Laboratory practice No. 4: LinkedList

Juan S. Cárdenas Rodríguez
Universidad EAFIT
Medellín, Colombia
jscardenar@eafit.edu.co

David Plazas Escudero
Universidad EAFIT
Medellín, Colombia
dplazas@eafit.edu.co

October 8, 2017

1) *CODE FOR DELIVERING ON GITHUB*

The source code can be found in `Code.py` inside the `codigo` folder.

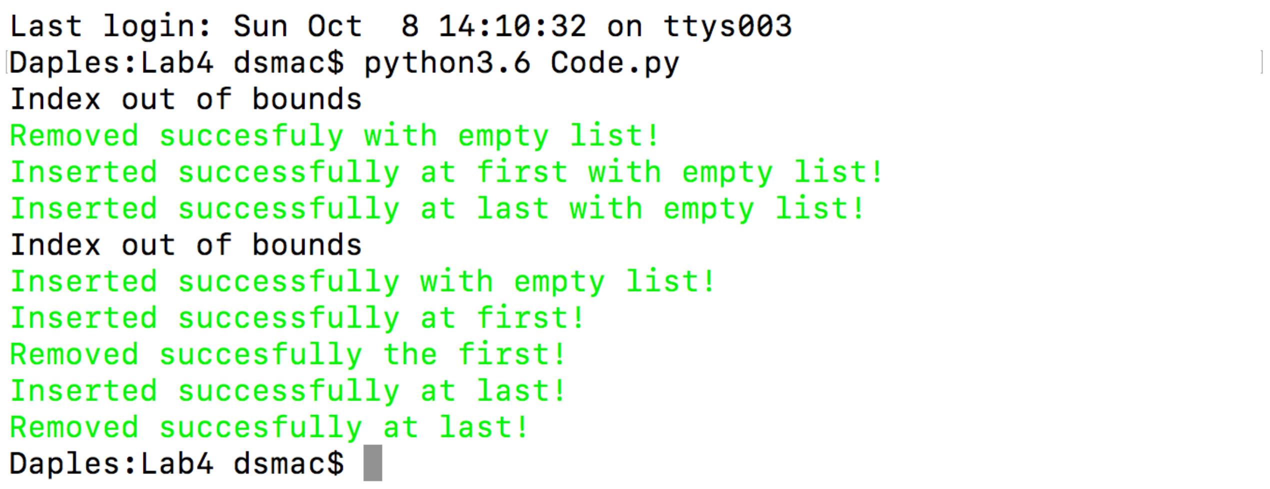
2) *ONLINE EXERCISES*

The source code can be found in `Code.py` inside the `codigo` folder.

3) *SIMULATION OF PROYECT PRESENTATION QUESTIONS*

3.a. *Tests*

You can see the code in `Code.py` inside the `codigo` folder for further details about each test.



```
Last login: Sun Oct  8 14:10:32 on ttys003
Daples:Lab4 dsmac$ python3.6 Code.py
Index out of bounds
Removed succesfuly with empty list!
Inserted successfully at first with empty list!
Inserted successfully at last with empty list!
Index out of bounds
Inserted successfully with empty list!
Inserted successfully at first!
Removed succesfully the first!
Inserted successfully at last!
Removed succesfully at last!
Daples:Lab4 dsmac$
```

Figure 1: Tests.

3.b. How does exercise 2.1 work?

In first place, we recieve the number of blocks that the user wants therefore, we create in a loop the linked list so the stack can have the form that the user needs. After that, we wait for the user to input a command and, then we process it so we know how much is a, b and what operation do we need to do.

So, in second place, we check if a is equal to b or if they are in the same column; if one of this statements is true we just proceed to write the error and continue to wait to another instruction. On the other hand, we just make the operation he asks us to do. The operations are:

- Move a over b: In this operation, we just search for a in all of the stacks that our linked list has. To search in the stacks, we just create a temporary stack named ”stack” and everything that we pop from the original is stored in this auxiliar. If we found a, we just pop it ouf the temporary stack and, all of the other elements in the data structure, we send them to the stack at the index of the linked list that corresponds to their number. After that, we just search for b doing the same process and, when we find it, we just insert a at the end of the stack.

- Move a onto b: To do this operation, we call the method “move_over” that just makes the process described above. And, after that, we just search in the linked list till finding b and, removing everything that is in between a and b doing a analogous algorithm as the move a over b.

- Pile a over b: To do this algorithm, we search our list in the same way that we did in the operations described below; but, when we found a, we save all the numbers that he had on top of him (including a) to another auxiliary stack, a_pile. After that, we just find where b is and pile the stacks on top of each other.

- Pile a onto b: To run this process, we do something like the operation we made on the move a onto b. We just call the method “pile_over” and, after that, we just erase what is in between a and b.

3.c. What's the complexity of exercise 2.1?

Therefore, `manage_string` is $O(c_{k1} + c_{k2}m + c_{k3}n + c_{k4}mn)$. When the product and sum properties are applied, `manage_string` is $O(mn)$, where n is the length of the string and m is the number of lines.

4) TEST SIMULATION

i.
- auxiliar.size()
- lista.add(auxiliar.pop())

ii.
- auxiliar1.size() > 0
- auxiliar2.size() > 0
- personas.offer(edad)

iii. c) $O(n^2)$