ACTA DIDACTICA NAPOCENSIA
Volume 15, Number 1, 2022 - DOI: 10.24193/adn.15.1.5

DATA VISUALIZATION IN PROGRAMMING EDUCATION

Imre BENDE

Abstract: Understanding data structures is fundamental for mastering algorithms. In order to solve problems and tasks, students must be able to choose the most appropriate data structure in which the data is stored and that helps in the process of the solution. Of course, there is no single correct solution, but in many cases, it is an important step to find the most efficient data structure to solve the task. In this paper, I will present the most common data representations and data structures that can help get to know them, to master them, and then to reuse them for solving different tasks.

Key words: algorithm visualization, programming education, algorithm, data structure

1. Introduction

Understanding data structures is fundamental for mastering algorithms. In order to solve problems and tasks, students must be able to choose the most appropriate data structure in which the data is stored and that helps in the process of the solution. “We noticed that the students are afraid of the searcher algorithms as they find them too complicated, that exceeds their abilities. So it is very important to make these algorithms tangible and bring them close. One great method for this is the visualization of the algorithm” (Kovásznai et al., 2011). Of course, there is no single correct solution, but in many cases, it is an important step to find the most efficient data structure to solve the task (runtime / memory requirements, solution complexity can be reduced by a well-chosen data structure). It is important to state that teaching the algorithms and data structures cannot work separately, as they must be reviewed by connecting and building on each other. In this paper, I will present the most common data representations and data structures that can help get to know them, to master them, and then to reuse them for solving different tasks.

2. Visualization of simple data

At a younger age, when students have not encountered data structures yet, only simpler data and variables are available to work with. I find it important that even though in many cases understanding them is not a problem, it is worth introducing different tools and visualizations in programming education at the beginning. With the help of these we can easily represent variables, relationships between variables, or objects with several data members, which can then help the students understand the instruction sequences and programs based on them.

It is important for students to understand the connection between the variables. An easy example of this is when a value is placed in a given interval. Examples could be the number of lives in a computer game or the speed of a car with a fixed max value. But it can be an even stronger connection: when the data is actually part of an identity. An example of this is a clock with its hands (hours, minutes, seconds). Depending on the implementation, a complex data group can also appear as data members of an object, which, while helping to strengthen the connection between the data, requires a higher level of knowledge of the particular programming language.

With the help of Logo and Scratch, simple variables can be easily represented and interpreted (for example: turtle position, variable-based drawing procedures). In addition, the Towers of Hanoi game is also interesting for data representation and algorithmic purposes. Here, the data appear as simple

Received January 2022.

Cite as: Bende, I. (2022). Data Visualization in Programming Education. *Acta Didactica Napocensia,* 15(1), 52-60, https://doi.org/10.24193/adn.15.1.5

increasing disks on a track that can be displayed both by computer (Figure 1) and offline methods (of course, depending on which type of game we choose and how many bars and discs we have, the storage in array may be considered). The problem can be brought into the classroom relatively quickly, as students with a fixed, low size (three to four) input can even think of a command-based solution. Once more knowledge has been gained, a recursive solution can emerge later on.

Figure 1. *Display of Towers of Hanoi game in 3D*

3. Visualization of data structures

3.1. Array

The first more complex data structure in programming education is the array. It is an essential tool for basic algorithms, and an important tool for understanding and implementing other data structures later. Thus, the purpose of visualizations is not only to see how they are stored in the memory, but also to add a visual association to specific problems, and it helps to recreate and use the arrays in a future programming language.

3.1.1. Traditional visualization

Some programming languages can automatically display arrays (whether they contain elementary data types or more complex objects). The first method, while getting acquainted with the first programming language may be to look at this, so we can see how it looks like and helps to interpret the usage of the memory. Displaying this is similar to declaring, assigning a value to an array in a programming language.

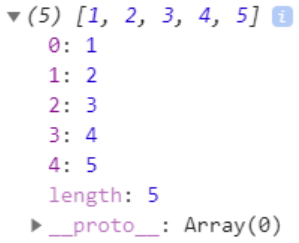

Figure 2. *Array visualization in a browser's console (JavaScript code ran by Google Chrome)*

The easiest way to try that out is to run a code written in JavaScript in a browser (Figure 2) (of course we can also display more complex class elements here). In this case, you can see the whole array, the values in it, and the indexes at which they are available. This is ideal for simple visualizations; however, it does not allow to comprehend more complex processes, and in case of larger, more complex data, clarity is impaired.

3.1.2. Display as bar graph

The most common method of displaying arrays containing numbers in visualizations is a bar chart (Figure 3), which can give a clear picture of the values in it and the whole array. This makes it easy to visualize the most commonly used programming theorems (counting, linear searching, maximum searching) and also helps us understand the sorting algorithms. By representing a column in a different colour, special elements can be highlighted, such as the following:

- for maximum search the largest value of the subinterval we examined so far (at the end the global maximum),
- for counting the values, which fulfils the condition,
- for linear search the first value, which fulfils the condition,
- for selection the values, which fulfils the condition (elements of the result array),
- for sorting algorithms, the two compared values, or the already sorted subarray elements.

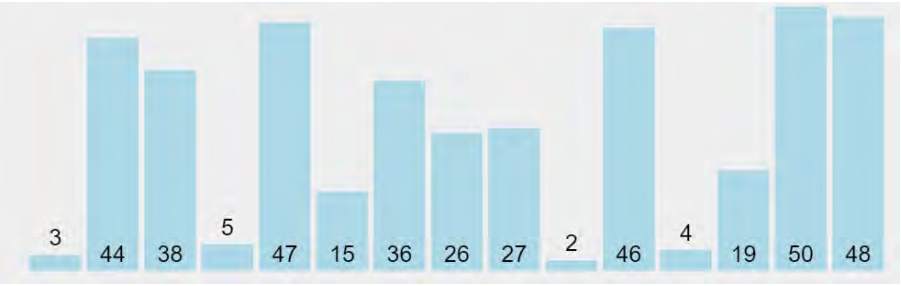

Figure 3. *Array visualization as column bar graph (Halim, 2011)*

For summation, a stacked column can be similarly good, because in the end we can see globally how the value of the sum came out.

3.1.3. Other type of visualizations on computers

Mostly, we use numbers as input in programming education, which is why the possibility of the aforementioned diagram representations was easy, but in many cases, we have more complex data, which we cannot represent this way (or at least not all of the data). For specific task- and problem-solving, however, we can use several creative modes.

For sorting algorithms, especially when comparing values, it is useful to know how many exchanges took place at which stage. A useful way to do this is to replace the columns with a smaller shape, so the changes made so far can be followed more easily, in addition to being able to monitor the current comparisons (for example, a visualization can be seen in Figure 4: circles indicate each item and shifts between positions are indicated by a line).

Figure 4. *Quicksort with circles (Zapponi, 2014)*

Slightly deviating from the basic and sorting algorithms, we can find interesting tasks that allow using special visualizations. The first is when the tasks relate to a coordinate system and the data appear in an array as two-dimensional points (geometric algorithms). These tasks can also help students understand mathematical concepts and create a connection between the two subjects. This method can be used for a wide range of tasks, but as an example, a typical practice task for competitions in this regard is to determine the smallest convex polygon that contains all the points.

Another common strategy for competitive tasks is the greedy algorithm. In its typical examples, the values are displayed in an array by interval (start-end point pairs) (Figure 5). Depending on the screen

size – up to a certain limit, as it shows limited amount of data –the visualization helps understand the algorithm, and also provides transparency and interconnection of the data, as well as a coherent picture of the whole (without starting the algorithm). An example of this strategy is when we know the arrival and departure time of the guests at an event, and then use those to determine the minimum number of photos that should be taken so that everyone is on at least one picture.

Figure 5. *Displaying intervals for greedy algorithm*

3.1.4. Offline visualization

In classroom conditions, the easiest method to visualize the array is when students represent a piece of data and array operations (such as an exchange) take place between them. The values of the students can be their name (based on this, it is even possible to sort them into an alphabetical order), or even a randomly assigned number (handheld board, badge). With this method, they can implement and play different algorithms in an offline environment[1]. Similarly, young people come across a sorting algorithm very early on, regardless of their prior knowledge of it: for example, for sorting a deck of cards in their hands or books on a bookshelf they use an algorithm similar to an insertion sort. Hereby I would like to mention a video material based on a similar principle, created in (Osztián et al., 2013), which shows the operation of more famous sorting algorithms in the form of folk dancing.

For people with visual impairment, it is possible to provide adjacent objects with various clearly visible markings (e.g., larger dots like on the dices) or Braille, which will allow them to play and learn how the algorithm works. Another concept is to use sound signals instead of physical objects. Recognizing and comparing individual sounds not only helps them to understand the progression of the algorithm, but also improves the hearing. Unfortunately, this significantly slows down the process (it takes a long time to play several different sounds, and the more data there is, the harder it is to distinguish them), so visualization of more presentable and more complex algorithms is not possible or very limited with this method.

3.2. Matrix

In terms of data representation, there are not many differences compared to the previously detailed array, since basically the matrix is usually implemented as an array of arrays. In practice, the simple representation works here as well, only in a two-dimensional form, the examined element(s) can be marked in different forms, thus highlighting the essential points of the algorithm. However, its use arises in many individual cases, tasks, and problems, so I introduce a few application areas of these.

Young people come across matrix-like solutions early on through simple games, think of a tic-tac-toe game (with a fixed size playground) or a chessboard. In both cases, the data is stored in a predefined "matrix", and displaying them with data (figures, signs) can simplify the understanding of the concept. But in this case, regular venues that reflect the structure of the matrix, such as a cinema, theatre, or even a multi-story house, may be considered. In the same way a regular maze (which may need some

[1] I do not mention height directly in the list, as that might be detrimental to some students.

graph knowledge) can perfectly be presented with a matrix and then the traversal could appear as a command-based solution (for example, if each element of the matrix stores the form of objects or arrays to which positions, we can go or not).

Another way is to reverse the process: when the input data does not clarify the use of the matrix, but the data structure is necessary to solve it, or significantly speeds up the required program's runtime. One of the examples for this kind of usage in problem-solving is the dynamic programming, the most well-known task in the topic is the knapsack problem: Given a backpack with a capacity of K and N objects, the mass and value of them is known, we want to determine which objects to put away, if we want to pack the most values, but still within the specified capacity. A simpler version for the topic is when we have a K * N table, where the goal is to calculate how many different steps we can take from the lower left corner to the upper right corner if we can only move up or to the right (Figure 6).

1	5	15	35
1	4	10	20
1	3	6	10
1	2	3	4
1	1	1	1

(v,w)	0	1	2	3	4	5	6	7
(2,3)	0	0	0	2	2	2	2	2
(2,1)	0	2	2	2	4	4	4	4
(4,3)	0	2	2	4	6	6	6	8
(5,4)	0	2	2	4	6	7	7	9
(3,2)	0	2	3	5	6	7	9	10

Figure 6. *Display of matrix (a) for counting the possible number of moves, and (b) for the knapsack problem (in the first column the value-weight pairs presented). Green cells show the results and blue cells show some currently investigated parameters*

3.3. Queue and stack

3.3.1. General visualization

While the queue and stack are important, array-based data structures that emerge relatively quickly during programming education are not very special in terms of visualization, thanks to their eloquent name, minimal usability, and simple basic operations. In this case, the device must be able to display the elements, then remove/insert them within an animation. The usual representation in Figure 7 and 8 on the other hand, can perfectly illustrate these.

They can be easily demonstrated in a classroom setting. A possible game to practice, as mentioned in (Bhagate et al., 2016), is when students line up according to the data structure or enter the classroom and only the one who follows the specific operation can leave the structure (in the case of a stack, the last student who entered, while in the case of a queue the first one). Creatively hand-made tools can also appear when displaying a queue, for example, we make a simple tube in which we place balls (this strictly allows the elements to be inserted or removed), if we close one end of the tube, we immediately get a stack (since only last inserted item can be taken out).

3.3.2. Example task for stack

Description: "For each element of a series of numbers, add/assign the closest smaller one from the preceding numbers (if no such, then -1)!" One of the possible solutions can be to continuously push the elements into a stack, which can still be in the result array. Figure 7 shows a step in the visualization mixed with minimal array visualization.

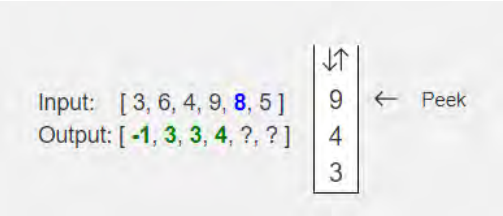

Figure 7. *Visualization of stack for the example task*

3.3.3. Example task for queue

Description: "At one train station, a detour was built from the main track, each of which can only be moved from right to left. You can park any number of cars on the detour, but no cars can stop on the main track. An assembly arrives from the right, with numbered, mixed wagons. The wagons are identified by different serial numbers between 1 and K. Create a program that specifies the maximum number of cars we can queue using the detour and send to the left so that there are no cars left on the detour in the end!" (Menyhárt, 2018) For the solution we store the trains on the detour in a queue, that contains train cars in numerical order. Figure 8 shows a step in the visualization mixed with minimal array visualization.

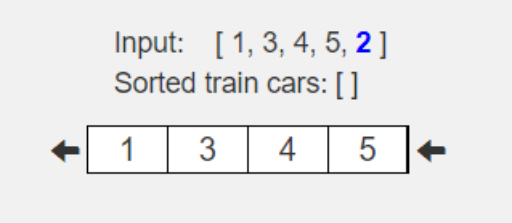

Figure 8. *Visualization of queue for the example task*

3.4. Linked list

Visualizing a linked list is very similar to visualizing the structures mentioned so far. In practice, the goal is to display consecutive elements so that the operations performed on it can be properly represented. In this case, we are talking about adding a new item, deleting an item, and using pointers for different purposes. The Figure 9 shows how to represent the data and then operations appear with the help of an animation. Like the array, it is easy to try and practice with offline tools. Based on the classroom example used for the array, students can represent each value and represent the pointers by touching another student's shoulder (using one hand for singly and both for doubly linked lists).

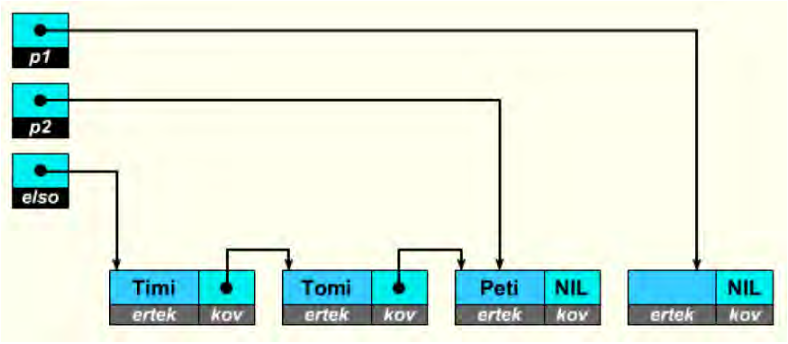

Figure 9. *Visualization of singly linked list (Végh, 2017)*

3.5. Graph

Usually graphs only appear at a higher level of programming education. At high school level, the knowledge of using this data structure is required for competitive tasks, and at university it usually

appears from the second year as a possible data structure (which can be used to represent the concept introduced in mathematics or to solve certain tasks in different, even faster ways).

3.5.1. Storage visualization

The most basic form of display is when the data structure appears similarly as in the memory. Adjacency list representation provides less scope for transparency and less help with the traceability of an algorithm. The situation is better with the usage of the adjacency matrix, but similar problems arise here: the matrix is not transparent, it will soon become too large, and in the case of an undirected graph there are a lot of duplicated data.

3.5.2. Graphic display

Unfortunately, generally generated representations of graphs are not easy to implement. For basic graph algorithms, the visualization helps to understand and improves the transparency of the data structure if the graph can be embedded in the plane (not a binding rule, but it can be useful for beginners), but this will not necessarily be the case for every input. In case of a planar graph, it is still difficult to display it in a transparent way (do not form groups, it should be cleaned, transparent). Therefore, this is usually represented either by a fixed input or with minimal modification options (for example, in the visualization package prepared by David Galles (Galles, 2011), only the edges between the vertices can be randomized, the positions of the vertices or their names are retained). For graphical representation, the standard mathematical representation is used (Figure 10). However, the advantage of the computer display compared to the offline version is that it is possible to highlight the edges/vertices (with a thicker frame, in a different colour). The reasons for highlighting could be the following:

- for traversing (DFS, BFS) the currently investigated edge, vertex and with different colour the already traversed ones,
- for finding the shortest path, the current best option,
- for graph colouring the algorithm is speaking for itself, the vertices coloured with the same method as the algorithm.

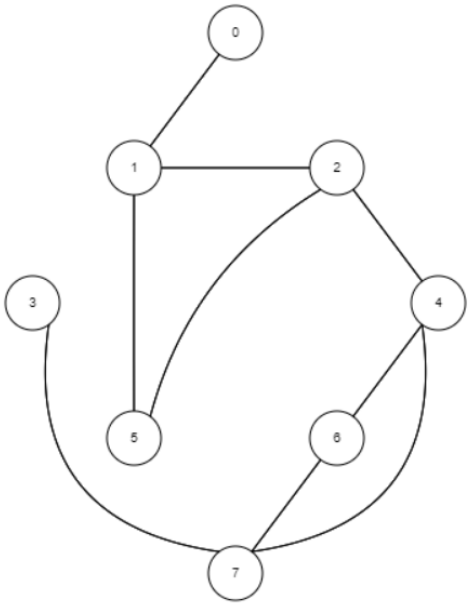

Figure 10. *Graphic display of graph (Galles, 2011)*

The tree is a similarly important data structure in informatics, but since these are practically acyclic connected graphs, there is no significant difference in terms of visualization. However, a programmed

general display is easier to implement since we can clearly determine the level of each value, while we specify the input.

4. Mastering data structures by creating AV

The effectiveness of algorithmic visualization tools in learning increases if students are active participants in the usage of the tool, learning process (Naps et al., 2002). Moreover, we can achieve an even better result if we do not use pre-created visualization, but the students create them. However, it can take too much time to create these which depending on the technology, (for example, a web page that presents a particular algorithm/data structure nicely and in detail for each input will take more time than a simple presentation for a fixed input) can mean a time difference, which does not help understand the algorithm, but improves the knowledge of the given environment. However, research in (Hundhausen et al., 2000) reveals that there was no significant difference in efficiency between someone creating a modern tool or a simpler visualization. For this reason, it is worth to assign such tasks to students, because in addition to mastering the concepts of programming education, they can also improve their creativity, which can result in spectacular productions. As Futschek stated "the aim of the teacher is not to present solutions but to support students in their learning process, he should motivate students to make progress in finding solutions" (Futschek et al., 2010).

5. Mastering data structures by using utility software

Another method could be to use a separate software or plugin that automatically displays the data structures in the written code at runtime. This definitely requires some prior knowledge, which can be in the form of a sample code or by a teacher presentation (since an existing source code is needed for that, we can run to see the change and movement of the data). In (Nathasya et al., 2019), it turns out that especially for students that can advance more slowly (in the case of others, the difference is smaller), we can see a visible difference in the success of solving a task and the amount of time required for that if such tools are used in the meantime. In that article, they used the DS-PITON they developed, but we could also include Jeliot 3 (Moreno et al., 2004) or PlayVisualizerC (Ishizue et al., 2018). These are generally implemented programming visualizations, that can help understand and debug more complex tasks and algorithms, but are not necessarily ideal for initial, demonstration materials: the generally displayed data structures make it less possible to examine special situations, as unnecessary variables, structures appear on the screen, which also makes the usage of the tool more difficult.

There is another type of applets, languages, which were created only for the purpose to easily create fixed animations. For this type of software, a good example is JAWAA, that „can provide students with an alternative and visual perspective, which may help increase their understanding" (Pierson et al., 1998). Unfortunately, as Törley mentioned it is "easy to make visualizations with it, but the system does not support the interactivity" (Törley, 2013).

6. Summary

As we have seen, it is possible to present and teach data structures in many ways. In each case, we tried to provide as many examples as possible, which thus simplifies its usage in education. In conclusion through simple examples, virtually any data structure can appear in IT classes at an early stage, thus facilitating subsequent educational tasks and giving colour to the entire programming teaching process. Another result to be mentioned is that we should rely as much as possible on the students' work and creativity, as this will ultimately help them understand how they can use these algorithms/data structures in solving specific tasks.

References

Bhagate, S., Nuli, U. (2016). Innovative Methods for Teaching Data Structures and Algorithms. *Journal of Engineering Education Transformations*.

Futschek, G., Moschitz, J. (2010). Developing Algorithmic Thinking by Inventing and Playing Algorithms. Constructionist approaches to creative learning, thinking and education. Bratislava: Library and Publishing Centre, Facutly of Mathematics, Physics and Informatics, Comenius University.

Galles, D. (2011). *Data Structure Visualizations*. Retrieved November 8, 2022 from https://www.cs.usfca.edu/~galles/visualization/Algorithms.html.

Halim, S. (2011). VisuAlgo. Retrieved November 8, 2022 from https://visualgo.net/en/sorting.

Hundhausen, C., Douglas, S. A. (2000). Using visualizations to learn algorithms: Should students construct their own, or view an expert's? *Proceedings 2000 IEEE International Symposium on Visual Languages*, 21-28. https://doi.org/10.1109/VL.2000.874346

Ishizue, R., Sakamoto, K., Washizaki, H., Fukazawa, Y. (2018). PVC: Visualizing C Programs on Web Browsers for Novices. *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 245-250. https://doi.org/10.1145/3159450.3159566

Kovásznai, G., Kusper, G. (2011). *Artificial Intelligence and its Teaching*. Eger: Eszterházy Károly Főiskola.

Menyhárt, L. G. (2018). *Sor típus*. Budapest: Eötvös Loránd Tudományegyetem.

Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, 373-376. https://doi.org/10.1145/989863.989928

Naps, L., Rössling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J. Á. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, Volume 35, Issue 2, 131–152. https://doi.org/10.1145/782941.782998

Nathasya, R. A., Karnalim, O., Ayub, M. (2019). Integrating program and algorithm visualisation for learning data structure implementation. *Egyptian Informatics Journal*, vol. 20, no. 3, 193–204. https://doi.org/10.1016/j.eij.2019.05.001

Osztián, P. R., Kátai, Z. (2013). AlgoRythmics: Tánctól a Kódig (AlgoRythmics: From Dance to Code). *INFODIDACT 2013*, Budapest: Webdidaktika alapítvány.

Pierson, W. C., Rodger, S. H. (1998). Web-based animation of data structures using JAWAA. SIGCSE '98: *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, 267–271. https://doi.org/10.1145/273133.274310

Törley, G. (2013). *Vizualizáció a Programozástanításban* (Visualization in Programming Teaching). Budapest: Eötvös Loránd Tudományegyetem.

Végh, L. (2017). *A Programozás Tanulásának és Tanításának Támogatása Elektronikus Tananyagba beépíthető Interaktív Animációs Modellekkel* (Support for Learning and Teaching Programming with Interactive Animation Models that can be integrated into Electronic Curriculum), Budapest: Eötvös Loránd Tudományegyetem.

Zapponi, C. (2014). *SORTING - Visualizing sorting algorithms*. Retrieved November 8, 2022 from https://sorting.at/.

Authors

Imre Bende, Eötvös Loránd University, Budapest (Hungary). E-mail: beiraai@inf.elte.hu