Laboratory practice 3: Linked List and Arraylist

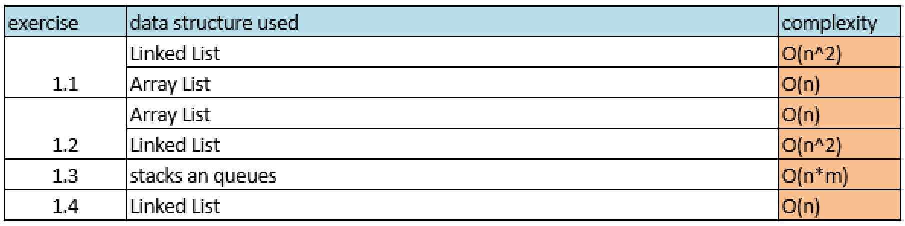
Santiago Isaza Cadavid
Universidad EAFIT
Medellín, Colombia
sisazac@eafit.edu.co

Hamilton Smith Gómez Osorio
Universidad EAFIT
Medellín, Colombia
hsgomezo@eafit.edu.co

October 8, 2018

1) Codes

1.a. Complexities



exercise	data structure used	complexity
	Linked List	O(n^2)
1.1	Array List	O(n)
	Array List	O(n)
1.2	Linked List	O(n^2)
1.3	stacks an queues	O(n*m)
1.4	Linked List	O(n)

1.1 and 1.2 is necessary to analyze the stored data to not add repeated elements so it is more optimal to use ArrayList since this in the get () method has complexity O (1) while LinkedList is O (n) in that method.

1.3 in this exercise it is more effective to use stacks and queues, or by default linkedList because they have a lower complexity when inserting and removing data.

2) Project questions Simulation

2.a. Explain the implementation of Exercise 2.2

For the solution of the problem, we considered a LinkedList, the variables int "index" and boolean "start" helped us to solve the problem in a relatively easy way. A for cycle runs through the entered string, and in case of finding the character "[", the variable "start" will be true and false if we find the character "]", then an if will evaluate other condition, the character at index n, if the character at index is not "[" neither "]" then the status of the variable start is evaluated. If it is true, the character is inserted in the current index of the list, and if it is false, the character is added at the end of the list and in both cases we proceed to increase the variable index, we increase it by one. After running through the entire string, we go through the list of characters already ordered and a new string is created with the characters that are in the list, finally, we return this new string.

2.b. *Exercise 2.2's complexity*

i.
```
public static String pc(String str){
        ArrayList<String> list = new LinkedList<>(); //C1
        boolean start= true; // C2
        int index=0;  //C3
        String newString=""; //C4
        for(int i=0;i<str.length()-1;i++){ //C5*n
            if(str.substring(i,i+1).equals("[")){ //C6*(n-1)
                start=true;//C7*(n-1)
                index=0;//C8*(n-1)
            }
            else if(str.substring(i,i+1).equals("]")){ //C9*(n-1)
                start=false; //C10*(n-1)
            }else if(!str.substring(i,i+1).equals("[") //C11*(n-1)
            && !str.substring(i,i+1).equals("]")){ //C12*(n-1)
                if(start){ //C13*(n-1)
                    list.add(index,str.substring(i,i+1)); //C14*(n-1)*1
                    index++; //C15*(n-1)
                }else{//C16*(n-1)
                    list.add(str.substring(i,i+1));//C17*(n-1)
                }
            }
        }

        for(int i=0;i<list.size();i++){ //C18*n
            newString= newString+list.get(i);//C19*(n-1)
        }

        return newString; //C20
    }
```

$$T(n) = C1 + C2 + C3 + C4 + C5 * n + C6 * (n-1)$$
$$C7 * (n-1) + C8 * (n-1) + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) +$$
$$C12 * (n-1) + C13 * (n-1) + C14 * (n-1) * 1 + C15 * (n-1) + C16 * (n-1) +$$
$$C17 * (n-1) + C18 * n + C19 * (n-1) + C20$$

$$T(n) = O(C1 + C2 + C3 + C4 + C5 * n + C6 * (n-1)$$
$$C7 * (n-1) + C8 * (n-1) + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) +$$
$$C12 * (n-1) + C13 * (n-1) + C14 * (n-1) * 1 + C15 * (n-1) + C16 * (n-1) +$$
$$C17 * (n-1) + C18 * n + C19 * (n-1) + C20)$$

$$T(n) = C1 + C2 + C3 + C4 + C5 * n + C6 * (n-1) + C7 * (n-1) +$$
$$C7 * (n-1) + C8 * (n-1) + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) +$$
$$C12 * (n-1) + C13 * (n-1) + C14 * (n-1) * 1 + C15 * (n-1) + C16 * (n-1) +$$
$$C17 * (n-1) + C18 * n + C19 * (n-1) + C20$$

$$T(n) = O(C1 + C2 + C3 + C4 + C5 * n + C6 * (n-1) + C7 * (n-1)+)$$
$$C7 * (n-1) + C8 * (n-1) + C9 * (n-1) + C10 * (n-1) + C11 * (n-1) +$$
$$C12 * (n-1) + C13 * (n-1) + C14 * (n-1) * 1 + C15 * (n-1) + C16 * (n-1) +$$
$$C17 * (n-1) + C18 * n + C19 * (n-1) + C20$$
$$T(n) = O(14 * (n-1) + n)$$
$$T(n) = O(13n - 14)$$
$$T(n) = O(13n)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O(n)

2.c. Explain what the variables means in the previous exercises

- **LinkedList list:** the list, data structure, where the strings are going to be stored, in order to the status of the variable start.
- **Boolean start:** determines if the substrings are going to be added at the end of the list or at the beginning.
- **Integer index:** does the function of a counter, determines the current position at the list.
- **String newString:** is the string in which the substrings stored in the list are going to be added, according to the their position in the list.

3) Midterm Simulation

3.a. Exercise 1

a) Looking for data in the list

3.b. Exercise 2

c) O(n)

3.c. Exercise 3

3.3.1 Complete line 02

q.size()>1

3.3.2 Complete line 03

<=

3.3.3 Complete line 04

q.remove()

3.3.4 Complete line 06

q.remove()

3.d. Exercise 4

3.4.1 What is the condition of the while cycle?

lista.size()

3.4.2 Complete line 07

lista.add(auxiliar.pop())

3.e. Exercise 5

3.5.1 What is the condition of the while cycle? Lines 12 and 16

auxiliar1.size()>0 , auxiliar2.size()>0

3.5.2 Complete line 18

personas.offer(edad)

3.f. Exercise 6

c) O(n^2)

3.g. Exercise 7

c) O(n^3)

3.h. Exercise 8

d) O(1)

3.i. Exercise 9

3.9.1 What is the asymptotic complexity in the worst case-scenario?

a) O(k)

3.9.2 What does the algorithm prints when k=21

b)9

3.9.3 What is the asymptotic complexity in the worst case-scenario, when you are adding data to a queue of n elements?

c) O(1)

3.j. Exercise 10

3.10.1 What is the asymptotic complexity in the worst case-scenario?

d) O(n)

3.10.2 What does the algorithm prints when x=8 and n=20

a)6

3.10.3 What is the asymptotic complexity in the worst case-scenario, when you are searching whether data is or not in a stack?

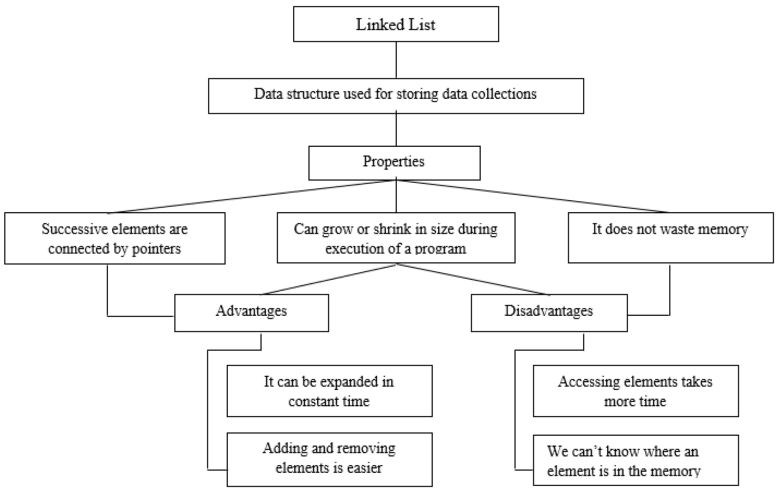
b) O(n)

4) Recommended reading

4.a. Summary

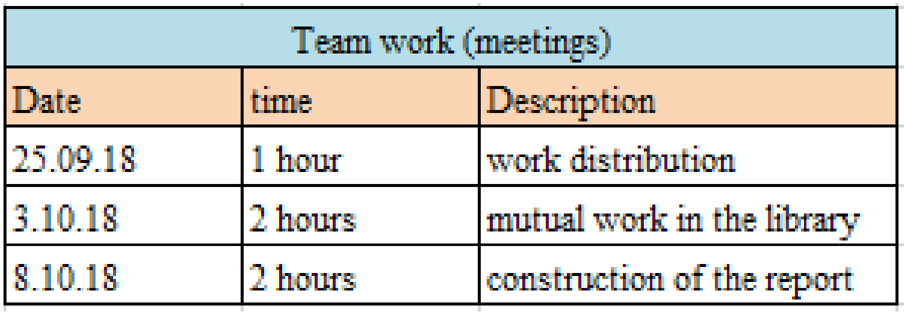
The linked lists are data structures in which is possible to store different types of elements without being properly organized in the memory of the computer, it means, it has the control over the consecutive order of the elements by links or connectors between elements but these can also be anywhere in the memory.

This structure is very useful because it allows to optimize the memory space, it has a shorter execution time when inserting or deleting an element and it is very used to build other structures such as stacks or queues. For its implementation nodes are usually used, which are the "closets" that are interconnected and store the desired information, which have positional attributes referring to the data next to it, and in the case of double-linked list, the previous element.


Linked List
Data structure used for storing data collections
Properties
Successive elements are connected by pointers
Can grow or shrink in size during execution of a program
It does not waste memory
Advantages
Disadvantages
It can be expanded in constant time
Adding and removing elements is easier
Accessing elements takes more time
We can't know where an element is in the memory

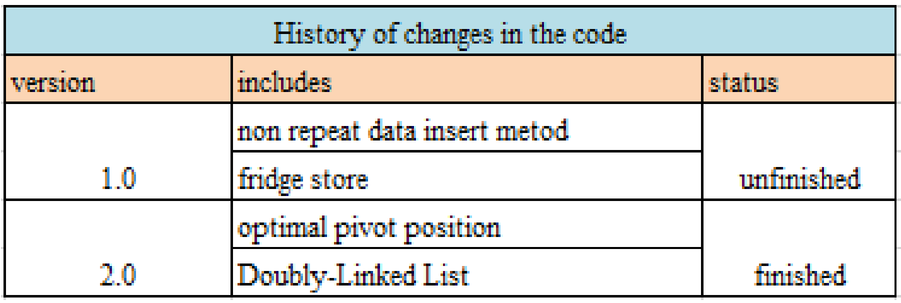
5) *Team work and progress*

5.a. *Meetings*



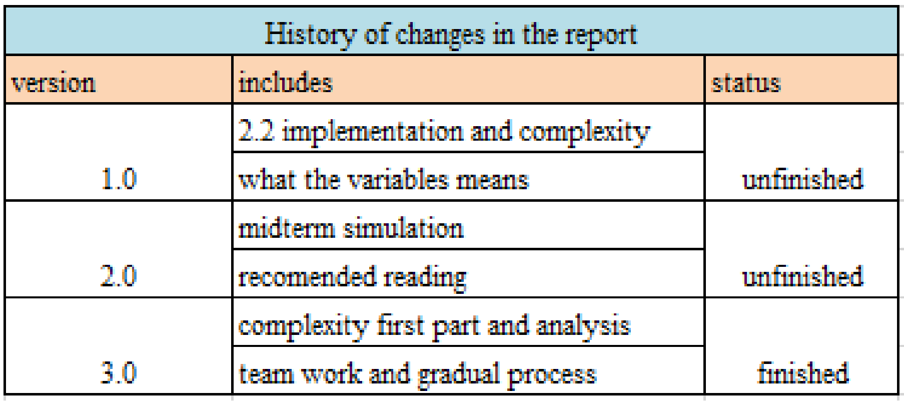
Team work (meetings)		
Date	time	Description
25.09.18	1 hour	work distribution
3.10.18	2 hours	mutual work in the library
8.10.18	2 hours	construction of the report

5.b. *History of changes in the code*



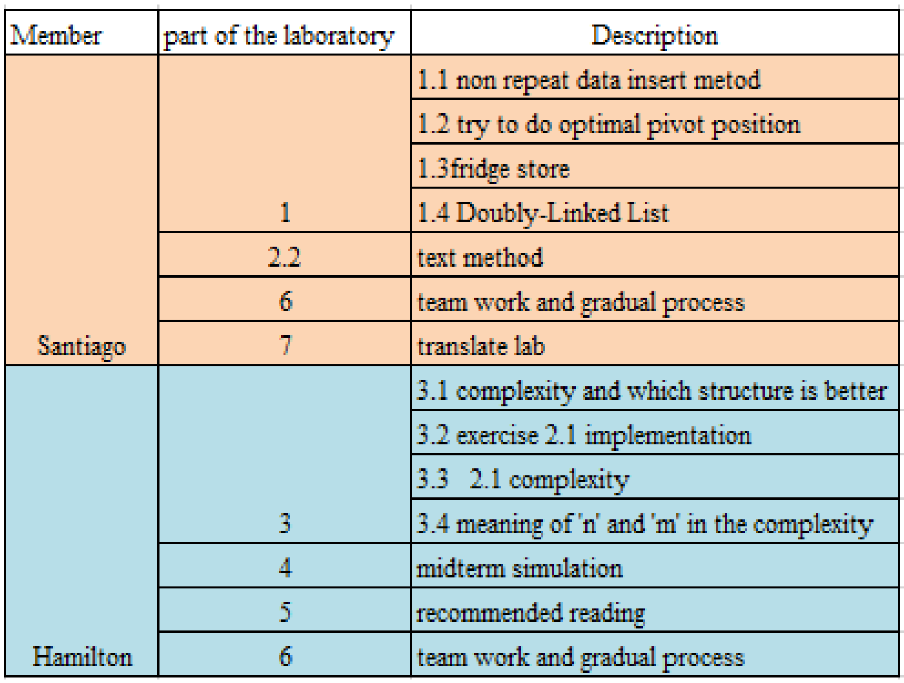
History of changes in the code		
version	includes	status
	non repeat data insert metod	
1.0	fridge store	unfinished
	optimal pivot position	
2.0	Doubly-Linked List	finished

5.c. *History of changes in the report*

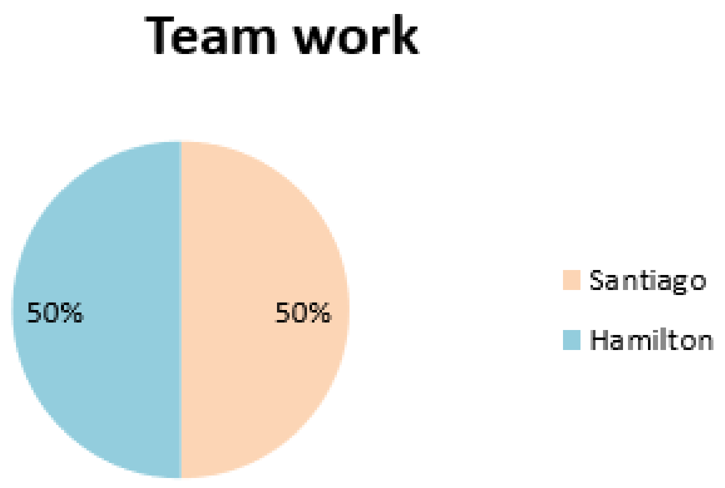


History of changes in the report		
version	includes	status
	2.2 implementation and complexity	
1.0	what the variables means	unfinished
	midterm simulation	
2.0	recomended reading	unfinished
	complexity first part and analysis	
3.0	team work and gradual process	finished

5.d. Team Work



Member	part of the laboratory	Description
Santiago	1	1.1 non repeat data insert metod
		1.2 try to do optimal pivot position
		1.3fridge store
		1.4 Doubly-Linked List
	2.2	text method
	6	team work and gradual process
	7	translate lab
Hamilton	3	3.1 complexity and which structure is better
		3.2 exercise 2.1 implementation
		3.3 2.1 complexity
		3.4 meaning of 'n' and 'm' in the complexity
	4	midterm simulation
	5	recommended reading
	6	team work and gradual process


Team work
Santiago
Hamilton