Egyptian Informatics Journal 20 (2019) 193–204

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Integrating program and algorithm visualisation for learning data structure implementation

Rossevine Artha Nathasya, Oscar Karnalim *, Mewati Ayub

Faculty of Information Technology, Maranatha Christian University, Indonesia

ARTICLE INFO

Article history:
Received 4 December 2018
Revised 3 April 2019
Accepted 13 May 2019
Available online 20 May 2019

Keywords:
Educational tool
Program visualisation
Algorithm visualisation
Data structure
Computer science education

ABSTRACT

Algorithm Visualisation (AV) tool is commonly used to learn data structures. However, since that tool does not address technical details, some students may not know how to implement the data structures. This paper integrates the AV tool with Program Visualisation (PV) tool to help the students understanding the data structures' implementation. The integration (which is implemented as a tool named DS-PITON) works similarly as a PV tool except that the data structures are visualised with the AV tool. Through quasi experiments, it can be stated that DS-PITON helps students to get better assessment score and to complete their assessment faster (even though the impact on completion time can work in reverse on slow-paced students). Further, according to a questionnaire survey, the students believe that DS-PITON helps them learning data structure materials.

1. Introduction

Student retention defines the success of universities [1]; higher retention leads to higher success rate. Consequently, several strategies are proposed to keep the retention high [2]. Some of them are: the use of persuasive social media [3], Student Success Course [4], and the integration of educational technologies [5].

To keep the student retention high in Computing education, educational technologies are often applied to help students learning a particular topic. These technologies typically rely on automated visualisation as their main feature; a particular topic is explained through intuitive graphics and animation. Program Visualisation (PV) tool [6] and Algorithm Visualisation (AV) tool [7] are two of the most common ones. The former focuses more on visualising how a particular program works while the latter focuses on visualising how algorithms and data structures work.

When learning data structures, most AV tools explain them without providing the technical details about how they are represented and behave in a real program. As a result, the students may know how the data structures theoretically work but may not be able to use them for solving a programming task. In other words, it enlarges the gap between students' theoretical and practical ability, which affects the students' problem solving skill (a part of employability attributes for university graduates [8]).

One of the possible solution to mitigate the gap is to let the students learn the technical details with a PV tool. However, the PV tool's visualisation can be challenging to understand since the data structures are treated like standard objects. The visualisation may not suit the structures' theoretical visualisation and will be mixed up with other in-program variables'.

To mitigate the gap between students' theoretical and practical ability related to data structures, this paper integrates PV with AV tools. This integration — represented as a tool named DS-PITON — works in similar manner as a PV tool except that some predefined data structures will have their own visualisation through the AV tool. In such a manner, the data structures' visualisation will be similar as their theoretical visualisation and displayed separately, which make them easy to understand at implementation level. To the best of our knowledge, no works have attempted to combine PV and AV tools for learning the implementation of data structures.

* Corresponding author at: Faculty of Information Technology, Maranatha Christian University, Prof. Drg. Surya Sumantri Street No.65, Bandung, West Java 40164, Indonesia.

E-mail addresses: oscar.karnalim@it.maranatha.edu (O. Karnalim), mewati.ayub@it.maranatha.edu (M. Ayub).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.eij.2019.05.001

Our proposed tool (DS-PITON) is developed on the top of PITON (a programming educational tool which was previously created in the same institution with two shared authors between them [9]). PITON was originally created to help students completing their programming laboratory assessment at introductory programming level. PITON can act as a PV tool in addition to a standard programming workspace. In such a manner, when the students get confused about how their code works, they can activate the visualisation to get better comprehension. It is true that PITON's goal is not related to DS-PITON; the former aims to help students learning how to code at introductory programming while the latter focuses more on learning how data structure implementation works. However, its PV tool mode can be used to achieve DS-PITON's goal. Currently, DS-PITON covers 7 data structures: array, linked list, stack with array, stack with linked list, queue with array, queue with linked list, and priority queue with linked list.

2. Related works

In Computing domain, educational tools are common to explain programs, algorithms, and data structures through visualisation. Generally speaking, these tools can be classified to three categories: Program Visualisation, Algorithm Visualisation, and others. This section will discuss educational tools for each of those categories. Further, we will also explain a brief overview of PITON, an educational tool which acts as a basis of our proposed tool.

2.1. Program visualisation tools

Program Visualisation (PV) tools help learners to learn how a particular program works. Each of them visualises all variables and execution states of the program while running [6]. Jeliot 3 [10] is a matured example of it. This tool is a Java-targeted PV tool that has been evolved several times from Elliot, Jelliot 1, and Jeliot 2000 [11,12].

Since programming language varies, some PV tools facilitate the integration of new target programming languages. That integration can be through either a mapping between the new languages and known language [13] or an embedding mechanism which requires some instructions written on the new languages [14].

Besides language-independence, some unique features are also proposed on existing PV tools. CodeChella [15] supports real-time tutoring and collaborative learning. JavlinaCode [16] displays a Unified Modeling Language's class diagram in addition to its standard visualisation. OmniCode [17] introduces a live programming mechanism. PITON [9] has an IDE-like programming workspace to assist learners writing their own program. PlayVisualizerC [18] handles capability, installability, and usability issues on existing PV tools. PythonTutor [19] covers various programming languages with an online architecture. SeeC [20] has human-language explanation assisting its visualisation.

2.2. Algorithm visualisation tools

Algorithm Visualisation (AV) tools help learners to learn how algorithms and data structures work [7]. They commonly cover basic algorithms and data structures such as sequential data structures (e.g., array, stack, queue, and linked list) [21,22] and searching & sorting algorithms [23–25].

Considering some learners face difficulties in advanced topics, several AV tools cover complex algorithms and data structures. Some of these topics are recursion, [26,27], strategic algorithms (e.g., backtracking, greedy algorithm, and dynamic programming) [28,29,22], and graph-related algorithms (e.g., Dijkstra's algorithm, and convex hull) [30].

Apart from classical topics, some AV tools cover domain-specific algorithms and data structures. For example, a work in [31] covers algorithms for network optimization problems. Other two examples are a work in [32] that covers SHA-512 algorithm and a work in [33] that covers matrix multiplication algorithm.

As the number of AV tools is increased, AlgoViz [34] was proposed. It works as a digital repository for AV tools where AV creators and users meet. On there, the success of an AV is measurable as the users can provide a feedback about it.

2.3. Other educational tools

For some learners, learning programming is not an easy task. Hence, Visual Programming (VP) tools are introduced as an alternative of Program Visualisation (PV) tools. Instead of writing a program code directly, a VP tool acts as a connector between the learners and their code. It removes some technical details so that the learners can focus on the algorithmic side of their program. Greenfoot [35] enables drag-and-drop feature for some program parts. Alice [36] and Scratch [37] let the learners to drag and drop their syntaxes instead of writing them directly. RAPTOR [38] and SFC Editor [39] display a flowchart to learners instead of program code.

VP and PV tools are not the only ones to help slow-paced learners in programming. Verificator [40], for example, is an educational tool which is also focused on such direction. It utilises a kind of traffic light system that limits the number of modifications applied on learner's program code. Upon reaching a limit, the learner should compile their code first before they can make further modification. This mechanism is expected to mitigate the number of displayed errors.

Some educational tools aim to explain the characteristics of algorithms. A work in [41] proposes Complexitor, a tool to learn algorithm time complexity in practical manner. It then inspires a work in [42] that proposes JCEL, which is similar to Complexitor except that it has simpler inputs and focuses on Java programming language. A work in [43] proposes GreedEx, a tool for learning the characteristics of greedy algorithms. It is then expanded to GreedExCol [44] with collaborative features on board.

2.4. PITON – Python integrated workspace and visualization

PITON (Python InTegrated wOrkspace and visualizatioN) [9] is a programming educational tool which combines programming workspace with Program Visualisation (PV) tool. Consequently, this tool does not only enable the direct development of source code but also the visualisation of the code. PITON is aimed to assist undergraduate students for completing their Python programming assessment in the introductory programming course. When the students are given an assessment, they are required to complete it using PITON. They should write the source codes on PITON and submit the resulted project.

During the assessment, if the student wants to check the correctness of their source code, they can execute the code through one of three modes. The first one is standard compile & run that works similarly like most programming workspaces; It displays the code's program outputs which can be as a response of the code's program inputs. The other two are related to visualisation, and commonly used when the student needs to understand their own code further. Step-by-step visualisation is an execution mode where a program created by the code is visualised and the student should press next or previous button to control the visualisation. Time-based visualisation is quite similar to the step-by-step visualisation except that the animation will be automatically updated instead of relying on user interaction.

Fig. 1 depicts the layout of PITON [9]. It consists of 7 panels:

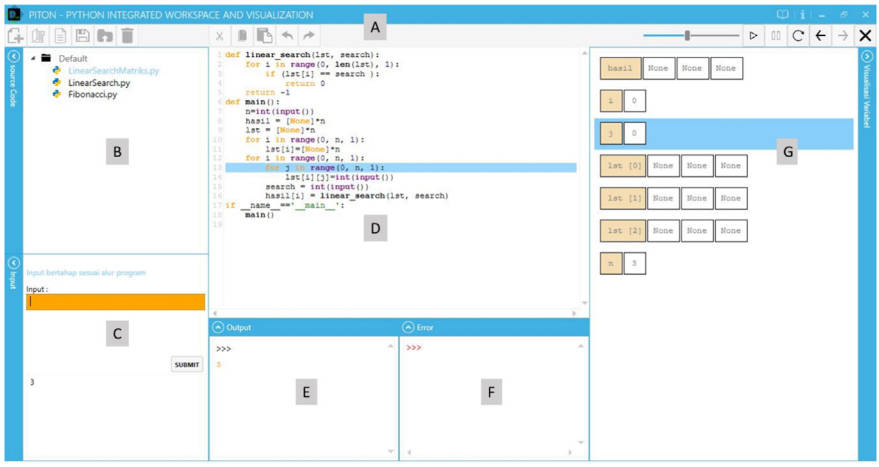

Fig. 1. The layout of PITON [9].

- Command toolbar (A), a panel to navigate the visualisation and manage source code projects.
- Working directory observer (B), a panel displaying source code projects.
- Input panel (C), a panel to provide inputs for a particular program execution.
- Source code editor (D), a panel where the student can write their source code for a programming assessment.
- Output panel (E), a panel to display the outputs of a particular program execution.
- Error panel (F), a panel to display the errors of a particular program execution.
- Variable content display panel (G), a panel for showing the content of all variables during the visualisation.

3. The tool: DS-PITON

When learning data structures, students are required to understand the data structures' implementation. In order to do that, these students can utilise a Program Visualisation (PV) tool; they can get the implementation (written in a particular programming language) from external resources, and feed it to the tool for visualisation. Nevertheless, existing PV tools' visualisation can be hard to understand; the data structures are treated like standard objects. The visualisation may not suit the data structures' theoretical visualisation. Further, in terms of presentation, it will be mixed up with other in-program variables'.

As a solution, this paper integrates PV with AV tools; the integration works similar as a standard PV tool except that, when a data structure is being visualised, its visualisation is handled by the AV tool. Consequently, the data structure will be visualised at algorithmic level, which can be easier to understand. Further, its visual will be separated from the standard variables and functions, which can provide more clarity. To our knowledge, this is the first attempt to combine PV and AV tools for learning data structure implementation.

3.1. Main architecture

The proposed combination between PV and AV tools is called DS-PITON. It is built on the top of PITON [9], and covers seven data structures for visualisation (array, linked list, stack with array, stack with linked list, queue with array, queue with linked list, and priority queue with linked list). Since PITON is developed for Python programming language, DS-PITON will only cover that programming language.

Fig. 2 shows the layout of DS-PITON. Since it is derived from PITON, the layout is quite similar to Fig. 1 except the right-bottom panel. That panel will display the data structures' visualisation. It is called data structure display.

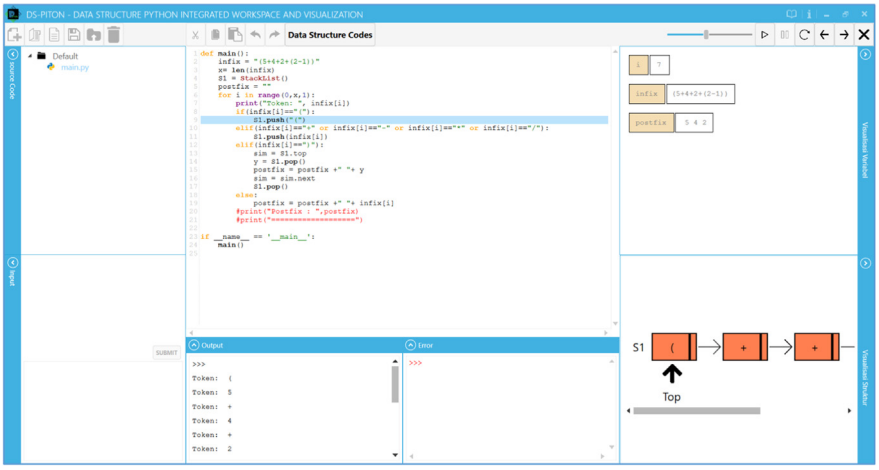

Fig. 2. The Layout of DS-PITON.

If the student wants to learn the implementation of a particular data structure, they can simply use the data structure on their code (by creating an object). They are not required to copy and paste the data structure's implementation on their code as it has been embedded on DS-PITON.

Prior starting the visualisation (either through PITON's step-by-step or time-based visualisation), the student can choose whether they want to see in-data-structure visualisation by ticking a checkbox. If that checkbox is checked, each time a data structure's method is invoked (see Table 1 for covered data structure methods), it will pop up a window visualising how that method works before continuing to the next instruction. An example of that window can be seen on Fig. 3. It contains three panels called source code editor, variable content display panel, and data structure display. They are referred as A-C respectively on that figure. Source code editor will display the method's code (written in Python). Variable content display will show all local variables involved on that method's execution. Data structure display will show the method's data structure's condition. It is important to note that this in-data-structure visualisation will be based on the invoked method's parameters.

Table 1
Covered Methods for Built-In Data Structures.

Data Structure	Covered Methods
Array	initialise, insert, remove, isEmpty, isFull, traversal, countElement, and search
Linked List	initialise, insertFirst, removeFirst, insertLast, removeLast, isEmpty, traversal, countElement, and search
Stack with Array	initialise, push, pop, peek, isEmpty, isFull, traversal, countElement, and search
Stack with Linked List	initialise, push, pop, peek, isEmpty, traversal, countElement, and search
Queue with Array	initialise, enqueue, dequeue, isEmpty, isFull, traversal, countElement, and search
Queue with Linked List	initialise, enqueue, dequeue, isEmpty, traversal, countElement, and search
Priority Queue with Linked List	initialise, enqueue, dequeue, isEmpty, traversal, countElement and search

For further understanding, DS-PITON enables multiple data structures to be shown at once (see Fig. 4). These structures are differentiated based on their variable name.

In terms of data structure visualisation, DS-PITON has three kinds of visualisation: array-based, list-based, and priority queue representation. Array-based representation is used to visualise array, stack with array, and queue with array. An example of it can be seen on Fig. 5. List-based representation is used to visualise linked list, stack with linked list, and queue with linked list. Its example can be seen on Fig. 6. Priority queue representation is used to visualise priority queue with linked list. It is similar to list-based representation except that its elements has three components: priority value, content, and next element's reference. Fig. 7 shows an example of it.

When the student wants to see the full implementation of our built-in data structures, they can click a button called "Data Structure Codes" placed at the top of source code editor (see Fig. 2). When that button is clicked, a pop-up window (as seen in Fig. 8) will be displayed. The student can select which data structure's implementation they want to see on the provided combobox. The code editor below that combobox will then display the implementation of that data structure. Unique to this feature, the implementation will be written with declarative comments so that the student can learn it comprehensively.

3.2. Functionality evaluation

The functionalities of DS-PITON were evaluated in threefold. The first one was a black-box testing conducted by the first author of this paper. The second one was an usability testing with five teaching assistants. The third one was an analysis of processing time by the second author of this paper.

Black-box testing was conducted by performing 15 scenarios related to the functionalities of DS-PITON. According to that testing, all features work correctly and no bugs are found.

Usability testing was conducted by asking five teaching assistants to complete two data structure assessments with DS-PITON. The former assessment is related to linked list while

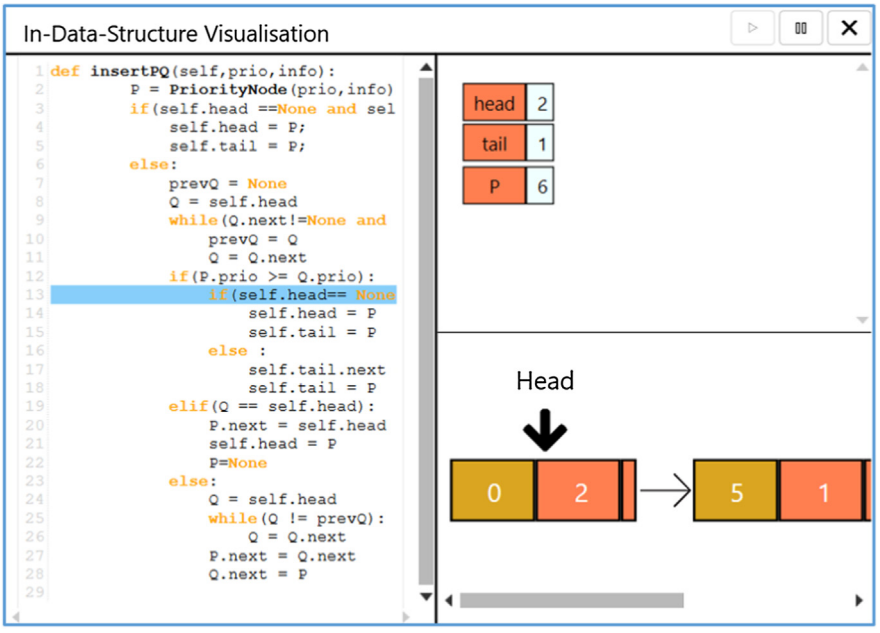

Fig. 3. An Example of Pop-Up Windows.

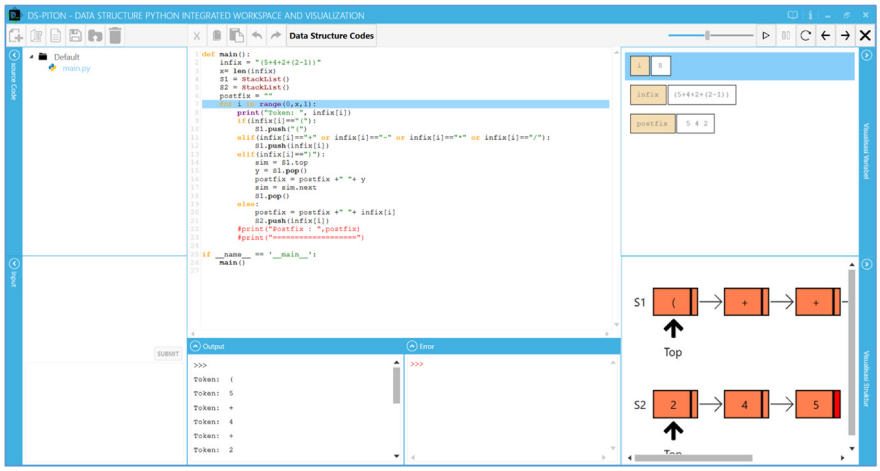

Fig. 4. An Example of Visualising Multiple Data Structures.

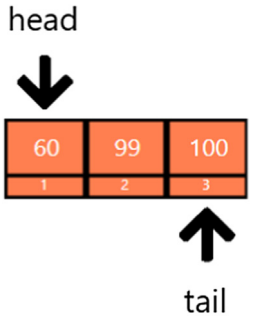

Fig. 5. An Example of Array-Based Representation: A Queue with Array.

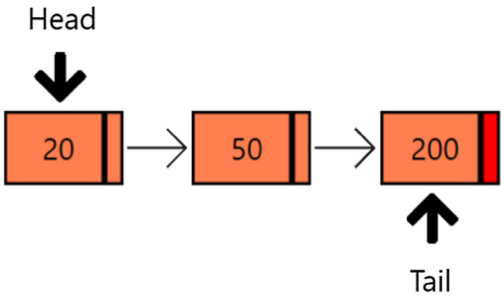

Fig. 6. An Example of List-Based Representation: A Queue with Linked List.

the latter is related to stack with array. To align with the goal of the tool (which is helping students for learning data structure implementation), each assessment has two types of questions, represented as 4 questions each; both types are about the behaviour of a given data structure in a program. The first question type asks the assistants to predict a data structure condition on a particular program state. Whereas, the second type asks the assistants to rearrange a list of data structure conditions under a sequence of program instructions. In addition to completing the assessment, they are also required to act as their students to find potential bugs and provide some feedback (if any).

According to our evaluation, all DS-PITON's functionalities work as expected. The assistants could complete both assessment tasks without finding any difficulties, even though they acted as their students. In fact, there was a bug found when a shortcut to close a window (*alt + F4*) had been still able to close the data structure visualisation window. However, that bug has been fixed upon the usability testing.

In terms of provided feedback, they can be categorised to three categories. The first one is to assure that a pop-up window is always displayed on top of the main window. It should always be the main focus till it is closed. The second one is to enlarge data structure display. The third one is to optimise the technical details of visualisation. All of them have been fulfilled at the final implementation of DS-PITON (that is proposed in this paper).

Considering DS-PITON is a combination between PV and AV tools, it is expected that the visualisation takes more processing time when compared to the standard PV or AV tool. In DS-PITON, several PV tool's outputs are passed to the AV tool for further processing. Regardless of the used data structure, DS-PITON's visualisation can be up to 100% slower than the standard tools, assuming that all PV tool's outputs are passed to the AV tool. For example, if a typical PV or AV tool requires 0.1 s to visualise a linked list, DS-PITON may require 0.2 s to do that. Nevertheless, according to our experience using the tool, that up to doubled processing time is not an issue since the students and authors do not experience any significant delay during visualisation. The real time required for visualising covered data structures is typically small since the

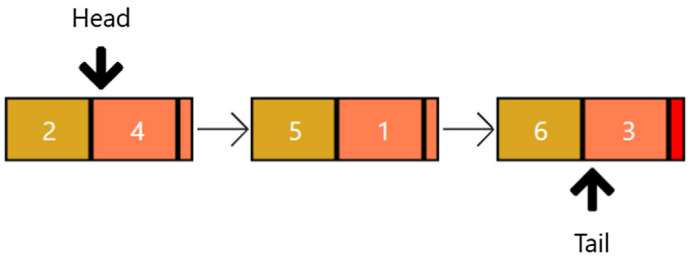

Fig. 7. An Example of Priority Queue Representation.

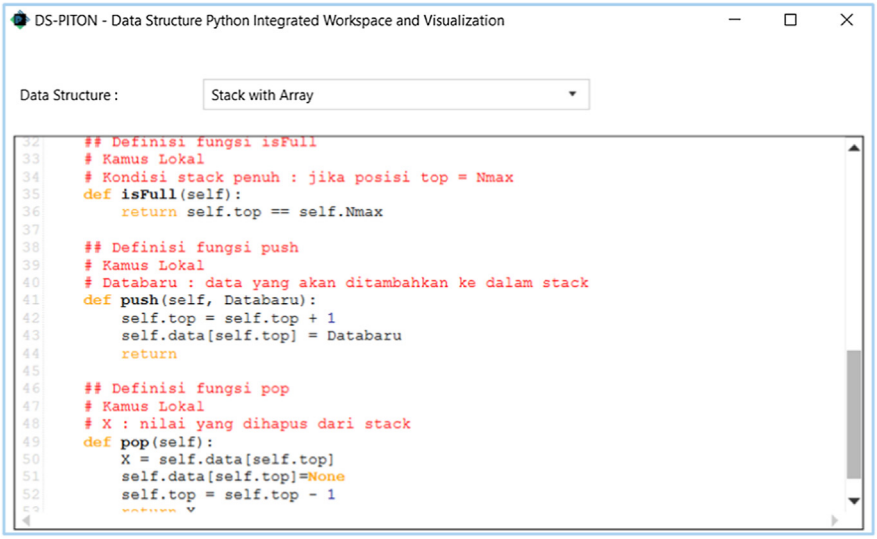

Fig. 8. A Pop-Up Window Displaying Data Structure Codes.

tool is used in academic environment where the amount of visualised information is limited.

4. The learning methodology

DS-PITON can be used to learn data structure implementation in both supervised and unsupervised manner. Supervised learning means at least one lecturer or tutor accompanies the students in the process. This kind of learning commonly happens in a class session where the students are asked to learn about the implementation of some data structures. Prior using the tool, the students are informed about how to use DS-PITON and what goals to achieve (e.g., understanding how a linked list is implemented). Afterwards, the students will use the tool by writing a piece of Python source code which main focus is to see how some data structures' implementation looks like and behaves.

If the students have no access to computers, DS-PITON can be displayed on a computer operated by the lecturer or tutor, and therefore used as a supportive resource for teaching session. The use of DS-PITON in this mode is quite similar to the use of a programming workspace while teaching programming; the lecturer or tutor uses it to show that a particular code (which aims to explore the characteristics of data structure implementation in our case) works as expected.

Unsupervised learning, on the contrary, relies on no lecturer or tutor during the process. It typically occurs when the students are taking an online course or completing an assessment related to data structure implementation. When this kind of learning is used, it is important to make the tool accessible through e-learning so that when they need it, they can easily download it. Further, the detailed tutorial about how to use the tool and what goals need to be achieved with that tool should also be described on the e-learning. In terms of usage, the students can utilise the tool in a similar manner as supervised learning with computer access; they can write a Python source code aiming to learn the implementation of some data structures.

5. Measuring the impact of learning with DS-PITON

The impact of DS-PITON for learning data structure implementation was measured by comparing it with the textbook-based learning (where the student learn a particular information from a given textbook). The textbook-based learning is chosen as the baseline since it is the conventional (and most common) strategy for learning data structure implementation. By comparing those two, we believe that our findings can be more relevant to the current condition of learning data structure implementation.

The impact of DS-PITON is not compared with the impact of its predecessor (PITON) due to two reasons. Firstly, the conventional strategy for learning data structure implementation is based on the textbook instead of an educational tool like PITON. Secondly, DS-PITON and PITON are not comparable when used for learning data structure implementation. The latter is not specifically designed for that task. It is focused on teaching introductory programming instead of data structures. Forcing it will benefit the former; PITON does not group the data structures' attributes and does not simplify their visualisation. Further, since PITON is not embedded with the source codes of data structures, it requires the student to embed that codes on their own code each time they want to visualise these structures, which can be quite demanding.

The comparison between DS-PITON-based and textbook-based learning relied on quasi-experiments [45] with two aspects on board: score and time outcome. It involved two groups of students: moderate-paced and slow-paced students. The first group contained 15 moderate students (S1–S15). They had completed Basic Algorithm and Data Structure (i.e., a course in our faculty which covers DS-PITON's built-in data structures) with a grade higher or equal to C in their previous semesters. The second group contained 15 slow-paced students (S16–S30). When they participated on the experiment, they were still completing Basic Algorithm and Data Structure course and their mid-test score on that course is lower than 55 (a minimum threshold to get a C grade in our faculty). It is worth to note that the experiment of slow-paced students were performed twice due to the limited number of participating students at the first attempt. The first experiment only involved five students (S16–S20) while the latter experiment involved the rest (S21–S30).

Two quasi-experiments were conducted on moderate-paced students. For each experiment, the students should complete two data structure assessments about the same topic (either priority queue with linked list or queue with array). These assessments are similar in terms of the number of questions (which is eight) and difficulty level. DS-PITON aims to help learners for learning some data structure implementations. Hence, these assessments'

questions are about the behaviour of a given data structure in a program. The questions ask the learners to either predict a data structure condition on a particular program state or rearrange a list of data structure conditions under a sequence of program instructions.

Per experiment, the students would act as a control group when completing the first assessment and as an intervened group when completing the second assessment. Both sessions would be conducted in 30 min each. For the first assessment, the students should rely only on a data structure textbook (textbook-based learning). Whereas, for the second assessment, they should rely only on DS-PITON. The use of DS-PITON is beneficial if a statistically-significant improvement occurs between the control and intervened groups in terms of scores and/or completion time (measured using a two-tailed paired t-test).

For slow-paced students, three quasi-experiments were conducted. The experiments behaved similarly as the experiments on moderate-paced students except that they cover different material set. In this context, these experiments cover priority queue with linked list, queue with linked list, and stack with array. Further, our experiments have fewer questions for assessments (three questions per assessment with 33.33% score contribution per question). Each experiment should be completed in 30 min (15 min per assessment). It is worth to note that the number of questions and completion time are modified so that slow-paced students would not be feel burdened.

To gather student perspectives, a questionnaire survey involving 20 students (S1-S20 from the quasi experiments) was conducted. To mitigate bias, the students were asked to answer the survey right after they had used DS-PITON (in our case, after they had participated on the quasi experiments).

Our survey contains eleven rating questions and one open-ended question. Rating questions ask the students' agreement toward some statements where their agreement is represented as 5-point Likert scale (1 = strongly disagree, 2 = disagree, 3 = neutral, 4 = agree, and 5 = strongly agree). Open-ended question asks the students' feedback toward DS-PITON; they should answer it with free-formed sentences.

Table 2 shows the details of our rating questions. In general, these questions ask whether DS-PITON helps the students in terms of understanding how a given program works and the behaviour of a particular data structure. It is important to note that the last three questions compare DS-PITON-based with textbook-based learning.

Table 2
Rating Questions.

series ID	series Survey Statement
Q1	In-data-structure visualisation helps me to understand how a given program works.
Q2	In-data-structure visualisation helps me to understand the behaviour of a particular data structure.
Q3	Data structure display helps me to understand how a given program works.
Q4	Data structure display helps me to understand the behaviour of a particular data structure.
Q5	Variable content display panel on in-data-structure visualisation helps me to understand how a given program works.
Q6	Variable content display panel on in-data-structure visualisation helps me to understand the behaviour of a particular data structure.
Q7	A Combination of PV and AV helps me to understand how a given program works.
Q8	A Combination of PV and AV helps me to understand the behaviour of a particular data structure.
Q9	Compared to learning from a data structure textbook, learning with DS-PITON is more effective for understanding how a given program works.
Q10	Compared to learning from a data structure textbook, learning with DS-PITON is more effective for understanding the behaviour of a particular data structure.
Q11	Compared to learning from a data structure textbook, learning with DS-PITON is more time-efficient for understanding data structure materials.

5.1. Score outcome results for moderate-paced students

Fig. 9 shows that, for moderate-paced students, DS-PITON is more helpful than a data structure textbook to learn priority queue with linked list. Two thirds of the students achieved higher score when DS-PITON is on board. Further, in average, the score of DS-PITON-based learning (80.833 with 16.275 as its standard deviation) is higher than textbook-based learning (65.833 with 17.970 as its standard deviation). When measured using a paired t-test, its improvement is statistically significant since its p-value (0.0363) is lower than the maximum threshold for significance (0.05).

When queue with array is used as the material, DS-PITON shows more significant improvement (see Fig. 10). With DS-PITON, most students (12 of 15) achieved higher score and no students achieved lower score. This finding is strengthened by the fact that its improvement (31.666, which is resulted by subtracting the average score of DS-PITON-based with textbook-based learning) is higher than such improvement on priority queue material (15). It is important to note that the improvement on queue with array material is also statistically significant since its t-test's p-value is 0.002.

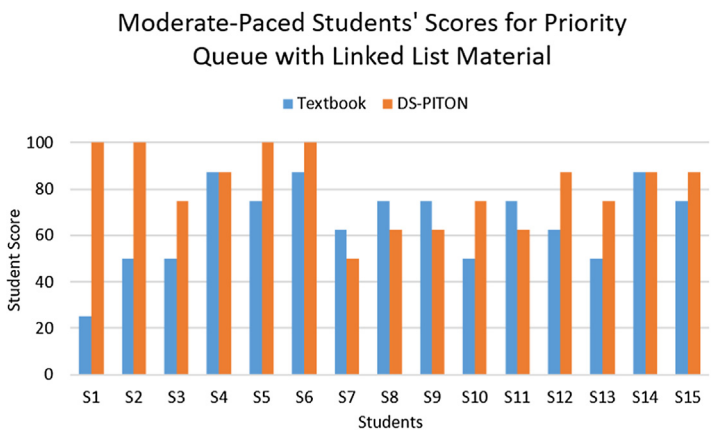

Fig. 9. Moderate-Paced Students' Scores for Priority Queue with Linked List Material.

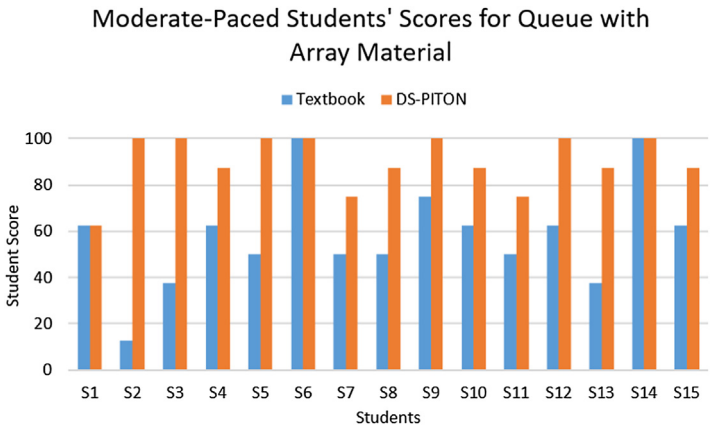

Fig. 10. Moderate-Paced Students' Scores for Queue with Array Material.

5.2. Score outcome results for slow-paced students

Fig. 11 depicts that, for slow-paced students, DS-PITON affects positively for learning priority queue with linked list. In average, DS-PITON-based learning yields 44.44 higher score. Its average score is 84.44 with 21.33 as its standard deviation. Whereas, textbook-based learning only has 40 as its average score with 18.69 as its standard deviation. When measured using t-test, its improvement is statistically significant since its p-value (0.00005) is lower than the maximum threshold for significance.

On queue with linked list material (see Fig. 12), DS-PITON also affects positively. The average score of DS-PITON-based learning (95.56 with 11.73 as its standard deviation) is 35.56 points higher than the average score of textbook-based learning (60 with 33.81 as its standard deviation). Further, DS-PITON's improvement is statistically significant (its p-value is 0.001).

Fig. 13 shows that DS-PITON's impact is also statistically significant on stack with array material (p-value = 0.043). DS-PITON-based learning (with 86.67 average score and 27.60 standard deviation) leads higher score than the textbook-based one (with 62.22 average score and 27.79 standard deviation).

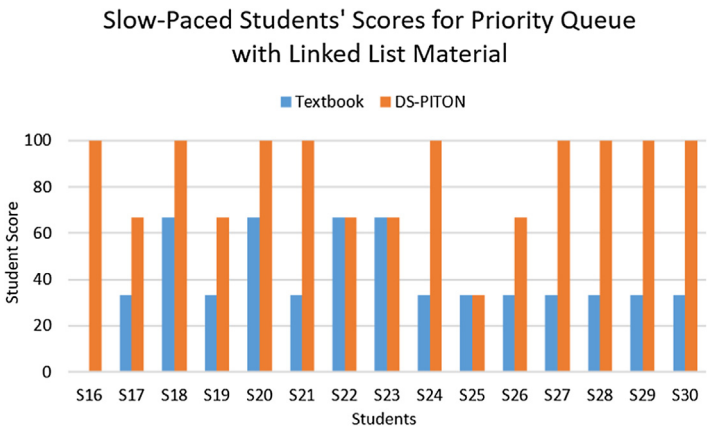

Fig. 11. Slow-Paced Students' Scores for Priority Queue with Linked List Material.

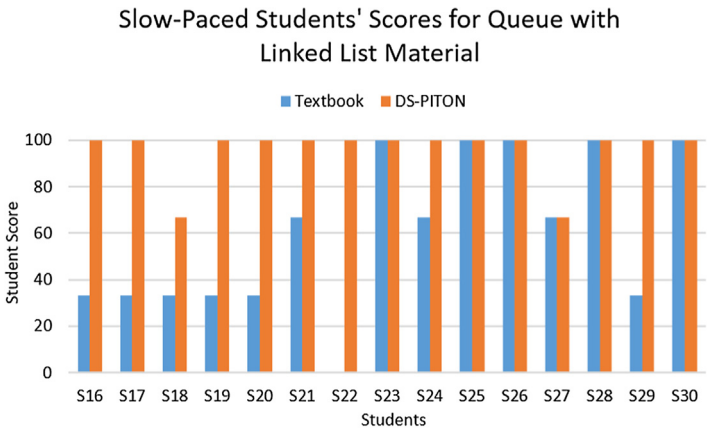

Fig. 12. Slow-Paced Students' Scores for Queue with Linked List Material.

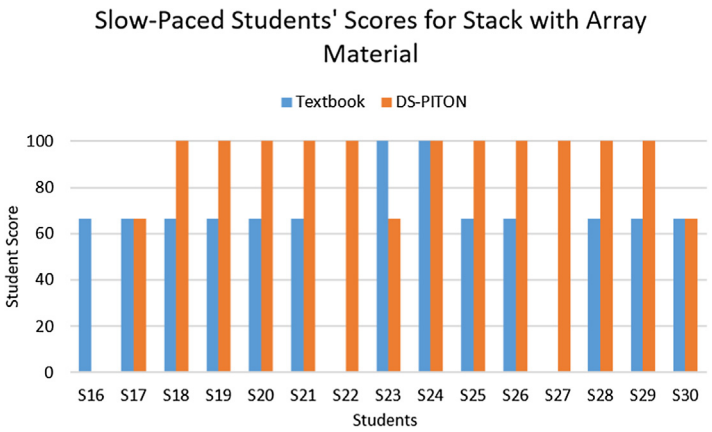

Fig. 13. Slow-Paced Students' Scores for Stack with Array Material.

5.3. Completion time results for moderate-paced students

Fig. 14 shows that, in terms of completing assessment about priority queue with linked list, DS-PITON is more time-efficient than a data structure textbook for moderate-paced students. Most of the students (11 of 15) completed that assessment faster, which is 141.33 s faster in average. DS-PITON-based learning yields 465 s in average with 129 s as its standard deviation. Textbook-based learning, on the other, yields 608 s in average with 229 s as its standard deviation. This finding is strengthened by the fact that the change is statistically significant, as its p-value (0.0214) is lower than 0.05 (a maximum threshold for significance).

For assessment about queue with array, DS-PITON is still more time-efficient than a data structure textbook. As seen on Fig. 15, eleven students completed the assessment faster with DS-PITON. In average, DS-PITON's scenario's completion time is 202.67 s faster than the textbook's scenario's completion time. It takes 585 s in average with 256 s standard deviation. Whereas, textbook scenario takes 746 s in average with 193 s standard deviation. According to t-test, that change is statistically significant (its p-value is 0.0115).

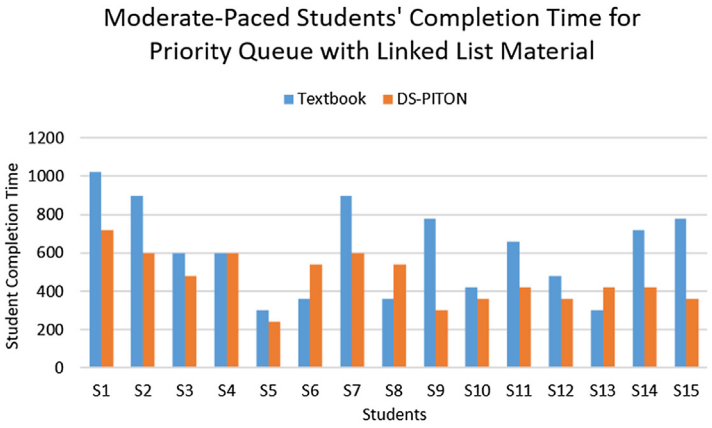

Fig. 14. Moderate-Paced Students' Completion Time for Priority Queue Material.

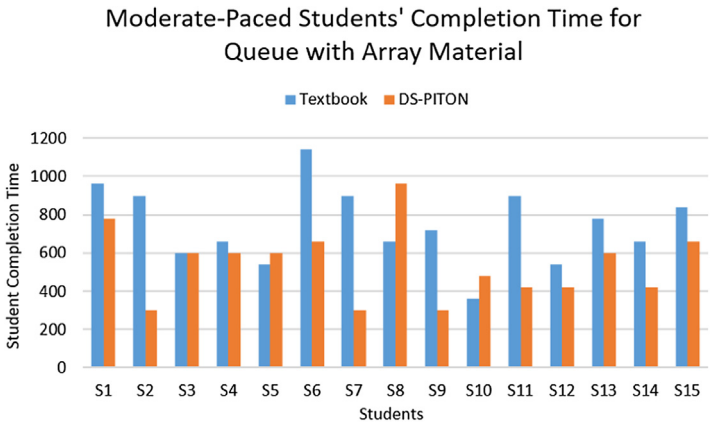

Fig. 15. Moderate-Paced Students' Completion Time for Queue with Array Material.

5.4. Completion time results for slow-paced students

According to our experiments, DS-PITON does not reduce the completion time of slow-paced students, and it is possible to affect in reverse. This statement is strengthened by the fact that two experiments (which are about priority queue and queue with linked list on Fig. 16 and Fig. 17 respectively) show no statistically significant reduction (their p-values are 0.186 and 0.357 respectively), and one experiment (which is about stack with array on Fig. 18) shows that DS-PITON increases completion time in a statistically significant manner (with p-value = 0.007). One of the possible reasons is the limited adaptability of the slow-paced students; they experienced some difficulties while operating the tool.

5.5. Questionnaire survey results

Fig. 19 shows the average scores for the rating questions. All questions were responded positively; their average score is higher than 4 (which represents agree). Among these questions, Q1 and Q10 achieve the highest average score while Q4 and Q9 achieve the lowest. However, since their scores are slightly different to each other, no additional findings can be obtained.

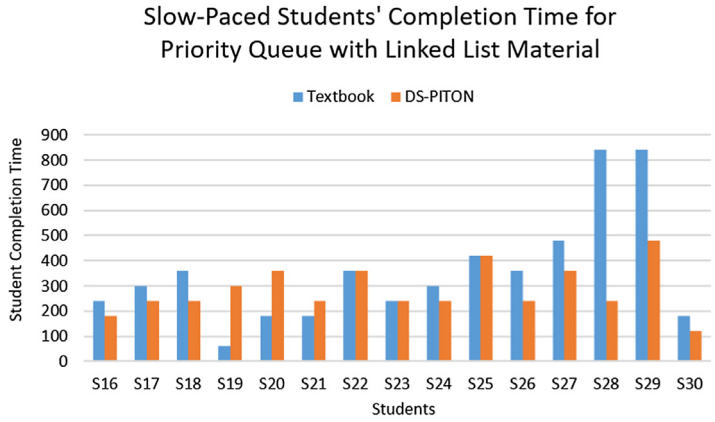

Fig. 16. Slow-Paced Students' Completion Time for Priority Queue with Linked List Material.

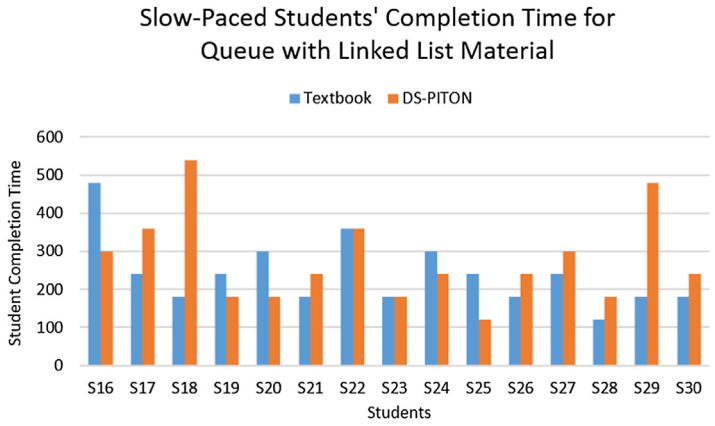

Fig. 17. Slow-Paced Students' Completion Time for Queue with Linked List Material.

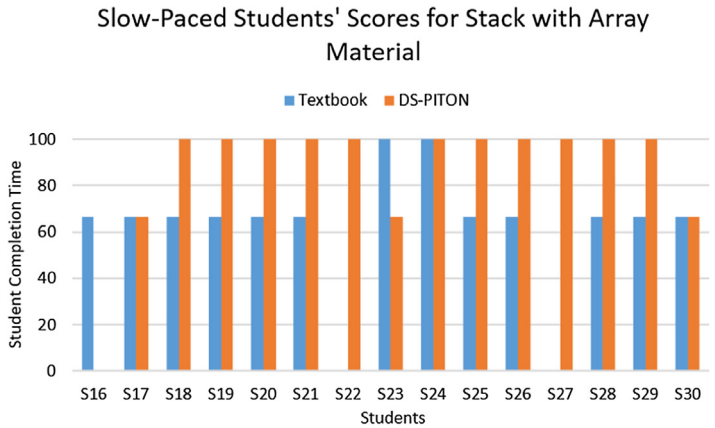

Fig. 18. Slow-Paced Students' Completion Time for Stack with Array.

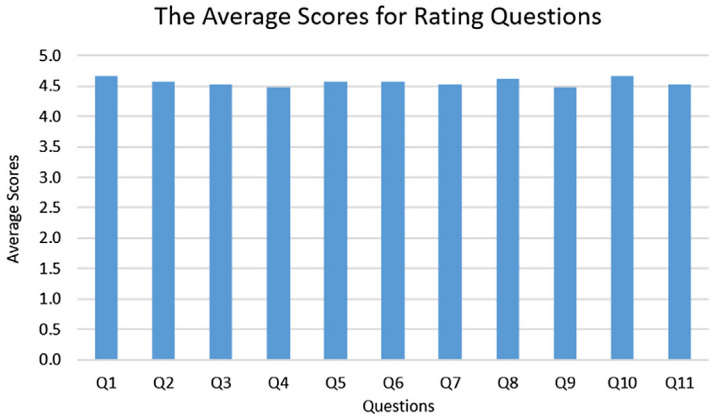

Fig. 19. The Average Scores for Rating Questions.

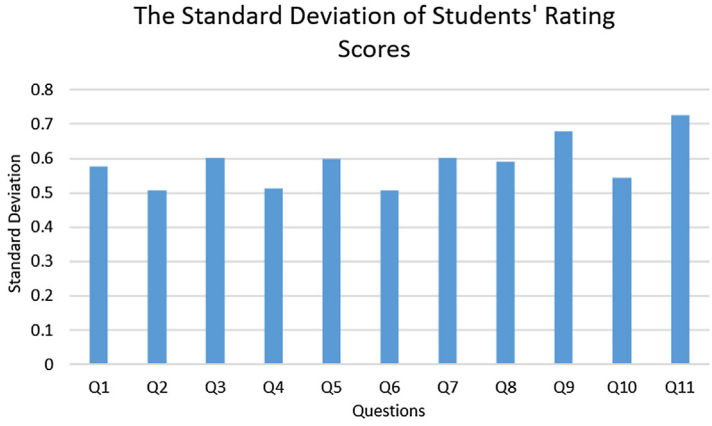

Fig. 20. The Standard Deviation of Students' Rating Scores.

As seen in Fig. 20, all rating questions have a low degree of variations; their standard deviation is between 0.5 and 0.75. The highest degree of variation occurs Q11 while the lowest one occurs on Q2 and Q6. Considering the difference between the highest and the lowest degree is low, no additional findings can be concluded.

The open-ended question collects two kinds of feedbacks. The first one is to optimise DS-PITON more as some laboratory computers have limited specification. The second one is to integrate DS-PITON with the data structure textbook (as inspired by [46]).

6. Conclusion and future work

This paper integrates Program Visualisation (PV) with Algorithm Visualisation (AV) tools with the aim to help learners understanding data structure implementation. The integration is represented as a tool called DS-PITON. It works similar to a standard PV tool except that, when visualising predefined data structures, it utilises an AV tool for visualisation. According to our evaluation, three findings can be deducted. First, it helps both moderate-paced and slow-paced students getting better assessment score. Second, it helps moderate-paced students completing their assessment faster while being possible to add more completion time for slow-paced students. Third, the students believe that DS-PITON helps them to understand data structure materials.

For future works, we plan to fulfill the students' feedback from our survey. We plan to reevaluate DS-PITON' implementation comprehensively and optimise some components if possible. Further, we plan to integrate a data structure textbook to DS-PITON.

Acknowledgement

This work was supported by Maranatha Christian University, Indonesia.

References

[1] Wild L, Ebbers L. Rethinking student retention in community colleges. Community College J Res Practice 2002;26(6):503–19. https://doi.org/10.1080/2776770290041864. URL: https://www.tandfonline.com/doi/full/10.1080/2776770290041864.

[2] Hone KS, El Said GR. Exploring the factors affecting MOOC retention: a survey study. Comput Educ 2016;98:157–68. https://doi.org/10.1016/J.COMPEDU.2016.03.016. URL: https://www.sciencedirect.com/science/article/pii/S0360131516300793.

[3] Zheng S, Han K, Rosson MB, Carroll JM. The Role of Social Media in MOOCs: How to Use Social Media to Enhance Student Retention. In: Proceedings of the Third (2016) ACM Conference on Learning @ Scale - L@S '16. New York, New York, USA: ACM Press; 2016. p. 419–28. https://doi.org/10.1145/2876034.2876047. URL: http://dl.acm.org/citation.cfm?doid=2876034.2876047.

[4] Kimbark K, Peters ML, Richardson T. Effectiveness of the student success course on persistence, retention, academic achievement, and student engagement. Community College J Res Practice 2017;41(2):124–38. https://doi.org/10.1080/10668926.2016.1166352. URL: https://www.tandfonline.com/doi/full/10.1080/10668926.2016.1166352.

[5] Truong HM. Integrating learning styles and adaptive e-learning system: Current developments, problems and opportunities. Comput Hum Behav 2016;55:1185–93. https://doi.org/10.1016/J.CHB.2015.02.014. URL: https://www.sciencedirect.com/science/article/pii/S0747563215001120.

[6] Urquiza-Fuentes J, Velázquez-Iturbide JÁ. A survey of successful evaluations of program visualization and algorithm animation systems. ACM Trans Comput Edu 2009;9(2):1–21. https://doi.org/10.1145/1538234.1538236. URL: http://portal.acm.org/citation.cfm?doid=1538234.1538236.

[7] Shaffer CA, Cooper ML, Alon AJD, Akbar M, Stewart M, Ponce S, Edwards SH. Algorithm visualization: the state of the field. ACM Trans Comput Edu 2010;10 (3):1–22. https://doi.org/10.1145/1821996.1821997. URL: http://portal.acm.org/citation.cfm?doid=1821996.1821997.

[8] Osmani M, Weerakkody V, Hindi NM, Al-Esmail R, Eldabi T, Kapoor K, Irani Z. Identifying the trends and impact of graduate attributes on employability: a literature review. Tert Educ Manag 2015;21(4):367–79. https://doi.org/10.1080/13583883.2015.1114139. URL: http://www.tandfonline.com/doi/full/10.1080/13583883.2015.1114139.

[9] Elvina E, Karnalim O, Ayub M, Wijanto MC. Combining program visualization with programming workspace to assist students for completing programming laboratory task. J Technol Sci Edu 2018;8(4):268. https://doi.org/10.3926/jotse.420. URL: http://www.jotse.org/index.php/jotse/article/view/420.

[10] Moreno A, Myller N, Sutinen E, Ben-Ari M. Visualizing programs with Jeliot 3. In: Proceedings of the working conference on Advanced visual interfaces – AVI '04. New York, New York, USA: ACM Press; 2004. p. 373. https://doi.org/10.1145/989863.989928. URL: http://portal.acm.org/citation.cfm?doid=989863.989928.

[11] Ben-Ari M, Bednarik R, Ben-Bassat Levy R, Ebel G, Moreno A, Myller N, Sutinen E. A decade of research and development on program animation: the Jeliot experience. J Visual Lang Comput 2011;22(5):375–84. https://doi.org/10.1016/J.JVLC.2011.04.004. URL: https://www.sciencedirect.com/science/article/pii/S1045926X11000310.

[12] Ben-Bassat Levy R, Ben-Ari M, Uronen PA. The Jeliot 2000 program animation system. Comput Educ 2003;40(1):1–15. https://doi.org/10.1016/S0360-1315(02)00076-3. URL: https://www.sciencedirect.com/science/article/pii/S0360131502000763.

[13] Rajala T, Laakso M-J, Kalla E, Salakoski T. VILLE: a language-independent program visualization tool. Proceedings of the Seventh Baltic Sea Conference on Computing Education Research, vol. 88. Darlinghurst: Australian Computer Society; 2007. p. 151–9. URL: https://dl.acm.org/citation.cfm?id=2449340.

[14] Sulistiani L, Karnalim O. An embedding technique for language-independent lecturer-oriented program visualization. EMITTER Int J Eng Technol 2018;6 (1):92–104. https://doi.org/10.24003/emitter.v6i1.234. URL: http://emitter.pens.ac.id/index.php/emitter/article/view/234.

[15] Guo PJ, White J, Zanelatto R. Codechella: Multi-user program visualizations for real-time tutoring and collaborative learning. 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE; 2015. p. 79–87. https://doi.org/10.1109/VLHCC.2015.7357201. URL: http://ieeexplore.ieee.org/document/7357201/.

[16] Yang J, Lee Y, Hicks D, Chang KH. Enhancing object-oriented programming education using static and dynamic visualization. 2015 IEEE Frontiers in Education Conference (FIE). IEEE; 2015. p. 1–5. https://doi.org/10.1109/FIE.2015.7344152. URL: http://ieeexplore.ieee.org/document/7344152/.

[17] Kang H, Guo PJ. Omnicode: a novice-oriented live programming environment with always-on run-time value visualizations. In: The 30th ACM Symposium on User Interface Software and Technology.

[18] Ishizue R, Sakamoto K, Washizaki H, Fukazawa Y. PVC: Visualizing C Programs on Web Browsers for Novices. Proceedings of the 49th ACM Technical Symposium on Computer Science Education – SIGCSE '18. New York, New York, USA: ACM Press; 2018. p. 245–50. https://doi.org/10.1145/3159450.3159566. URL: http://dl.acm.org/citation.cfm?doid=3159450.3159566.

[19] Guo PJ. Online python tutor: embeddable web-based program visualization for cs education. Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13. New York, New York, USA: ACM Press; 2013. p. 579. https://doi.org/10.1145/2445196.2445368. URL: http://dl.acm.org/citation.cfm?doid=2445196.2445368.

[20] Egan MH, McDonald C. Program visualization and explanation for novice C programmers. In: Proceedings of the Sixteenth Australasian Computing Education Conference – Volume 148. Auckland: Australian Computer Society Inc; 2014. p. 51–7.

[21] Christiawan L, Karnalim O, AP-ASD1: An Indonesian Desktop-based Educational Tool for Basic Data Structure Course, Jurnal Teknik Informatika dan Sistem Informasi 2 (1). http://jutisi.maranatha.edu/index.php/jutisi/article/view/422..

[22] Halim S, Chun KOH Z, Bo Huai LOH V, Halim F. Learning algorithms with unified and interactive web-based visualization. Olympiads Inform 2012;6:53–68. URL: https://www.mii.lt/olympiads_in_informatics/pdf/INFOL099.pdf.

[23] Vrachnos E, Jimoyiannis A. Design and evaluation of a web-based dynamic algorithm visualization environment for novices. Procedia Comput Sci 2014;27:229–39. https://doi.org/10.1016/J.PROCS.2014.02.026. URL: https://www.sciencedirect.com/science/article/pii/S1877050914000283.

[24] Avancena AT, Nishihara A, Kondo C. Developing an algorithm learning tool for high school introductory computer science. Educ Res Int 2015;2015:1–11. https://doi.org/10.1155/2015/840217. URL: http://www.hindawi.com/journals/edri/2015/840217/.

[25] Yohannis A, Prabowo Y. Sort Attack: Visualization and Gamification of Sorting Algorithm Learning. 2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games). IEEE; 2015. p. 1–8. https://doi.org/10.1109/VS-GAMES.2015.7295785. URL: http://ieeexplore.ieee.org/document/7295785/.

[26] Velázquez-Iturbide JÁ, Pérez-Carrasco A, Urquiza-Fuentes J. SRec:: an animation system of recursion for algorithm courses. Proceedings of the 13th annual conference on Innovation and technology in computer science education – ITiCSE '08, vol. 40. New York, New York, USA: ACM Press; 2008. p. 225. https://doi.org/10.1145/1384271.1384332. URL: http://portal.acm.org/citation.cfm?doid=1384271.1384332.

[27] Hamouda S, Edwards SH, Elmongui HG, Ernst JV, Shaffer CA. RecurTutor: An interactive tutorial for learning recursion. ACM Trans Comput Educ 2018;19 (1):1–25. https://doi.org/10.1145/3218328. URL: http://dl.acm.org/citation.cfm?doid=3282284.3218328.

[28] Jonathan FC, Karnalim O, Ayub M. Extending The Effectiveness of Algorithm Visualization with Performance Comparison through Evaluation-integrated Development, in: Seminar Nasional Aplikasi Teknologi Informasi (SNATI), 2016. http://www.jurnal.uii.ac.id/index.php/Snati/article/view/6263..

[29] Zumaytis S, Karnalim O. Introducing an educational tool for learning branch & bound strategy. J Inform Syst Eng Business Intell 2017;3(1):8. https://doi.org/10.20473/jisebi.3.1.8-15.

[30] Teresco JD, Fathi R, Ziarek L, Bamundo M, Pengu A, Tarbay CF. Map-based algorithm visualization with METAL highway data. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education – SIGCSE '18. New York, New York, USA: ACM Press; 2018. p. 550–5. https://doi.org/10.1145/3159450.3159583. URL: http://dl.acm.org/citation.cfm?doid=3159450.3159583.

[31] da Silva Lourenço W, de Araujo Lima SJ, Alves de Araújo S. TASNOP: a tool for teaching algorithms to solve network optimization problems. Comput Appl Eng Educ 2018;26(1):101–10. https://doi.org/10.1002/cae.21864. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/cae.21864.

[32] Ma J, Tao J, Keranen M, Mayo J, Shene C-K, Wang C. SHAvisual: a secure hash algorithm visualization tool. In: Proceedings of the 2014 conference on Innovation & technology in computer science education – ITiCSE '14. New York, New York, USA: ACM Press; 2014. https://doi.org/10.1145/2591708.2602663. pp. 338–338, URL: http://dl.acm.org/citation.cfm?doid=2591708.2602663.

[33] Desai S, Kulkarni S, Vasant Vaibhav M, Varalakshmi P Mohamed. VPMM visualization of parallel matrix multiplication algorithms. J Comput Sci Colleges 2002;33(1):24–31.

[34] Shaffer CA, Naps TL, Rodger SH, Edwards SH. Building an online educational community for algorithm visualization. Proceedings of the 41st ACM technical symposium on Computer science education – SIGCSE '10. New York, New York, USA: ACM Press; 2010. p. 475. https://doi.org/10.1145/1734263.1734421. URL: http://portal.acm.org/citation.cfm?doid=1734263.1734421.

[35] Kölling M. Michael, The Greenfoot Programming Environment. ACM Trans Comput Educ 2010;10(4):1–21. https://doi.org/10.1145/1868358.1868361. URL: http://portal.acm.org/citation.cfm?doid=1868358.1868361.

[36] Cooper S, Dann W, Pausch R. Alice: a 3-D tool for introductory programming concepts. J Comput Sci Colleges 2000;15(5):107–16. URL: https://dl.acm.org/citation.cfm?id=364161.

[37] Resnick M, Silverman B, Kafai Y, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, Millner A, Rosenbaum E, Silver J. Scratch: programming for all. Commun ACM 2009;52(11):60. https://doi.org/10.1145/1592761.1592779. URL: http://portal.acm.org/citation.cfm?doid=1592761.1592779.

[38] Carlisle MC, Wilson TA, Humphries JW, Hadfield SM. RAPTOR: a visual programming environment for teaching algorithmic problem solving. ACM SIGCSE Bull 2005;37(1):176–80. URL: https://dl.acm.org/citation.cfm?id=1047411.

[39] Watts T. The SFC editor a graphical tool for algorithm development. J Comput Sci Colleges 2004;20(2):73–85. URL: https://dl.acm.org/citation.cfm?id=1040159.

[40] Radošević D, Orehovački T, Lovrenčić A, Verificator: educational tool for learning programming, informatics in education 8 (2), 2009. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.148.9441.

[41] Elvina E, Karnalim O. Complexitor: an educational tool for learning algorithm time complexity in practical manner, ComTech: Computer. Math Eng Appl 2017;8(1):21. https://doi.org/10.21512/comtech.v8i1.3783. URL: http://journal.binus.ac.id/index.php/comtech/article/view/3783.

[42] Kurniawati G, Karnalim O. Introducing a practical educational tool for correlating algorithm time complexity with real program execution. J Inform Technol Comput Sci 2018;3(1):1. https://doi.org/10.25126/jitecs.20183140. URL: http://jitecs.ub.ac.id/index.php/jitecs/article/view/40.

[43] Velázquez-Iturbide JÁ, Pérez-Carrasco A. Active learning of greedy algorithms by means of interactive experimentation. Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education – ITiCSE '09, vol. 41. New York: ACM Press; 2009. p. 119. https://doi.org/10.1145/1562877.1562917.

[44] Debdi O, Paredes-Velasco M, Velázquez-Iturbide JÁ. GreedExCol, A CSCL tool for experimenting with greedy algorithms. Comput Applic Eng Educ 2015;23 (5):790–804. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/cae.21655.

[45] Creswell JW. Educational research: planning, conducting, and evaluating quantitative and qualitative research. Pearson; 2012.

[46] Fouh E, Karavirta V, Breakiron DA, Hamouda S, Hall S, Naps TL, Shaffer CA. Design and architecture of an interactive eTextbook – The OpenDSA system. Sci Comput Program 2014;88:22–40. https://doi.org/10.1016/J.SCICO.2013.11.040. URL: https://www.sciencedirect.com/science/article/pii/S016764231300333X.