THE NATIONAL HIGHER SCHOOL OF ARTIFICIAL INTELLIGENCE

Report

Subject : Data Structures and Algorithms mini project about E-MAIL SYSTEM

Team Leader :

- Lalagui Baraa Fatima Zohra

Team Members :

- Lalagui, Baraa Fatima Zohra (Student) <baraa.lalagui@ensia.edu.dz>;
- Boukedjar, Hafida <hafida.boukedjar@ensia.edu.dz>;
- Meriem (Student) Djidjeli <meriem.djidjeli@ensia.edu.dz>;

<u>Explanation of the problem :</u>

In this "Email System" challenge, our purpose is to create a "Mail Server" system that manages electronic mail messages (e-mail). The system can be used by a large number of users, who are recognized by their names and individual User IDs.

Users that are signed up in the same mail server can exchange emails , The processing of these emails passes through the following two (2) phases:

<u>Phase 1</u>: Message Sending Phase

When a user from this server (a “sender”) wishes to mail another user

(a “receiver”) :

- This message is added to the mailserver's queue .
- Then sent to the recipient's mailbox
    - if the message does not reach the intended recipient, that is, the recipient is not found in that mailServer's "Users":
        - The mail server tries to resend it by pushing into the mails queue and processing it again for three times when there are no more new messages to be sent
            - If still not sent the sender will be notified (a message will be sent to the sender by the mailServerDeliverySystem to inform him that his message could not be sent to the specified receiver) .

<u>Phase 2</u>: Message Reception Phase

After receiving a message (an email is pushed into the mailserver’s mails queue)

- The mail-server transmits it to the receiver’s mailbox by pushing-front the email into it , ie: the mail-server uses the user’s mailboxes as a stack .
- When the receiver wants to read his mailbox :
    - The emails are popped-front one by one from the mailbox (most recent to the latest) and whenever one of them is read (labeled as seen), it is pushed-back into the mailbox ,ie: the user uses its mailbox as a queue.
- When the receiver wants to read only his most recent email
    - The mailbox pops only the email that is not seen

We adopted the concept of Divide and Conquer , therefore we created :

- We made a class message that holds the message the user wants to send it has as data members :
    - Sender : that is a user _ the user that sends the message_
    - Receiver : that is a user _the user that receives the message_
    - A string text that has in it the text of the message to be sent
    - A boolean variable seen that tells if a message is read by the receiver or yet.
- For the representation of the User we made:
    - Class UserInfo :
        - Containing the personal User info “ Name , ID”.
    - A list of messages that is the user’s MailBox that is managed as a queue according to the user and as a stack for the mailserver
- To represent the MailServer
    - ADT of type (AVL tree , BS tree, Hash Table) data member of users to represent all users in the mail server “Users”.
    - A Queue of messages that is “Mails” that stores all the emails sent by the users in the mail server.
    - And another data member that is the current size which is the number of elements inserted in the queue.
    - The utility of mail server is to manage messages delivery.
- To manage the mail server we made a class Space that has
    - Mail server as a data member
    - A user that is the mailserver Delivery System that notifies users

<u>The methods :</u>

Class message :

- ReadMessage function that writes the message on the receivers' screen to give so they read it and then mark it as seen.

Class User :

- Pushing method : to push a new message into the user's mailbox (using the mailbox as a queue)
- readMail method : to read the unseen emails (checking if they're seen using the checkMail method) from the user's mailbox by popping the most recent one reading it , marking it as seen then then pushing it back into the mailbox so the user can still have access to them in his mailbox (the ability to read them with the reading method)

Class MailServer :

- putInServer method that takes a message as an argument and put (enqueue) that message into the mails queue

Class Space :

- WriteMessage method that takes as an argument a user that is the sender : it gives the sender the ability to write a message to another user (he/she would specify)
- SignUp : adds a new User to the mailserver by inserting it in the User ADT (AVL tree , BS tree, Hash Table)
- SignIn : to open a user's account
- MakeEmpty : pop the messages from mailserver's mails queue to push them into the ideal receiver's mailbox after checking the existence of the receiver in the mail-server by calling the send method that sends only the message that is in the front to its receiver

<u>The purpose of each ADT :</u>

The use of a Queue :

- The queue is used to store the messages of the users in the mailserver for its property because we need to process the sending of the messages according to the order of their arrival where the first message to come is the first one to be sent to its receiver.
- The queue property was also used in the mailbox when reading messages for the user and accessing them.

The use of the list for the implementation of MailBox in User:

- If the User receives email, he/she must have access to the latest emails received , and that using the STACK properties such that :
- After reading a given message that message is labeled as seen , with keeping the opportunity of the User to access it even after it's been read , and that using the QUEUE properties.

The use of stack :

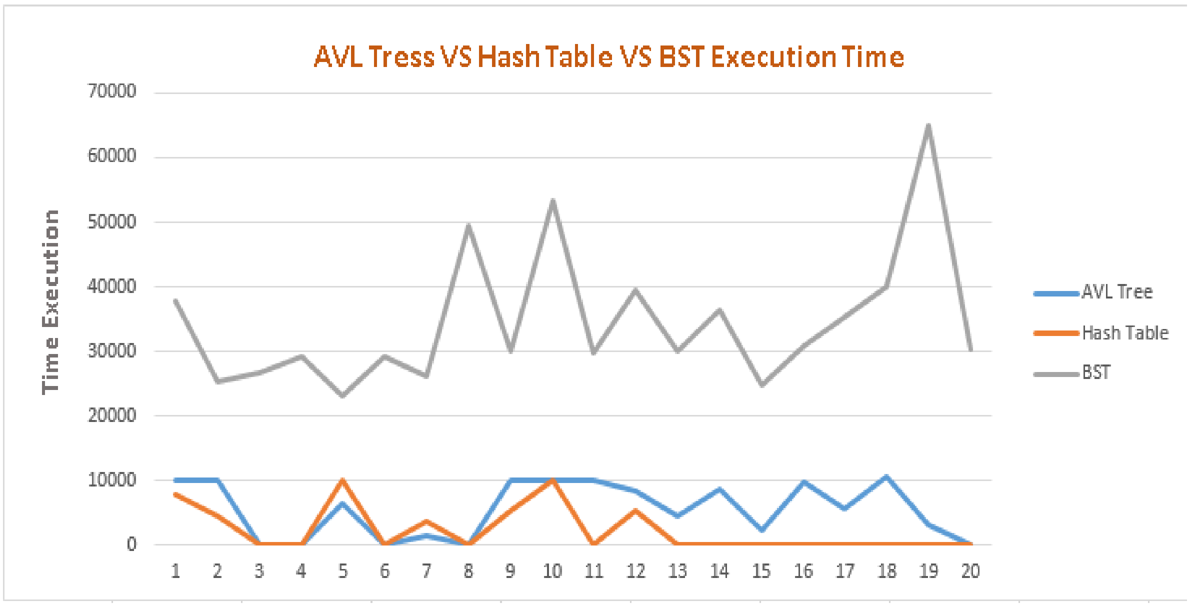
- Receiving a message means the email server push-front the message into the receiver's mail-box.
- Reading a message will be processed from recent to latest and that using the STACK property of pop-ing the top 'recent' .

<u>Average time Interpretation:</u>

Making use of the function getAverageSendingTime , we could trace the table below :

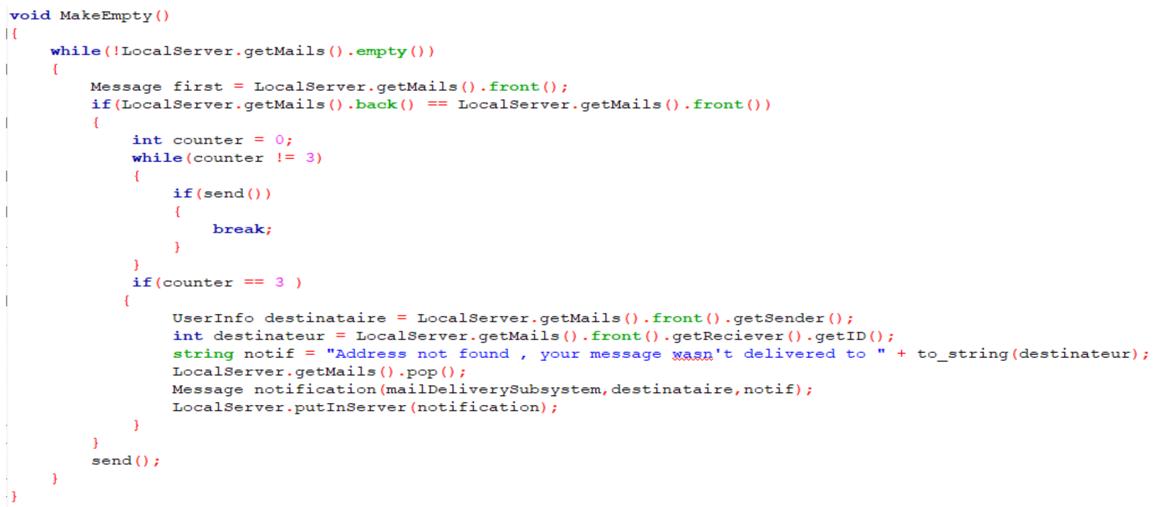
- From the analyses of the given results .
    - The hash-table gives : 2159.205.
    - The AVL tree : 5536.35.
    - The BSTree : 32596.6.
- The winning ADT performing in the given problem of users classification is the hash table .

	AVL Tree	Hash Table	BST
Time execution	9963	7847	37729
	9990	4361	25379
	0	0	26803
	0	0	29117
	6394	9972	23097
	0	0	29128
	1536	3601	25983
	0	0	49535
	9984	5334	29918
	9971	9985	53242
	9985	0	29819
	8377	5325	39580
	4465	0	29945
	8672	0	36442
	2359	0	24811
	9757	0	30921
	5680	0	35274
	10569	0	39893
	3025	0	65007
	0	0	30401

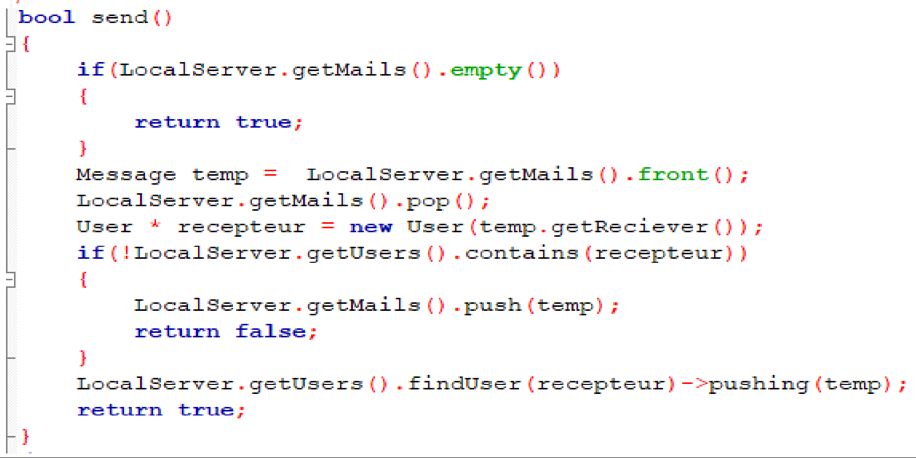


Problems occur in the testing phase:

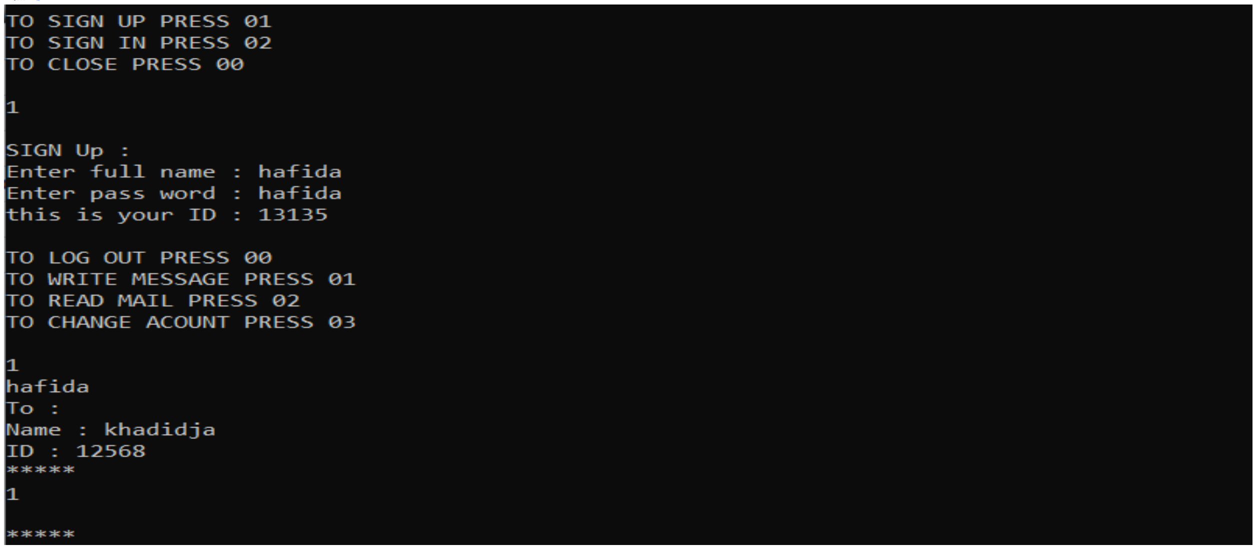
- ***1/- Problem* :** In the space class if message not send it's handled , as explained above "being resent if no new messages to be send" , here occurs the exception which breaks the system "infinite loop" as shown , ie: if the system has more than one message to be resent :



```
void MakeEmpty()
{
    while(!LocalServer.getMails().empty())
    {
        Message first = LocalServer.getMails().front();
        if(LocalServer.getMails().back() == LocalServer.getMails().front())
        {
            int counter = 0;
            while(counter != 3)
            {
                if(send())
                {
                    break;
                }
            }
            if(counter == 3 )
           {
                UserInfo destinataire = LocalServer.getMails().front().getSender();
                int destinateur = LocalServer.getMails().front().getReciever().getID();
                string notif = "Address not found , your message wasn't delivered to " + to_string(destinateur);
                LocalServer.getMails().pop();
                Message notification(mailDeliverySubsystem,destinataire,notif);
                LocalServer.putInServer(notification);
            }
        }
        send();
    }
}
```

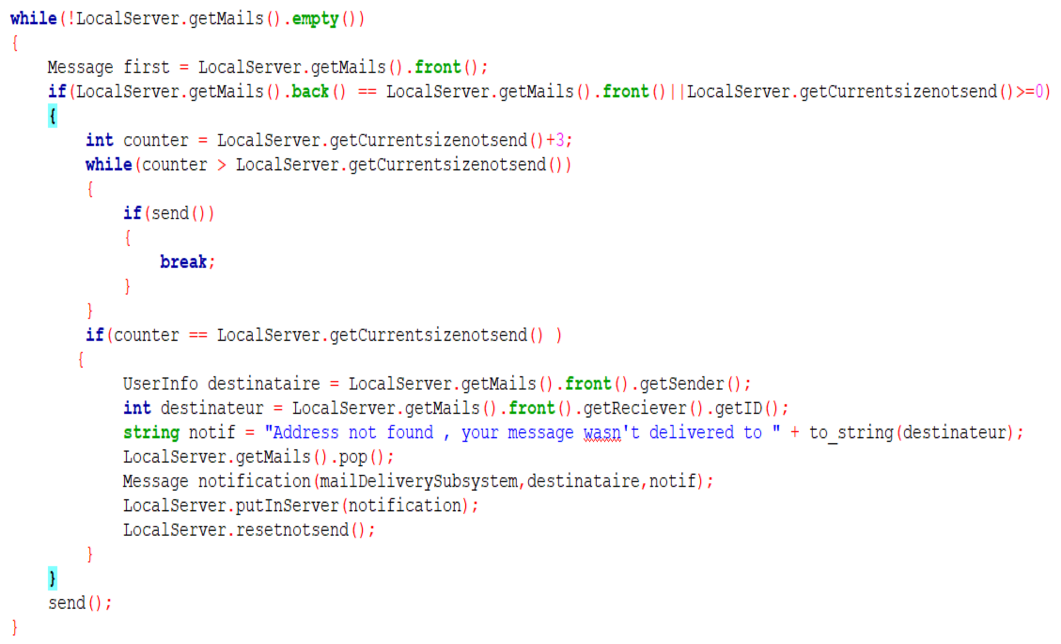


```
bool send()
{
    if(LocalServer.getMails().empty())
    {
        return true;
    }
    Message temp =  LocalServer.getMails().front();
    LocalServer.getMails().pop();
    User * recepteur = new User(temp.getReciever());
    if(!LocalServer.getUsers().contains(recepteur))
    {
        LocalServer.getMails().push(temp);
        return false;
    }
    LocalServer.getUsers().findUser(recepteur)->pushing(temp);
    return true;
}
```

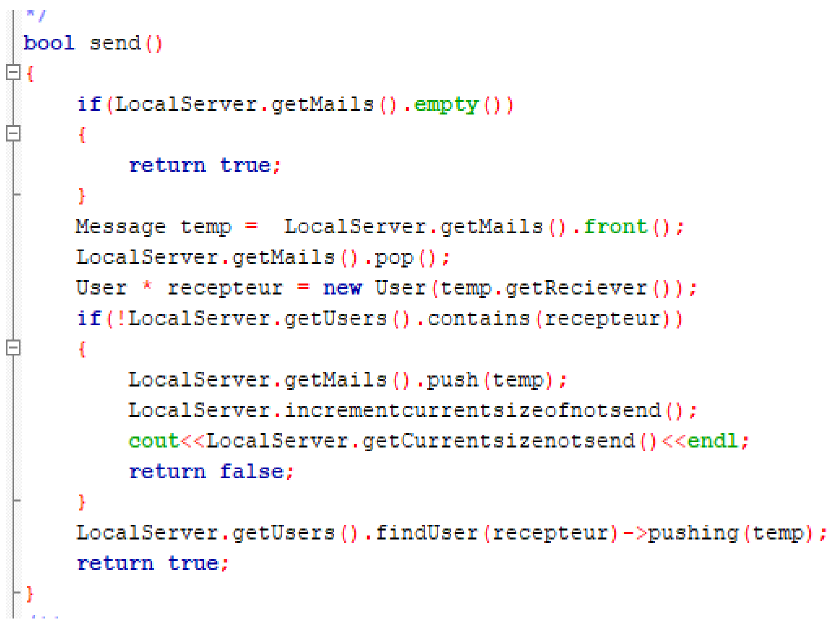


```
TO SIGN UP PRESS 01
TO SIGN IN PRESS 02
TO CLOSE PRESS 00

1

SIGN Up :
Enter full name : hafida
Enter pass word : hafida
this is your ID : 13135

TO LOG OUT PRESS 00
TO WRITE MESSAGE PRESS 01
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03

1
hafida
To :
Name : khadidja
ID : 12568
*****
1

*****
```

- ***1/- solution :***
    - Add a variable type that keeps track of the number that messages have not been sent effectively , if the previous number has been incremented three times that message is destroyed .

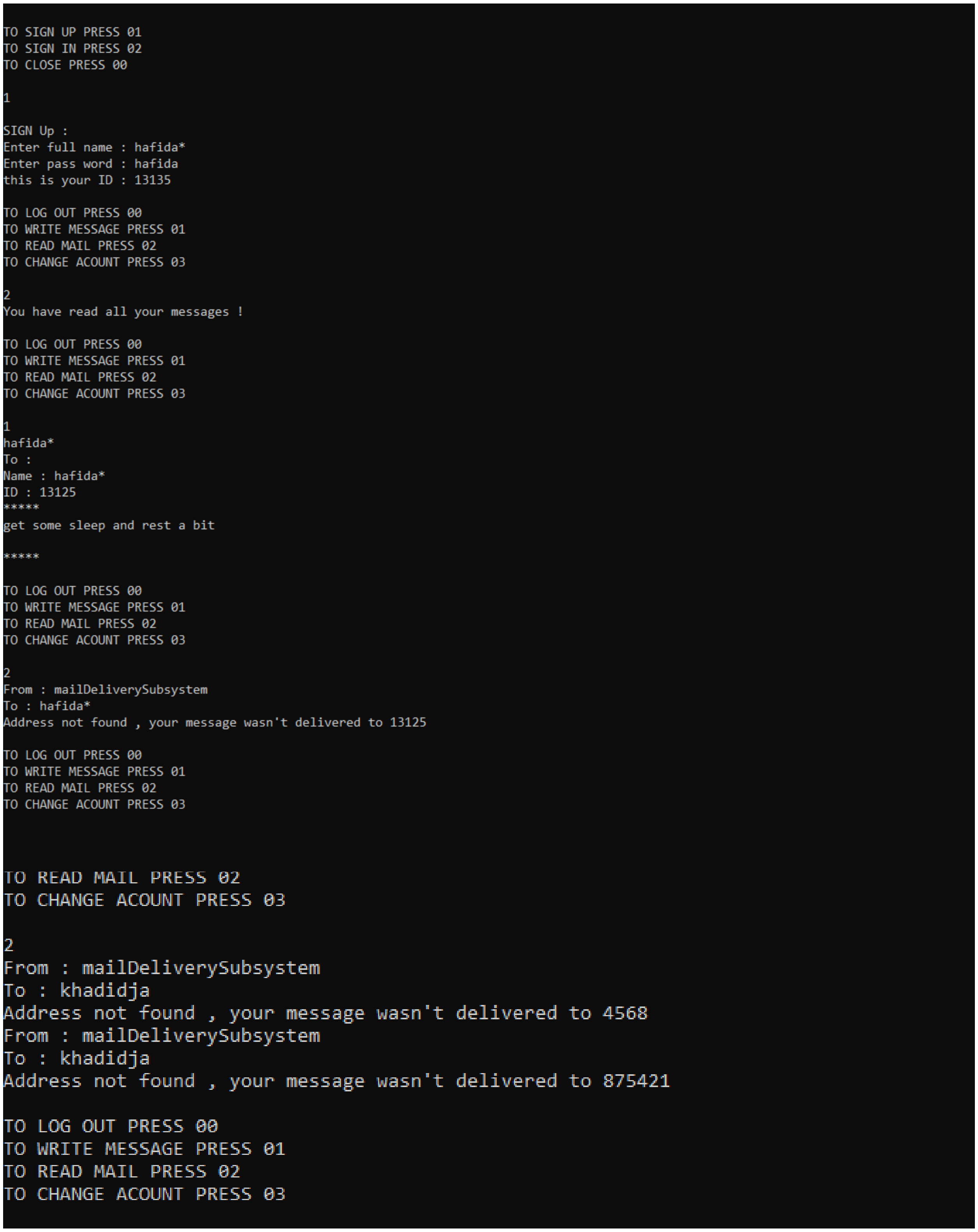


```
while(!LocalServer.getMails().empty())
{
    Message first = LocalServer.getMails().front();
    if(LocalServer.getMails().back() == LocalServer.getMails().front()||LocalServer.getCurrentsizenotsend()>=0)
    {
        int counter = LocalServer.getCurrentsizenotsend()+3;
        while(counter > LocalServer.getCurrentsizenotsend())
        {
            if(send())
            {
                break;
            }
        }
        if(counter == LocalServer.getCurrentsizenotsend() )
       {
            UserInfo destinataire = LocalServer.getMails().front().getSender();
            int destinateur = LocalServer.getMails().front().getReciever().getID();
            string notif = "Address not found , your message wasn't delivered to " + to_string(destinateur);
            LocalServer.getMails().pop();
            Message notification(mailDeliverySubsystem,destinataire,notif);
            LocalServer.putInServer(notification);
            LocalServer.resetnotsend();
        }
    }
    send();
}
```



```
*/
bool send()
{
    if(LocalServer.getMails().empty())
    {
        return true;
    }
    Message temp =  LocalServer.getMails().front();
    LocalServer.getMails().pop();
    User * recepteur = new User(temp.getReciever());
    if(!LocalServer.getUsers().contains(recepteur))
    {
        LocalServer.getMails().push(temp);
        LocalServer.incrementcurrentsizeofnotsend();
        cout<<LocalServer.getCurrentsizenotsend()<<endl;
        return false;
    }
    LocalServer.getUsers().findUser(recepteur)->pushing(temp);
    return true;
}
```

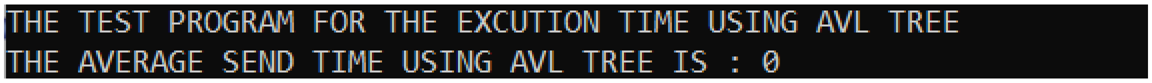
- ***1/- Result :***



```
TO SIGN UP PRESS 01
TO SIGN IN PRESS 02
TO CLOSE PRESS 00

1

SIGN Up :
Enter full name : hafida*
Enter pass word : hafida
this is your ID : 13135

TO LOG OUT PRESS 00
TO WRITE MESSAGE PRESS 01
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03

2
You have read all your messages !

TO LOG OUT PRESS 00
TO WRITE MESSAGE PRESS 01
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03

1
hafida*
To :
Name : hafida*
ID : 13125
*****
get some sleep and rest a bit

*****

TO LOG OUT PRESS 00
TO WRITE MESSAGE PRESS 01
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03

2
From : mailDeliverySubsystem
To : hafida*
Address not found , your message wasn't delivered to 13125

TO LOG OUT PRESS 00
TO WRITE MESSAGE PRESS 01
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03
```

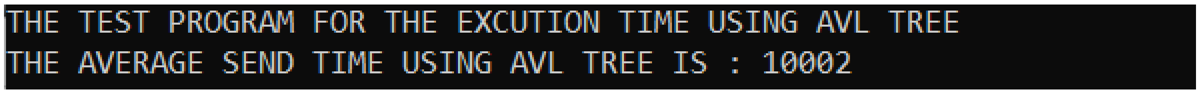
```
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03

2
From : mailDeliverySubsystem
To : khadidja
Address not found , your message wasn't delivered to 4568
From : mailDeliverySubsystem
To : khadidja
Address not found , your message wasn't delivered to 875421

TO LOG OUT PRESS 00
TO WRITE MESSAGE PRESS 01
TO READ MAIL PRESS 02
TO CHANGE ACOUNT PRESS 03
```

- ***2/- Problem*** **:** In counting the average time for sending a message to a receiver , some ADT returns 0 , “using the microseconds count” .



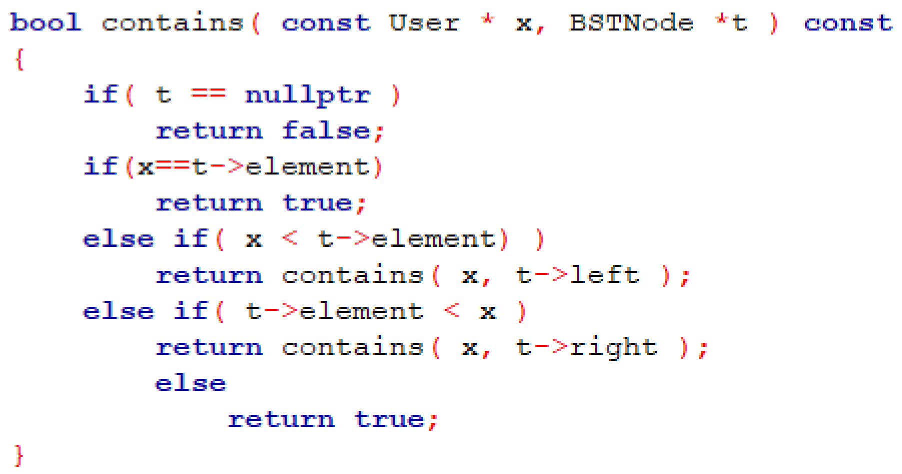
```
THE TEST PROGRAM FOR THE EXCUTION TIME USING AVL TREE
THE AVERAGE SEND TIME USING AVL TREE IS : 0
```

- ***2/- solution*** : using auto duration = duration_cast<nanoseconds>(stop - start);



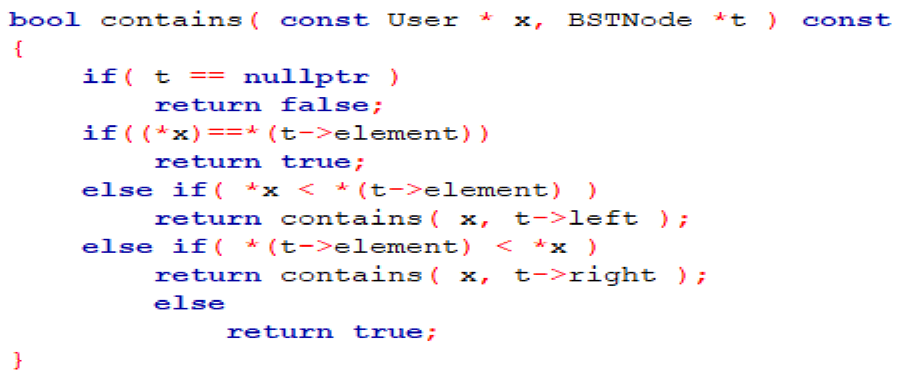
```
THE TEST PROGRAM FOR THE EXCUTION TIME USING AVL TREE
THE AVERAGE SEND TIME USING AVL TREE IS : 10002
```

- ***3/- Problem :*** Losing access to users after Sign-Up or Sign-In ie: not being able to reach, read or check the mailbox of any .
- ***3/- Solution:***
  - Turn the element in all ADT into pointers .
  - Adding the method FindUser that returns a pointer to a wanted element .
- ***4/- Problem*** **:** In ADT 'contains' returns always false .



```
bool contains( const User * x, BSTNode *t ) const
{
    if( t == nullptr )
        return false;
    if(x==t->element)
        return true;
    else if( x < t->element) )
        return contains( x, t->left );
    else if( t->element < x )
        return contains( x, t->right );
        else
            return true;
}
```

- ***4/- Explanation :***
  - - The methods used to verify if the two pointers points to the same element .
- **4/- Solution :**

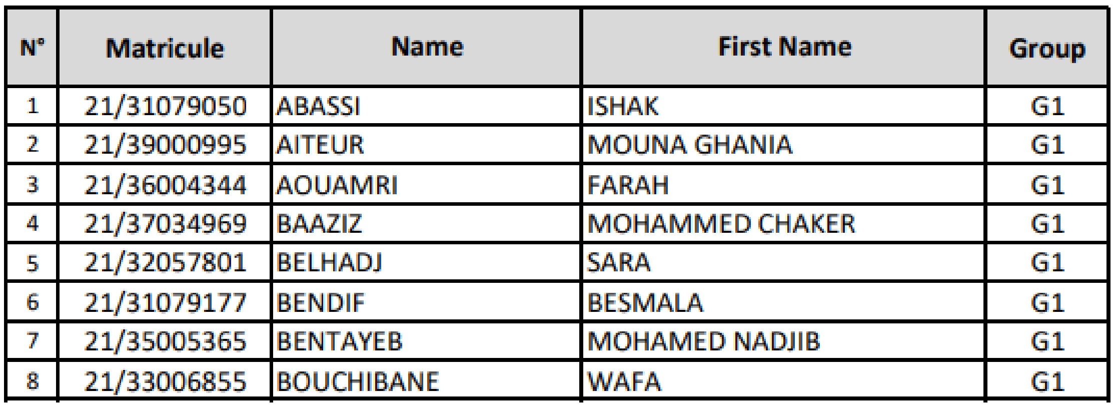


```
bool contains( const User * x, BSTNode *t ) const
{
    if( t == nullptr )
        return false;
    if((*x)==*(t->element))
        return true;
    else if( *x < *(t->element) )
        return contains( x, t->left );
    else if( *(t->element) < *x )
        return contains( x, t->right );
        else
            return true;
}
```

Data : ***Set***

Data collection is a critical part that goes along with the hole process of getting the System prepared and executed .

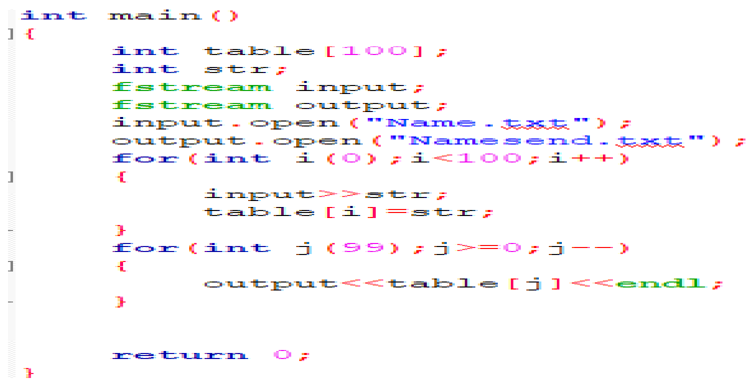
- Testing**:** In this phase a range of data is needed to implement the User in MakeUsers plus the WriteMessages .
    - Names:
        - Getting the original file Name.cpp data from the Group list of the ENSIA second



N°	Matricule	Name	First Name	Group
1	21/31079050	ABASSI	ISHAK	G1
2	21/39000995	AITEUR	MOUNA GHANIA	G1
3	21/36004344	AOUAMRI	FARAH	G1
4	21/37034969	BAAZIZ	MOHAMMED CHAKER	G1
5	21/32057801	BELHADJ	SARA	G1
6	21/31079177	BENDIF	BESMALA	G1
7	21/35005365	BENTAYEB	MOHAMED NADJIB	G1
8	21/33006855	BOUCHIBANE	WAFA	G1

year students.

        - To fill the NameSend.cpp , we created a program to reverse the file Name.cpp, so in the sending process the Users would send emails to others and not themselves.



```
int main()
{
        int table[100];
        int str;
        fstream input;
        fstream output;
        input.open("Name.txt");
        output.open("Namesend.txt");
        for(int i(0);i<100;i++)
        {
                input>>str;
                table[i]=str;
        }
        for(int j(99);j>=0;j--)
        {
                output<<table[j]<<endl;
        }

        return 0;
}
```

    - *ID:*
        - To get ID we generated a program that created a unique ID, then put them in file ID.cpp .
        - With the same profiling as names use the reverse program to get users to send mails to others not themselves , pasting them into IDsend.cpp.

    - *Emails :*
        - The cpp file Email.cpp , was created from a large range of template emails from the net.
            - [34 Free Email Templates & Examples for Small Businesses (localiq.com)]

- _*coding :*_
    - *Implementation :*
        - Using the code implementation from class for the different ADTs “AVL Tree, BSTree, Hash-Table Using Separate Chaining .

The working plan :

The realization of the System went through few phases

- Phase 01 : the understanding of the problem
    - Each team member studied the problem and brainstormed all in the aim of solving it the most optimal way.
    - A pseudo(PC) code was created .
- Phase 02 : the coding
    - The creation of the principal classes .general coding implementation (GI).
    - The implementation of the ADTs for the three parts.
- Phase 03 : data collection .
- Phase 04 : the graphical representation .
    - Explaining the different ADT’s using grapes for Part A.
- Phase 05 : report :
    - Drafting .
    - Explaining the different methods (EM).
    - Explaining problems and their solutions (ES).
    - Discuss the results (DR).

Team members	Phase 01	Phase 02	Phase 03	Phase 04	Phase 05
Lalagui Baraa F/Z	Prepared (PC)	(GI) BSTree	Users		Drafting (EM)
Meriem Djidjeli				Done	Drafting (DR)
Boukedjar Hafida	Prepared (PC)	(GI) AVL tree Hash Table	Emails		(ES)

Reference list :

- Hash Table code implementation _taken from Lab sheets solutions with major changes_
- AVL tree implementation _Boukedjar Hafida implementation_
- BSTree implementation _Boukedjar Hafida implemention_
- The emails data set _34 Free Email Templates & Examples for Small Businesses (localiq.com)_
- The Users data set _from groupe list of ENSIA second year students_