6CCS3PRJ

Background and Specification Progress Report

Alessandro Amantini

December 16, 2020

1 Introduction

1.1 Project Motivation

The number of students taking computer science modules every year seems to be a steadily growing trend, proving to be one of the most loved university degrees. Despite its success, this subject has often held the highest ranks of the most dropped university subjects.

Donald E. Knuth, emeritus computer science professor at Stanford University, claimed this discipline can be compared to problem solving [7]. Fundamental for solving problem are data structures and algorithms, hence, failing to adequately teach them will inevitably lead to more students dropping from the course. Unfortunately, in spite of the advance in technologies offering new and more sophisticate software every year, outdated teaching approaches are still pretty common in academic environments, negatively impacting students.

1.2 Aim and Objectives

The idea behind this project is to assist academic tutors teaching algorithms and data structures by helping them create an animation of the latter. The goal is to deliver a website which includes a page for the instructor, or **teacher**, who can generate the algorithm or data structure visualisation of their choice and export it as a configuration file; another page will be addressed to the final user, or **student**, whom, importing the configuration file, will see and play with the graphical representation of the algorithm or data structure.

The aim of the graphical visualisation of the algorithm or data structure is to have students interact with it to increase their engagement and facilitate their learning in what is considered to be a fundamental, yet very hard, topic of computer science.

2 Background

2.1 Dropping rate

From the close relationship between IBM and the Columbia University, one of the first academic-credit courses in computing emerged in 1946 [4]; it had then been about 15 years to the establishment of the first Department of Computer Sciences in the United States, at Purdue University, in October 1962 [2]. The number of students taking majors in computer science has kept increasing ever since, becoming the second most-loved subject in 2019 [5]. In spite of this form of appreciation, a lot of students drop it every year: a study conducted from the Higher Education Statistics Agency [3] identified computer science as the most dropped discipline of the year both in 2016/2017 and 2017/2018, with respective dropping rates of 9.9% and 9.8%.

2.2 Teaching issues

There are several reasons why students decide to drop, some related to personal issues, others directly derived from teaching approaches, such as the use of outdated teaching-learning methods [10], which result in students':

- lack of motivation;
- disinterest; and even
- learning difficulties.

However, teaching is not an easy job: teachers must have a sound understanding of the subject, they have to be able to identify the key-points to clearly express them, and they have to efficiently retain students' interest and engagement. Many universities employ graduate students to teach low-level computer science modules, but most of the time, graduate students, who are not capable of teaching these courses, end up covering instructor roles with little or no training, with a negative impact on their students.

2.3 Data Structures and Algorithms Definition

Computer science pivots around data: storing, elaborating, and accessing information can be regarded as the three core functions of a computer. To facilitate these operations, organised collectors known as **data structures** are commonly used. Data stored inside data structures are generally accessed and modified by **algorithms**. Algorithms are a finite sequence of

well-defined instructions used to solve a problem; they are deterministic, meaning the output is always predictable, and can be expressed using a finite amount of space and time.

Due to their interdependence, data structures and algorithms are often seen as two sides of the same coin. Their combined knowledge is fundamental for a computer scientist as they are widely used across IT, and mastering them is crucial to elaborate effective solutions to problems. For these reasons, they are taught together as a unique module during the first year of study in the discipline, and then required throughout the rest of the degree. A solid understanding of the matter will not only positively affect students' current performance, but also their future career paths.

3 The Power of Visualisation

To better understand a concept, people often need a graphical illustration of it, and when the latter is not available, we tend to recreate it in our mind picturing a visual, although mental, representation of it. The operations used to interact with data structures, and algorithms in general, are abstract concepts representing motion, as such, it is not possible to efficiently represent them using only static pictures. For this reason, instructors used to draw intricate diagrams of data structures and control flow on blackboards, simulating dynamism, but they often made errors like skipping or rearranging steps, confusing values, or improperly estimating the space needed for a good layout [11]. Using an animation software would be a more compelling solution for the display of algorithms behaviour as, if correctly implemented, it would remove errors, respect all the steps, and picture the problem in a neater way.

Visualising the steps of an algorithm would deliver cognitive support, so learners are facilitated to grasp the concept. The visualisation would depict the dynamics of time-view progression, assisting learners in comparing the algorithms and comprehending how they work [15].

4 Literature and Website Review

4.1 Theoretical Literature

Algorithms, including operations performed on data structures, are abstract and complex concepts, usually hard to grasp for new computer scientist students, but their visualisation can be exploited to support the learning process in the matter. According to Pavio's **Dual Code Theory**, in fact, the formation of mental images aids learning by allowing to later recall concepts cued by words and pictures [13]. However, the use of depictive representations [14] (pictures and physical models) should not replace descriptive representations [14] (written texts, mathematical equations and logical expressions), rather, the two of them should be used in conjunction to generate an optimal learning outcome [9]. Within my project, descriptive representations are provided in two shapes:

- the display code, highlighted at the right moment; and
- the respective comments.

The animation itself, instead, embodies the depictive representation.

Besides the three ones just cited, no other representation is directly allowed by the software as the information provided is already complete, and other related information, whether depictive or descriptive, would produce an attempt by the student to mentally integrate more unnecessary content [9].

The idea of visualising the algorithm is to deliver cognitive support, that is, to simplify the main concept by turning abstract concepts into concrete and visible representations, and to display the dynamics using motion. Also, allowing to generate different algorithms that rely on the same graphical construct provides an effective mean to compare them [15].

One important aspect to underline is that, according to experiments conducted on students, the sole animation does not bring reliable benefit in teaching how to solve procedural and conceptual problems about algorithms [8], and it is necessary to integrate it with gamification. The use of game and related elements, in fact, engages and motivates learners improving their learning outcome [15]. Within the software, I decided to provide a toolbar that allows the user to stop, resume, and change the speed of the animation to better engage them. Moreover, the software gives the possibility to define the input data to see how different inputs affect the algorithm as viewing animations for data provided by the users positively affect their learning [12],

4.2 Practical References

To define the design and the mechanisms of the software, I explored the web, looking for similar projects. The best websites I found which display a graphical representation of how algorithms and data structures work were **University of San Francisco Data Visualizer** [1] and **Visualgo** [6], as such, I decided to rely on them to shape my own one.

The University of San Francisco Data Visualizer is very rich in the sense that comes with many algorithms out-of-the-box; it also exposes its API giving creators the possibility to make their own algorithm and to import it inside a web page. Unfortunately, the layout is very bad, there is no text explaining what is going on, the animations, although nice to watch, do not contribute much to the actual understanding of the algorithm, and the documentation to access the API is very hard to read and to understand.

Visualgo, on the other hand, is remarkable: comes with the translation in many languages, is full of information about each step performed by the algorithms, which helps really understand what is going on, and the overall layout is much more pleasant to look at. The downside is the lack of extensibility, in fact, should someone want to define their own algorithm, the program does not allow that by any means.

From the comparison of these projects, I defined mine, which aims to be pretty similar to Visualgo, but offering high levels of customisation and extensibility to the teacher user.

4.3 Extra Literature

There are some further considerations that can be made about the generated animation to improve its effectiveness [12], like:

- using black-and-white animations;
- displaying labelled data;
- prompting the user to make predictions;
- including sounds.

However, these may not appear in the final project.

5 Requirements

The program comes with a total of three pages, intended to be used by two types of actors:

- **Student**: this actor can be seen as the final user who will benefit from the visualisation of the animation. The only page dedicated to them is the one to play and interact with the visualisation of the algorithm or data structure.
- **Teacher**: this is the person who will configure the algorithm or data structure animation. Two pages are dedicated to this user, the instructions page and the one to create the configurations.

5.1 Use cases

Students will interact with the program via a basic graphic interface which allows them to:

- Import a JSON configuration file — JSON understanding is not required — to define the animation to visualise.
- Depending on the configuration, interact with the program to see how it behaves in different scenarios.
- Stop, resume, and change the speed of the animation.

The teacher's interaction with the program will be mainly to generate and test an animation; this user will model the main aspects of the configuration such as:

- The data structures and data type that the program will use.
- The Javascript process that defines the animation.
- The code and comments displayed to the student user.

5.2 Users Requirements

The program must meet the following user requirements:

- The program must be easy to understand and to use.
- The GUI must be clean, avoiding elements that may distract the user.

- The generation of a configuration file and its conversion into an animation must be reasonably fast.
- The program must provide a user-friendly environment to generate and display the animation.

5.3 Functional Requirements

The program should allow the teacher user to define customisable animations that a student can benefit from, as such, it should meet the following requirements:

- The program must be able to read Javascript code supplied by the user and use it to power the animations.
- The program must be able to accept non-executable code and comments, and to properly display them.
- The program must be able to generate animations which respect the steps described by the algorithm supplied by the user.
- The program must be able to present the data in a graph, plot or list-like shape according to the user preference.
- The program must provide interactive functionalities to facilitate students' learning and retain their interest.
- The program must provide a way to export and import the configuration of an animation as a file.

5.4 Hardware & Software Requirements

The program will run in a web environment, so the only requirement is a browser able to display light animations and that does not block Javascript code execution. The teacher user will be able to access the web page from any device, but the use of any environment different from a laptop is highly discouraged for proficiency reasons. For a better visualisation, users are discouraged to use smartphones, but the program will run anyway.

6 Specification

6.1 Required Knowledge from Teacher Users

The teacher user is provided with exhaustive information on how to interact with the software, however, there is some pre-existing knowledge necessary to properly interact with it:

- Javascript knowledge is required as the language used to specify the code to run by the program.
- The user should understand Javascript asynchronicity to grasp how the program will produce the animations.
- The user is expected to use the generator functions keyword 'yield', so being able to code using yield syntax notation is required, but generators knowledge is not.
- Solid control structures understanding is recommended to generate better visual output.
- Solid knowledge of data structures and algorithms is required, but very depending on the type of code the user intends to generate.

At the discretion of the teacher user who provides the configuration file, the student may need to understand a specific programming language or pseudo-code.

6.2 Software, Frameworks, and Libraries

6.2.1 Angular 2+

In order to achieve clean-code, and to respect good programming practices, I relied on Angular2+. Angular is one of the most established fronted frameworks used to create single page applications in Javascript. Its structure allows to create reusable and maintainable code, easy to read and to extend; it also forces me to use **Typescript**, a superset of Javascript which allows for typed code reducing the risk of bugs.

6.2.2 Monaco Code Editor

To provide the teacher user with the best experience, the 'generator page' includes a Javascript IDE called 'Monaco', based on the popular code editor VS Code. The user is not by any means forced to use the embedded editor, but its presence is meant to speed up the creation of the configuration file.

7 Design & Implementation

7.1 GUI Architecture

The website is composed of three pages:

- **Instructions Page**: this page aims to support the teacher in the realisation of the configuration file by providing examples, rules to write well-formed Javascript code that correctly interacts with the system, and explaining what each component in the generator page does.
- **Generator Page**: this is the actual page that the teacher user will need to create the animation configuration as described by the previous page. A preview of the output will be available immediately to the user without the need to switch page.
- **Visualiser Page**: here the student user will be able to import the configuration file to display the animation they can interact with.

7.2 Software Inputs

To create the configuration file that will be used by the student user, the software needs the following:

- The executable Javascript code: this is the actual code that will be run by the program. The interaction with the endpoints exposed by the software code should happen here, allowing the teacher to model the animation and the underlying functionality.
- The code-comment blocks: this is a collection of lines, each composed of 2 elements – a line of code and its comment. This code will not be run, rather, it will be displayed to student users to help them understand how a given algorithm or data structure works, without the boilerplate required from the Javascript code. Not being actually compiled, it can be written in any language, even pseudo-code.

References

[1] Data structure visualizations. https://www.cs.usfca.edu/~galles/visualization/Algorithms.html. Accessed: 2020-12-15.

[2] History of the department. https://www.cs.purdue.edu/history/#:~:text=INTRODUCTION,natural%20phases%20in%20its%20history. Accessed: 2020-12-15.

[3] Non-continuation: Uk performance indicators 2018/19. https://www.hesa.ac.uk/data-and-analysis/performance-indicators/non-continuation-1819. Accessed: 2020-12-15.

[4] The origins of computer science. https://www.ibm.com/ibm/history/ibm100/us/en/icons/compsci/#:~:text=The%20first%20computer%20science%20departments,the%20pioneers%20in%20the%20field. Accessed: 2020-12-15.

[5] University subject rankings: Top ten most viewed subjects. https://www.topuniversities.com/university-rankings-articles/university-subject-rankings/university-subject-rankings-top-ten-most-viewed-subjects. Accessed: 2020-12-15.

[6] visualising data structures and algorithms through animation. https://visualgo.net/en. Accessed: 2020-12-15.

[7] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: Some thoughts and observations. *SIGCSE Bull.*, 37(2):103–106, June 2005.

[8] Michael Dwyer Byrne, Richard Catrambone, and John T Stasko. Do algorithm animations aid learning? Technical report, Georgia Institute of Technology, 1996.

[9] Paul Chandler and John Sweller. Cognitive load theory and the format of instruction. *Cognition and Instruction*, 8(4):293–332, 1991.

[10] P. da Silva Neves Lima, L. das Almas Silva, J. L. dos Santos Oliveira, A. A. Franco Brandão, and L. de Oliveira Brandão. Computational games in stem courses: a systematic review of the literature. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2020.

[11] M. Erwig. *Sorting out Sorting*, pages 101–118. 2017.

[12] CHRISTOPHER D. HUNDHAUSEN, SARAH A. DOUGLAS, and JOHN T. STASKO. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages Computing*, 13(3):259 – 290, 2002.

[13] Allan Paivio. *Dual Coding Theory*, pages 53–83. 09 1990.

[14] Wolfgang Schnotz and Maria Bannert. Construction and interference in learning from multiple representation. *Learning and instruction*, 13(2):141–156, 2003.

[15] A. Yohannis and Y. Prabowo. Sort attack: Visualization and gamification of sorting algorithm learning. In *2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games)*, pages 1–8, 2015.