UNIVERSITY OF COPENHAGEN
Department Of Computer Science DIKU

Bachelor Project

A Visualization Tool for Learning Algorithms and Datastructures

Midtvejsrapport

2016 block 3-4

Valdas Zabulionis

Rune Franch Pedersen

Amr El Sayed

April 11, 2016

A Visualization Tool for Learning Algorithms and Datastructures

DIKU - Bachelor Project

Studens Names	:	Valdas Zabulionis - LHN100
	:	Rune Franch Pedersen - VQR730
	:	Amr El Sayed - VWJ159
University	:	University of Copenhagen
Institution	:	Department of Computer Science DIKU
Supervisor	:	Oleksandr Shturmov
Period	:	Block 3 - block 4
Year	:	2016
Pages	:	_

Certificate

This is to certify that the work contained in the thesis entitled "A Visualization Tool for Learning Algorithms and Datastructures" by Valdas Zabulionis, Rune Franch Pedersen and Amr El Sayed, has been carried out under our supervision and that this work has not been submitted elsewhere.

Supervisor
Oleksandr Shturmov
oleks@di.ku.dk
Department of Computer Science DIKU
University of Copenhagen

Contents

Part I

Introduction

1 Introduction

1.1 About the Project

A Visualization Tool for Learning Algorithms and Datastructures is a web-based learning tool conceptualized by Oleksandr Shturmov, Amr El Sayed, Valdas Zabulionis and Rune Franch Pedersen with the aims of improving teaching of data structures and algorithms through dynamic interactive visualizations. Students from Computer Science institute Copenhagen University will use our tool to help them with understand and implement algorithms.
The tool consists of two major components: the visualization component and the training component.

1.2 Problem definition

For students, who are relatively unfamiliar with algorithms and programming, getting a comprehensive understanding of algorithms can be very difficult.
In this project we will design a visual tool to help student create their own basic algorithms and do a step-by-step run-through of them. This is intended to help students learn the basics of algorithms.
The primary focus is an easy to use tool, which will use visual building blocks, that are predefined statements providing function body, statements, condition expressions, cases, loops, etc.. They can be easily used by the students to create an algorithm. The tool would also include a built-in function which will provide a detailed step-by-step description of each step in the implemented algorithm.

1.3 Scope

2 Problem Analysis

To create a program that helps students, we face many problems that have to be solved. The first and most important problem is to choose how the users gain access to the program, in other words how it is delivered.

Second is the problem of implementation. We must choose the subjectively best tools for implementing the program. This problem contains the choices of programming languages.

Another problem would be figuring out how the UI should look, so that it is relatively straightforward for the users to use, and at the same time it should be relatively straight forward to upgrade by adding new items or visuals.

Then there is the problem of making the UI frontend work with the backend.

How should the user interact with the program. How should they be able to create algorithms.
How to visualize an algorithm, how to show what is happening using graphics.
How to recognize what the user input is in the algorithm creation field, and how to compute it, and at the same time how to display it to the user (step-by-step).

This program is intended to be used by lecturers and teaching assistants as a tool for helping students learn algorithms. It is important that the program could have a set of building blocks specific for each week of the course for the students to use. It is also very important that the the building blocks are easy to add, easy to be removed, and are easily edited to correct bugs.

Part II
Design

Design

3 Design

To make it easier for students to have access to our program, we have decided that it should be web-based - meaning that the it should be accessed via internet by using a standard web browser. This way there would be no need to distribute the program and all of the required packages for it to run, as was our first thought. By making the program web-based, we ensure the accessibility of it for most, if not all, users.
The choice of programming language for our project is not a clear one. Because we decided to make it web-based, a possibility is to use Javascript for the whole implementation, but we decided that the code itself would quickly become hard to decipher. So we thought of splitting the program into two parts - the frontend and the backend. Frontend would be coded in HTML, CSS, and JavaScript, and the backend would be coded in a programming language of our choice.
At the moment we have two programming languages in mind for our backend. One of them is Python. Python is a choice because we are relatively familiar with it, and it is easily coded. A downside would be, that we have heard that is it relatively complicated to write server-client packages and modify their contents.
The other language that we have in mind is GO, which is quickly becoming a popular programming language. A pro for it is, that it is easier to handle server-client requests using GO. It would also be a good thing to learn one more programming language, especially one that is on the rise in popularity and may become one of the more used programming languages in the world.

We have decided that the UI should be separated into three windows. On the left, the user would have access to various buttons that would insert functions, variables, etc. into the main window which would be placed in the middle of the screen. This main window would be like a notepad, where the user would try to implement their algorithm. On the right side, there would be a small window with a visualization component for showing what happens while running the algorithm step-by-step. Below it there would be a log window, where the user would see values of various variables and function calls.

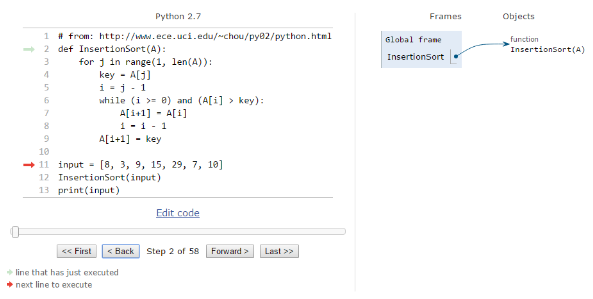
To solve the problem of communication between the frontend and the backend, we need to do some more background studies. We have a couple of ideas of how to implement function calls, what data packages and when to send them from the client to the server. We should have a clearer idea of how to solve this problem, when we have moved further along into the implementation process.
A big part of our project is to visualize what is happening when a given algorithm is being run. For example the students would like to build an implementation of the heap-sort

algorithm. They would have the choice to select the heap-sort algorithm from a list of algorithm, and when they are finished implementing and would like to run the step-by-step program, they would be shown what happens in a heap at the same time that they are running their algorithm. This functionality would be done in a separate module, so that it would be possible to upgrade it at a later time, since we are only going to implement this only as a place-holder, so it won't look as good as it could if done with great care for the visual design.

How to recognize user input, and how to read their algorithm, this is still a work in progress. The idea is for users to click/drag-and-drop building blocks which would have fields where the users would write their input. This input would be saved client side for a fixed amount of time, and it would be sent as a series of instructions to the server. The order in which the building blocks would appear in the middle of the screen text field, would decide how the server would read and execute instructions it receives. The point of saving user input client side, is to ensure that their work progress is not lost if they suddenly lose internet connection. This would also help to decrease the connection load server side, so as not to have a constant stream of data, but only instructions being sent when requested or needed.

4 Inspiration

To get some inspiration for our work, we tried to find similar projects to see how they have designed their layout and what kind of functionality they support.
Pythontutor **?** is a tool which lets the user execute python code one step at a time, giving the users a visual representation what is happening while the code is running at a given step.



```
# from: http://www.ece.uci.edu/~chou/py02/python.html
def InsertionSort(A):
    for j in range(1, len(A)):
        key = A[j]
        i = j - 1
        while (i >= 0) and (A[i] > key):
            A[i+1] = A[i]
            i = i - 1
        A[i+1] = key

input = [8, 3, 9, 15, 29, 7, 10]
InsertionSort(input)
print(input)
```

Figure 1:

Part III
Implementation

5 Implementation

5.1 UI

For the implementation of the user interface we have chosen to use Bootstrap. Bootstrap is a front-end framework for web development. This tool helps us flesh out the user interface, ensuring that it looks the same no matter what platform the user is using to access the website.
The interface will support the expansion of the implemented buttons used to create the algorithm. The interface is intended to use some functionality to include files on the server, which

represent the buttons, that is the building blocks discussed below. This is a functionality we are still designing at this time.

5.2 Building Block Grammar

The building blocks are intended to be files written in the either Python or GO, depending on which one we settle on. One file will contain the code for one button on the interface. The interface will automatically include all the files that match a predefined blueprint, that we provide, thereby making it possible to easily add further functionality to our system. These files will be server-side and only people with access to the server will therefore be capable of adding more functionality. The button added by the file will allow a user of our system to add a function or value to the algorithm that the user is trying to implement, this could be a loop, binary operator or whatever the one who has written the file wishes too implement, as long as it fits the blueprint. We will add the needed building blocks to implement a few basic algorithms. The grammar discussed below are how we intend to implement the building blocks we design. The building blocks will need to include all the information that is needed by the interface to process the functionality of the button. Therefore it is necessary for the building blocks to contain the information for the button, the mathematics that the functionality needs to perform and the text for the algorithm as well as the log.

We have completed the definition of the building block grammar. There might be some improvements or corrections at a later date, but this should as close to the final version as possible.
By dividing the building blocks up in certain types we can ensure that we avoid situations where we get nonsense code in the algorithm, like a while loop that doesn't contain boolean value for it's condition statement. We also intend to give the building blocks a hierarchy, that will force the user to do things in a certain order thereby making it easier implement the backend.

PROCEDURE ::= *ProcName* $'('$ *Args* $')'$
Lines

ARGS ::= *ValName*$','$*Args*
| *ValName*

LINES ::= *Line* $'\backslash n'$ *Lines*
| Line

LINE ::='*return*' *Expr*
| PROCNAME '(' Args ')'
| *Expr*

EXPR ::= $\lfloor Expr \rfloor$
| $\lceil Expr \rceil$
| *Expr* 'Binary Operator' *Expr*
| *Value*
| *ValName*
| $'['$*Values*$']'$

ARRAY ::= [*Values*]

VALUES ::= *Values*$','$*Value*
| *Value*

VALUE ::= *Const*

LOOP ::='*for*' (*Statement*)
Lines
| '*while*' '(' *Statement* ')'
Lines

STATEMENT ::= *Expr* 'Boolean Operator' *Expr*
| *Statement* 'AND' *Statement*
| *Statement* 'OR' *Statement*

6 Work progress

At this point in time we have partially completed the first work task which was to design and implement the interface for our program. We have mostly completed the graphical design of the program, and what is left is to connect the HTML code to our backend, all the while adding some finishing touches to the visual representation of the interface.
We have completed the grammar needed to create building blocks, and this should probably be the final version of the grammar that we will be using in our program.

6.1 Time schedule

We have fallen a bit behind schedule with implementing the interface, and this could mean delays for the rest of our schedule. The implementation of building blocks could be moved a week back. The same goes for the import/export functionality, since this should be done after implementing the building blocks. Other than that, our work schedule is pretty much not unchanged.

- Building blocks - date changed to 27/4 from 20/4
- Import/export - date changed to 01/5 from 23/4
- Step-by-Step simulation - date unchanged 15/5
- Pretty print - date unchanged 01/6

The current delay could mean that the implementation of our step-by-step simulation being delayed as well, but we still should have time buffer after our last item (the pretty-print) before our deadline to mitigate any further delays.

https://scratch.mit.edu/
http://pythontutor.com/