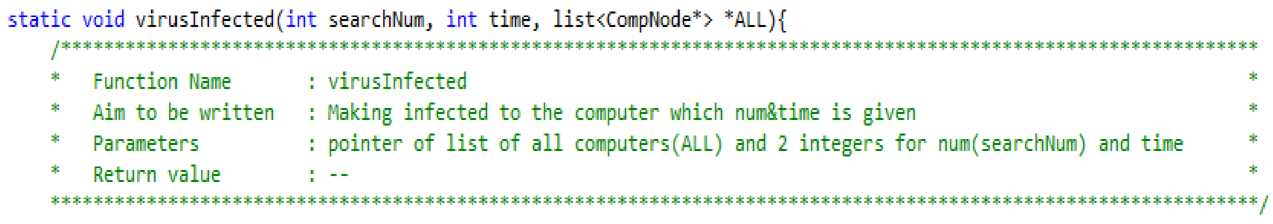
addNeighbour

```
static void addNeighbour(list<CompNode *> cp1, CompNode *cp2){
    /************************************************************************************
    *   Function Name       : addNeighbour                                              *
    *   Aim to be written   : Adding an edge between given node and each of the given lists' nodes  *
    *   Parameters          : 1 pointer list of computer nodes and one pointer of computer node     *
    *   Return value        : --                                                        *
    ************************************************************************************/
```

makeTheyConnected

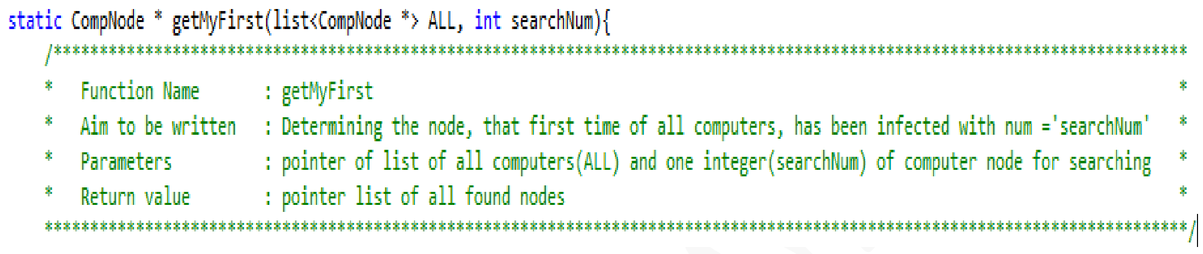
```
static void makeTheyConnected(CompNode *cp1, CompNode *cp2){
    /****************************************************************************
    *   Function Name       : makeTheyConnected                                 *
    *   Aim to be written   : Making an edge between given parameter nodes      *
    *   Parameters          : 2 pointers of computer nodes(cp1, cp2)            *
    *   Return value        : --                                                *
    ****************************************************************************/
```

virusInfected



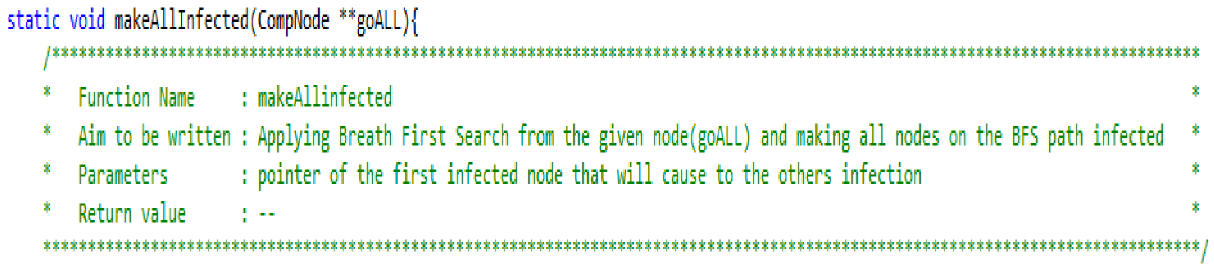
```
static void virusInfected(int searchNum, int time, list<CompNode*> *ALL){
    /*************************************************************************************************
    *   Function Name       : virusInfected                                                            *
    *   Aim to be written   : Making infected to the computer which num&time is given                  *
    *   Parameters          : pointer of list of all computers(ALL) and 2 integers for num(searchNum) and time    *
    *   Return value        : --                                                                       *
    *************************************************************************************************/
```

getMyFirst



```
static CompNode * getMyFirst(list<CompNode *> ALL, int searchNum){
    /*************************************************************************************************
    *   Function Name       : getMyFirst                                                               *
    *   Aim to be written   : Determining the node, that first time of all computers, has been infected with num ='searchNum'   *
    *   Parameters          : pointer of list of all computers(ALL) and one integer(searchNum) of computer node for searching   *
    *   Return value        : pointer list of all found nodes                                          *
    *************************************************************************************************/
```

makeAllInfected



```
static void makeAllInfected(CompNode **goALL){
    /*************************************************************************************************
    *   Function Name     : makeAllinfected                                                            *
    *   Aim to be written : Applying Breath First Search from the given node(goALL) and making all nodes on the BFS path infected   *
    *   Parameters        : pointer of the first infected node that will cause to the others infection  *
    *   Return value      : --                                                                         *
    *************************************************************************************************/
```

Compilation and Running

I have compiled, without any error, my code in Win7 and Linux operating systems. For Linux environment I have tried to compile my code not only by connecting the ITU server by using SSH, but also opening the virtual machine for Linux. My compilation code for Linux:

```
g++ main.cpp –o main
```

Running command:

```
./main input_file_name first_infected_computer_no first_virused_time
```