A Lightweight e-Learning System for Algorithms and Data Structures

Volodimir Begy
University of Vienna
Währingerstr. 29
Vienna, Austria
volodimir.begy@univie.ac.at

Erich Schikuta[*]
University of Vienna
Währingerstr. 29
Vienna, Austria
erich.schikuta@univie.ac.at

ABSTRACT

We present webAD, a web-based e-learning platform for the visualization of algorithms and data structures. It follows a light-weight server-less implementation paradigm and pursues a minimalistic vision: no installation or configuration effort, multi device support, clear structure of didactic content and simple extensibility for developers. Compared to other visualization tools for algorithms and data structures webAD puts a high value on an innovative flexible tape-recorder mechanism. Based purely on HTML5 and JavaScript with the smallest usage of external libraries it is designed according to the Model-View-Controller architectural pattern and works as a fat client. A thorough analysis of the system in respect to usability and extensibility is performed and findings and recommendations are critically discussed.

CCS Concepts

•Applied computing → Interactive learning environments; E-learning; •Human-centered computing → *Web-based interaction;*

Keywords

E-learning, Visualization, Algorithms and Data Structures, Web-based Tool

1. INTRODUCTION

Algorithms and data structures build the foundation of program development. Thus, an effective and high quality education in their sphere is vital for any computer science student.

The main problem of teaching and studying algorithms and data structures is, due to the inherent complexity, to visualize how these algorithms operate. Conventionally, algorithms are described in a static way by their code representation within a (mostly) formally defined description framework (programming language or pseudo code). This makes it often hard to understand their specific flow of control.

Traditional learning methods imply insufficiencies due to their static nature. Using predefined learning materials students may develop false interpretations, since the possibility of trying out any desired combination of data sets on demonstrated algorithms and data structures is absent.

Several lecture books try to overcome this problem by sketching the state of an algorithm by the graphical representation of the data it is operating on. This is done by graphically depicting specific states of the data structure during the execution of the algorithm. However, this approach gives snapshots of the execution flow at specific predefined points only and does not allow students to interactively explore the behaviour of the algorithm in focus.

Studies mentioned by [2] demonstrated, that in average classical and digital approaches of teaching achieve the same effect size of knowledge extraction. The data on the effect size collected within these studies forms a normal distribution, confirming the hypothesis, that only in rare occasions e-learning stimulates significantly better results than ordinary learning and vice versa. Visualization tool for algorithms and data structures is obviously one of such rare cases, because it requires interaction and animation. Furthermore, computer science students are rather accustomed to e-learning.

The paper is organized as follows. In the next section we give an overview on previous work and an analysis of the state of the art in this area. Based on this survey we defined in the following section 3 the research goals to be met by our new approach. In section 4 the design decisions and the derived system architecture are presented. The developed webAD system by its appearance and its functionality is introduced in section 5. For evaluation the results of a conducted usability analysis and a students' project use case (section 6) are described. Section 7 depicts a discussion of the findings and possible improvements. The paper closes with a conclusion and look at further work.

2. BASELINE RESEARCH AND STATE OF THE ART

The preceding developments of our group in the field of algorithm and data structure visualization include several attempts to construct such a tool: VADer [7], LUCAS [8]

[*]Additional authors are listed in the additional authors section at the end of paper.

iiWAS '16, November 28 - 30, 2016, Singapore, Singapore
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4807-2/16/11...$15.00
DOI: http://dx.doi.org/10.1145/3011141.3011181

and NetLuke [6]. All of them had critical issues ceasing further contribution. VADer could not be maintained anymore due to a radically new Java release, which made the adaptation of the platform too costly. LUCAS was not aimed at mobile devices due to the technological trends of its era. The use of web technologies in NetLuke became surplus, forming a too complex architecture with overflown workflow. During the design phase of webAD we took into account these previous nuisance experiences and eliminated them.

We analyzed other tools for visualization of algorithms and data structures in order to see, how well webAD is positioned in the sphere. AlgoViz [1] is a scientific portal dedicated to such visualizers. In this section selected examples from the portal's Hall of Fame (JHAVÉ [2], Algorithms in Action [3]) and previous projects of our group (VADer, NetLuke) are analyzed in context of didactics and usability. These tools are selected for the analysis and comparison because of numerous reasons. First of all, AlgoViz awarded JHAVÉ and Algorithms in Action for their implementations. Thus, it is interesting to examine one of the best programs in the portal's ratings. VADer and NetLuke are chosen, because they reflect the paving of the path from first to last attempts to implement such a tool. At last, all of the tools together with webAD form a wide spectrum of completely different architectures and technologies, which makes it interesting to compare heterogeneous products. There are seven characteristics picked to compare the four tools with webAD: amount of implemented modules, multi device support, presence of tape-recorder, additional theoretical information about topics, limitless dynamics of user input (user may input any combination of data sets), existence of pre-defined examples and low dependence on internet connection (application can operate without internet and limitations once downloaded, and the download is provided). These criteria are chosen, because they focus on two main spheres, which need to be considered by an e-learning visualization tool: coverage of didactic material and usability. Additionally, some of them reflect the previously described goals set for webAD, and others are inspired by the analysis performed by authors of NetLuke[6], if found suitable.

The results are depicted below (see figure 1).

	modules amount	multi device	tape-recorder	theoretical info	dynamics of input	pre-defined examples	low internet dependence
JHAVÉ	32	✗	✓	✓	✗	✓	✗
Algorithms in Action	33	✗	✓	✓	✓	✓	✗
VADer	10	✗	✓	✗	✓	✓	✓
NetLuke	4	✓	✓	✓	✓	✓	✗
webAD	14	✓	✓	✓	✓	✓	✓

Figure 1: Comparison of Available Tools.

As a downloadable Java program, JHAVÉ is not adapted for mobile devices, not all algorithms and data structures allow users to input arbitrary data sets for visualization. Before any module can be accessed, it has to be retrieved from the server over the network. Each interface component of Algorithms in Action is launched in a separate browser window, making the tool not optimal for gadgets with small screen sizes. The program is designed as a thick client, but internet is required for navigation among didactic modules, while no download is provided. VADer was an old software from 1999; obviously it was not designed for other devices than desktop computers. The architecture of NetLuke implies heavy network reliability.

webAD complies with all identified requirements, the amount of implemented modules is solid taking into account the development period.

3. RESEARCH GOALS

Based on our analysis of the state of the art we divided the priorities set for webAD into two groups: ideological and technical ones. The first group consists of two factors. The reflection of didactic materials of the corresponding university course is essential. The students are the customers of the platform, thus they should be familiar with its structure and have a short learning curve. The second principle of this category is minimalism, clear structure and simple use for cognitive offload. Cognitive offload is provided by usability, which takes into account human perception capabilities. We integrated only the vital functions into the workflow of the tool, avoiding overflown interface and interaction. Thus, we aimed at a comprehensive tool. From the information system point of view, we identified several concrete goals in order to form a software framework for the realization of our vision, which are highlighted in this paragraph. Since this is a university project, the platform should be extendible, because numerous contributors will work on it; this is why we invested a lot in a proper Model-View-Controller (MVC) realization, providing a loosely coupled modular architecture. After several attempts to produce such a visualizer, some of which were condemned because of the incompatibility with latest trends, we grasped the importance of portability and robustness in the face of inevitable technological change. This is one of the reasons why the time proven technologies of JavaScript and HTML5 were chosen. The tool should be based on a fat client, with no installation or configuration effort and provide multi device support, supporting our minimalistic spirit for the convenient use by students. JavaScript combined with HTML5 again provides the suitable foundation for such goals. Furthermore, we value independence from external libraries and tools. We believe, that a system consisting of native components provides a clear overview. This is why we use only very light JavaScript libraries, which do not affect the architecture: jQuery [4] for handling of HTML documents, KineticJS [5] for HTML5 canvas drawing and annyang [6] for easy access to built in browser speech recognition. Last, but not least, we concentrated on a flexible innovative tape-recorder, which enables the user to perform any operation on any chosen state.

4. SYSTEM ARCHITECTURE

This section justifies the design decisions of webAD on the deployment and software level. All of them were met in order to realize a simple, intuitive and homogenous tool.

4.1 Thick Client

[1]http://algoviz.org
[2]http://jhave.org
[3]http://aia.cis.unimelb.edu.au/demo/index.html

[4]https://jquery.com
[5]http://kineticjs.com
[6]https://www.talater.com/annyang

When designing an application operating over a network on the deployment level the developer generally faces a choice between a thin and a thick client. webAD is realized as a complete thick client, which means the application is capable of providing full functionality without internet connection once it is loaded. Contrary, a thin client presents the output only, while the server performs the computations. The decision to apply this architecture was mainly caused by negative experiences collected from the thin client realized by NetLuke: the tool was heavily dependent on internet; the execution of the workflow was not perfectly smooth because of the synchronization with the server. A thick client based on JavaScript and HTML5 embodies numerous advantages, which we experienced with webAD. The character of this architecture supports the argument in favor of such design: the application becomes more robust once it cannot be interrupted by network disturbances. Consequently, the user feels in charge while using the tool. The design is more responsive, because there are no round-trips between server and client for view rendering. The server deployment and maintenance is primitive, in order to operate only the static project directory needs to be provided. Last, but not least, new marginal users barely affect the workload of the server, resulting in low costs. However, a thin client also has its specific advantages. This kind of architecture makes it easier to implement a business model, session management and stealing for the application. These factors motivated NetLuke for such choice.

4.2 Model-View-Controller

The loose coupling of the independent modules of the software is essential for high quality implementation. It ensures, that the code is more flexible, extendible and readable. Since webAD is an interactive visualizer for algorithms and data structures, it was obvious for us, that Model-View-Controller architectural pattern fits our needs perfectly: models represent the behavior of algorithms and data structures, views serve for visualization, and controller realizes the interaction. MVC is a de facto standard for programs with graphical user interfaces. Our implementation of this pattern is defined by the technological characteristics of HTML5 and JavaScript (see figure 2).

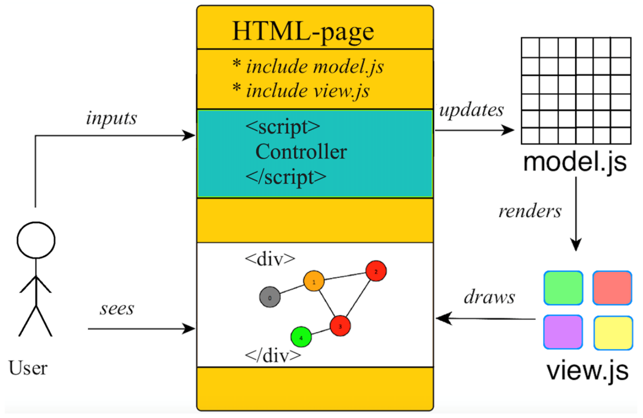


Figure 2: Model-View-Controller realization.

Such realization implies robustness and flexibility for both developers and users. Contributors may modify models without having to take into account the attached view, and vice versa. During the runtime students can use functions of models and views concurrently without conflicts. For example, while a sorting algorithm is in progress, the colors and the scales can be adjusted.

5. THE WEBAD SYSTEM

webAD consists of a portal (see figure 3) and interlinked sites for particular thematic modules. The layout is robustly scalable in any modern web browser adapted for HTML5 and is realized through Scalable Vector Graphics, which is a W3C standard. This ensures better look and feel, since the images can be zoomed in endlessly without losing quality.

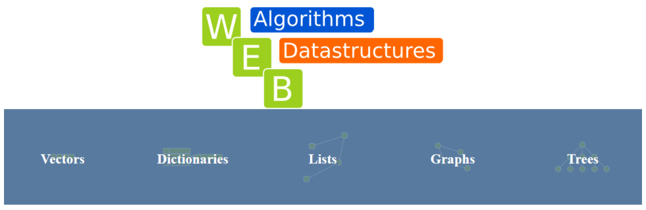


Figure 3: The webAD Start Screen.

All implemented modules are split into didactic chapters according to the contents of the Algorithms and Data Structures practical course: vectors, dictionaries, lists, graphs and trees. The original webAD system [1] comprises in sum 14 algorithms and data structures, which were chosen based on their priority within the university course.

Vectors include Bubblesort (see figure 8), Selectionsort (see figure 9), Quicksort (see figure 10), and Heapsort (see figure 11).

Dictionaries contain Linear probing hash (see figure 12), Double hashing (see figure 13), and Linear hashing (see figure 14).

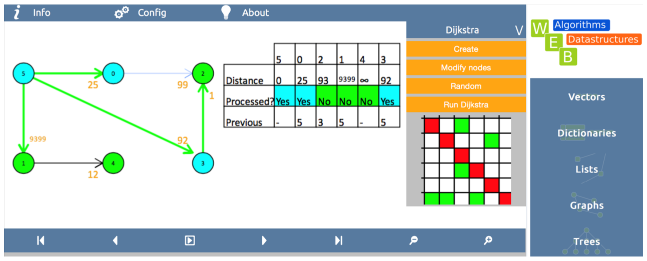
Graphs consist of Breadth first search (see figure 15), Depth first search (see figure 16), Kruskal minimum spanning tree algorithm (see figure 17), and Dijkstra shortest path algorithm (see figure 4).

Visualized trees are Binary search tree (see figure 18), B+ tree (see figure 5), and Binary minimum heap (see figure 19).

Lists are not implemented yet.

This structure in combination with minimalistic interface forms a comprehensive tool. When a certain module is chosen, the algorithm/data structure manipulation window appears (see figure 4). The upper panel includes Info, Config and About sections. User may read theoretical information about the algorithm, adjust suitable settings, such as animation speed or colors and gather an insight about webAD. The shrinkable menu, which provides corresponding operations, which students may perform on the data structure (such as insertion, deletion and others) is as well appended to the upper panel. The lower panel has the tape-recorder and the zooming for the view. The right panel includes same didactic chapters as the homepage. The logo in the right upper corner leads to the main site. The main operating area is in the middle of the screen.

All graphs have an additional component of the interface, the mini adjacency matrix. It was introduced, because graphs with different adjacency matrices may have the same appearance on the operating area, for example when some overflown nodes are disconnected from initial node's path. While switching such instances using tape-recorder student could get the feeling that nothing is changing. To avoid this factor the matrix is integrated into the interface. As described later, such an interface proved to be intuitive according to the results of usability testing.



	5	0	2	1	4	3
Distance	0	25	93	9399	∞	92
Processed?	Yes	Yes	No	No	No	Yes
Previous	-	5	3	5	-	5

Figure 4: User Interaction interface.

5.1 Tape Recorder

Tape-recorder is a component of webAD, which stores all states produced during the execution of an algorithm. Students are able to navigate between such stored instances in order to analyze the dynamic behavior. Though numerous algorithm visualization tools support a tape-recorder for instance switching, we focused on a flexible innovative technique, which would support the user in any desirable way. The instances can be switched back, forth, to the very first and last states. Tape-recorder stores only valid instances; this is why any object can be manipulated in any way at any chosen state. Such approach is very effective for educational purposes, because a student may go back and validate own interpretation not only by analyzing the change set, but also by applying a different operation on the instance instead of the previous one and see the alternative outcome. If user navigates to an older state and performs a new operation on the object, the consecutive states are erased and the newly appeared one is appended to the tape-recorder in order to keep the track of states consistent. If an algorithm supports continuously running animation, it can be re-launched. The tape-recorder is demonstrated by one minimal scenario (see figure 5).

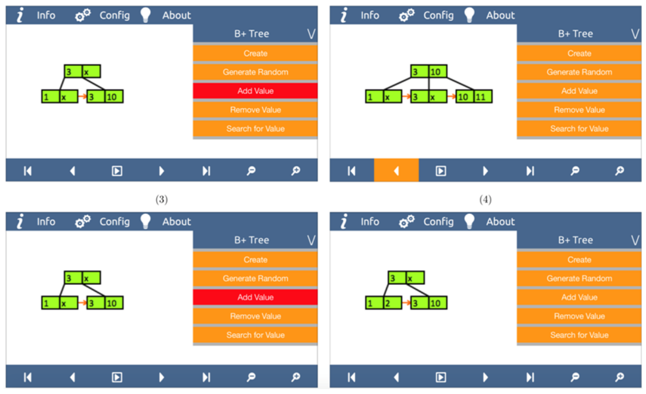


Figure 5: Tape recorder interface.

An 11 is inserted into a B+ tree. Consequently the animation illustrates the modification of the data structure: a leaf node is split and the indexes within the root node are updated. In the next step the user navigates back to the previous state and would like to see, what would happen, if a 2 was added instead. While the alternative tree is being constructed, no split is occurring. Observing the visualization students may draw conclusions about the working principles of node splitting within a B+ tree. In an analog manner any available operation can be performed on any stored state, and any phenomena can be studied in detail.

5.2 Speech Control

Besides the trivial digital input methods we also implemented support of speech commands. The feature is based on the annyang JavaScript library and uses the built in browser speech recognition. Currently such input method is realized in bubble sort module. The user may trigger the sorting by pronouncing "Sort". If the sorting is in process at that time, the program will respond "Already sorting". Students may as well pause the animation by saying "Stop". In an analog way, if the execution is aborted at present, webAD will reply "Already stopped". Besides these instructions the animation speed can be increased to the maximum by commanding "Fast" and to the minimum by telling "Slow". Contemporary native browser tools for speech recognition are nowadays not flawless, thus the instructions should be pronounced sharply in order for webAD to be able to understand them. Since webAD is founded on standard technologies and is web-based, we believe, that this feature will become more fault tolerant with the time, once the browsers improve their speech recognition functionality. Obviously not all browsers provide this feature. It can be used for example in current versions of Google Chrome and Mozilla Firefox. However, it is not provided for instance in Safari.

6. WEBAD EVALUATION

6.1 Usability Analysis

We conducted usability testing with the goal to collect quantitative and subjective measurements on the system. Observations mentioned in [3] indicate, that 10 testing users would most likely reveal 90 percent of usability faults. This is why we chose 10 computer science students from our faculty for the performance of usability testing. Some of them at the time of the testing had already taken the Algorithms and Data Structures practical course, while others had not. Picking such participants we tried to sample students, who would reflect the actual user group as objectively as possible. The tests were performed within the scope of most up-to-date webAD version. We constructed 10 tasks, which would approach two main questions concerning the platform, how easily can users navigate among the sites of the platform, and how intuitive is it to manipulate the visualization. The tasks together built a scenario, which covered all the functionality of the platform. First, the user is at the homepage and has to navigate to linear probing (task 1). At the linear probing one has to insert a number 10 into the table in the static mode (task 2). Then the participant should change the art of the table to the extendible (task 3). Afterwards user should navigate from linear probing to quick sort (task 4). While surfing on the site dedicated to quick sort, user should open the info about quick sort (task 5). Following this, one has to adjust all default 5 colors of the vector for the quick sort (task 6). Task 7 was to set up a manual array of 5 elements. Within the 8th task the user should use tape-recorder in order to navigate to the initial array state, before the manipulation had started, once the array is sorted. Task 9 was to create a graph consisting of 5 nodes, while each of these nodes is connected to exactly 2 other nodes within the Dijkstra algorithm. The last 10th task was to modify the previously created graph, inserting one new node into it and connecting it to 2 other nodes. Users performed these tasks on desktop (laptops) and mobile (smartphones and tablets) devices, while we measured the time in seconds, needed for the completion of each task by each participant on each device. Analyzing the collected data we identified a trend, that some tasks (for example task 10) are rather suited for desktop, while others for mobile devices (task 4). The time is depicted in seconds on the y-axes; the discrete device type is on the x-axes. The facet is wrapped around the tasks number 4 and 10 (see figure 6). This is probably caused by the differences of the input technologies (mouse, touch pad and touch screen) and the device screen size.

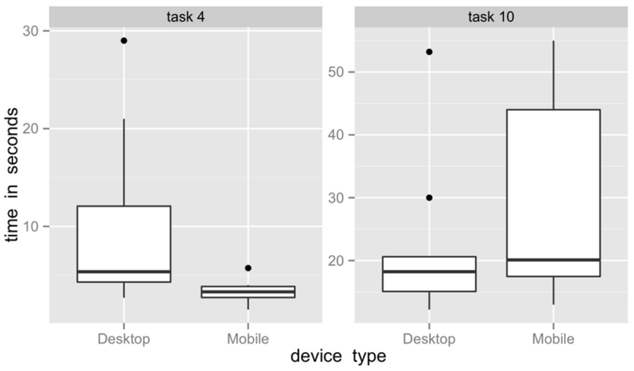


Figure 6: Box plot representing quantitative measures.

After the completion of the tasks the participants were asked to give their opinion on the platform, providing subjective measures as well. Each student was rating following 5 properties from 0 (lowest grade) to 10 (highest grade): visual design of the tool (P1), ease of usage and cognitive offload (P2), quality of software implementation (P3), usefulness of tape-recorder (P4) and likelihood of usage while studying for the Algorithms and Data Structures course (P5). The survey results are depicted through a box plot as well (see Figure 7).

It can be seen, that webAD left an overall positive subjective impression on the users, with only few outliers. During these tests we detected one major problem with usability, which confronted all of the participants: after an array-sorting algorithm terminates, user had to push the "Pause" button before being able to use the tape-recorder. We eliminated this issue afterwards, making the tape-recorder go into the stand-by mode automatically, once the sorting workflow is finished. At last we asked the participants, which missing for them thematic modules they would like to see within webAD. Each user could name unlimited amount of algorithms and data structures desired. Three participants voted for bucket sort, two students chose counting sort, cuckoo hashing, linked list and radix sort, while for coalesced hashing, doubly linked list, extendible hashing, insertion sort, merge sort, random sort and separate chaining each one vote was cast.

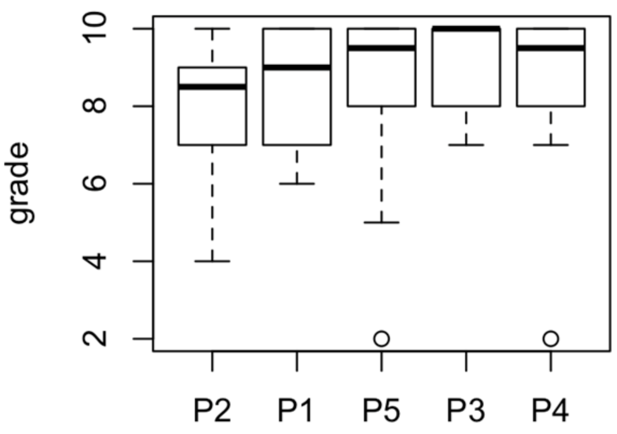


Figure 7: Box plot representing subjective measures.

6.2 Evaluation of Extensibility

One of the guiding principles of the development of webAD was extensibility. It is key for the success of webAD that new components realizing new algorithms or data structures have to be added easily by other developers. For justification and evaluation of this issue a students' project was established in the summer term 2016. In the course of their Bachelor projects eight students added eight new algorithm and data structure components to webAD. Goal was to show that for an average software developer with no specific pre-knowledge of technology the development of new code components should be feasible. Prerequisites to this effort were also strict deadlines for the milestones, as analysis, design, development of code, and thesis/manual delivery. The only specific accompanying helper was a developer's guide provided by webAD [1]. The eight developed components comprised the following algorithm and data structure topics:

- Mergesort (see figure 20)
- Counting Sort (see figure 21)
- Bucketsort (see figure 22)
- LSD Radixsort (see figure 23)
- Separate Chaining Hash (see figure 24)
- Extendible Hashing (see figure 25)
- Tree Traversal (see figure 26)
- Trie (see figure 27)

At the end of this student project the students reported a critical discussion of the webAD development framework, highlighting strengths and weaknesses, which are reported in the next section.

7. DISCUSSION AND FINDINGS

After numerous iterations of re-design and implementation of the platform and thorough analysis of other tools plentiful experiences and findings are extracted. Generally it is hard to make unambiguous suggestions concerning the applied architecture, used technologies and tools, because different approaches imply support of different goals set by different developers. This is why the reflections of this section mostly focus on generic aspects of usability.

7.1 Recommendations

One of the biggest lessons learned by webAD, is that the design matters a lot. The implementation may have great quality from the information system point of view, but repulsive appearance may and will discourage the users. Color is a major part of the appearance. An important law for choosing the palette in accordance to different regions, is that the smaller the area, the harder it is to perceive the color. This is why such spaces should be marked through bright and highly saturated colors, they appear greater. On the other hand, if such colors are applied to large surfaces, they stimulate the cognition in an exaggerated way and should be ignored [5]. Since we tried to take best care of cognitive offload for the users, it was not the only factor taken into account speaking of colors. Another important lesson retrieved, is that a limited amount of colors should be used for nominal data, because an average human can differentiate about 8 hues in the view [4, 5]. In webAD this concerns color legends for data structures, for example the choice of the palette for nodes or edges. We recommend to always follow these principles while decorating the user interface and the views.

There are several lessons learned from NetLuke, which have proven to be reasonable in webAD. The amount of the interface components should be minimized. When the platform extends with overflown functionality, it becomes hard to find the needed feature. However, all vital functionality should be visible constantly, users should not strain the nerve in search. In comparison, in NetLuke the operations performed on algorithms/data structures are grouped into the shrinked Algorithm Control Menu, while in webAD they are visible by default. The workflow of the program should be intuitive and lessen the required interaction. For instance in NetLuke after creation or manipulation of the data structure, it should be brought into the algorithm manipulation mode, before an algorithm can be launched. Again, this constantly requires more effort from users. In webAD all such specialities are handled by the system on the code level. Finally, the analog workflows should be consistent in different implemented modules, so that the user does not have to change the habits. These are the three advises we give concerning the integration of workflow.

Concerning the architecture of the platform, an online fat client is recommended for robustness and following of the cloud trend, which are especially valued by mobile devices. JavaScript is a perfect candidate for both purposes. A visualizer should doubtlessly realize the Model-View-Controller architectural pattern to avoid spaghetti code and unclear workflow. This was not the case with webAD, but the library Backbone.js may be of big interest for its realization in JavaScript. For the drawing of the views KineticJS is obviously a good option, because it provides handy features along with high performance and became a standard. Finally, as mentioned before, jQuery reduces the effort required for manipulation of HTML documents significantly.

7.2 General Improvements

Even though webAD has justified our goals and expectations, some design decisions pay off through minor disadvantages. Some of them could be improved using chosen actual technologies:

- The decision to avoid the usage of timers and callbacks for pausing/resuming of the animation in order to achieve higher performance and browser independence resulted in the fact, that the animations can be paused in consistent states only, because the state of the model is fully reflected in the animation. It would be better for users to be able to pause and resume at any time.
- Several phenomena (for instance split/merging in the B+ tree or complex removal in the binary search tree) could be animated in a more detailed way, showing how they occur in a more continuous manner, which would require vast implementation effort with chosen technologies. Now only the consistent states before and after their occurrence are shown.
- webAD has currently no options for persistence. The previously used TaffyDB, promoted as JavaScript Database just provided wrappers for runtime storage objects, which could be accessed in a classic manner of relational database management systems, but could not be persisted. In the current version of webAD no persistence is needed, mainly because we believe, that it violates the pure client side architecture. If new components, which imply such storage will be desired (for example feedback by users), the introduction of additional technologies will be necessary. A good candidate is PHP, which is developed for browsers and supports database drivers.
- Often operations on models (for example insertion into a certain data structure) have a limited range for values because of view purposes: enormous numbers intersect the drawn borders. Though the allowed intervals are still huge and enable high dynamics, it would be optimal to write view algorithms, which resize the numbers according to their size (such algorithm is written for weighted directed graphs) for all modules. The current implementation of system's workflow and models always has limitless dynamics, these ranges are introduced for better look&feel only and can be easily turned off by removing the validation, once the proper view algorithms are implemented.

7.3 Students' Project Findings

Generally, the framework was judged as easy to understand delivering a smooth learning curve. The extension of webAD with new components for an average experienced software developer, as computer science students in the Bachelor phase, was evaluated as a straight forward and relatively easy accomplishable task. Criticised was the missing of strict functionality guidelines for the implementation of the tape-recorder and the general control of the algorithm execution. Hereby webAD provides too much freedom to the developer, so that the different components show a somewhat uneven interface. This was obviously also a result of efforts of the developers to highlight specific algorithm characteristics. Further it was complained that the choice of color used in the visualization of the different algorithm and data structure component and the control of the color map was not uniformly implemented. Finally it was recommended to integrate webAD more strongly with the course material of the accompanying algorithm and data structure course of the Bachelor curriculum. For example to insert links at the specific components to the respective chapters of the course slides.

8. CONCLUSIONS AND FUTURE WORK

webAD demonstrates, that our design solution fully supports our goals. The visualizer is already used in the Algorithms and Data Structures lecture, fulfilling the main task of supporting students and teachers. The platform is accessible using any desktop or mobile device and any browser and received some positive acclaims from students. At this stage, while a major phase of development is completed, there are no unresolved issues.

In the coming semesters we plan to arrange a large-scale survey testing webAD. The students taking the Algorithms and Data Structures course at the University of Vienna are going to be split into two groups. The first group is going to utilize webAD and the second one classical static lecture slides for the exam preparation. After evaluation of the grades we intend to assess the efficiency of webAD compared to traditional learning methods. Further student contributors will increase the number of modules within the platform.

More information on the project and its source code are freely accessible at
`http://gruppe.wst.univie.ac.at/workgroups/webAD`.

9. ACKNOWLEDGMENTS

We would like to thank specifically Helmut Maderbacher, Georg Prenner, Alexander Rotheneder, Sylvia Schikuta and Kirilo Begy, who contributed to webAD and to preceding projects, as VADer, LUCAS and NetLuke, easing the educational entry for CS students into the amazing domain of algorithm and data structure design.

10. ADDITIONAL AUTHORS

Steve Aumüller (email: `steve.aumueller@gmail.com`),
Laura Fagagnini (email: `a1302472@univie.ac.at`),
Robert Habetinek (email: `r.habetinek@hotmail.com`),
Bernhard Hirsch (email: `a1269039@unet.univie.ac.at`),
Kathrin Kronfuß(email: `kathrin@kronfuss.at`),
Amolkirat S. Mangat (email: `a1125776@unet.univie.ac.at`],
Yasmin Tarasiewicz (email: `yt1@gmx.at`) and
David Tomic (email: `david.tomic11@gmail.com`)

11. REFERENCES

[1] V. Begy. Webad: Visualizing algorithms and data structures. master thesis, Faculty of Computer Science, University of Vienna, 2015.

[2] R. C. Clark and R. E. Mayer. *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. John Wiley & Sons, 2016.

[3] J. S. Dumas and J. Redish. *A practical guide to usability testing*. Intellect Books, 1999.

[4] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[5] T. Möller. Encode: Single View Methods. Lecture Slides to Course Databases and Processing of Large Data Sets, University of Vienna. Accessed July 18, 2015.

[6] G. Prenner, A. Rotheneder, and E. Schikuta. Netluke: Web-based teaching of algorithm and data structure concepts harnessing mobile environments. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, pages 7–16. ACM, 2014.

[7] E. Schikuta and H. Maderbacher. Web-based visualization of algorithms and data structures. In *IASTED International Conference Applied Informatics (AI 2001)*, Innsbruck, Austria, 2001. IASTED.

[8] E. Schikuta and S. Schikuta. Lucas - an interactive visualization system supporting teaching of algorithms and data structures. In *World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA'09)*, page 3776–3781, Honolulu, Hawaii, USA, 2009. AACE.

APPENDIX

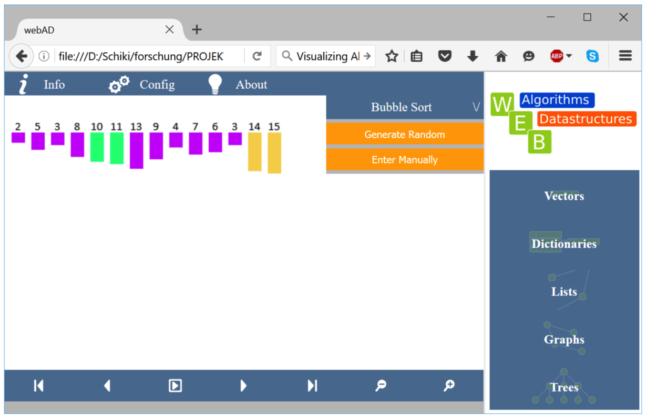


Figure 8: Bubblesort.

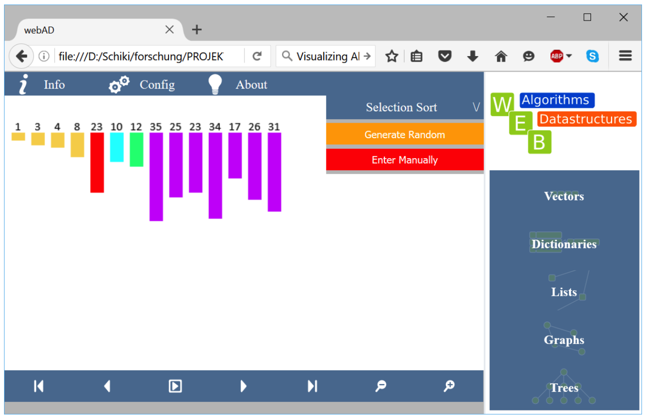


Figure 9: Selectionsort.

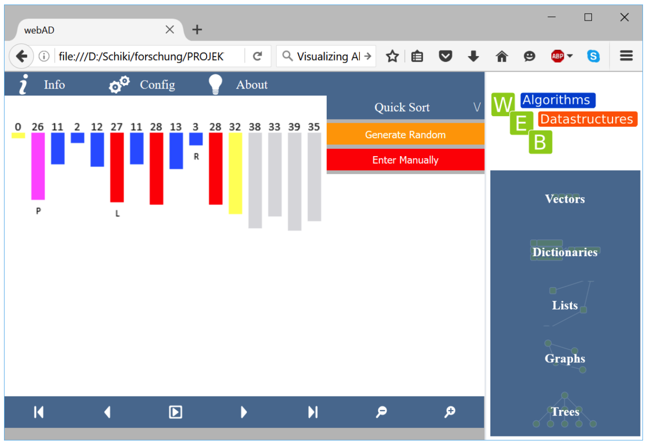


Figure 10: Quicksort.

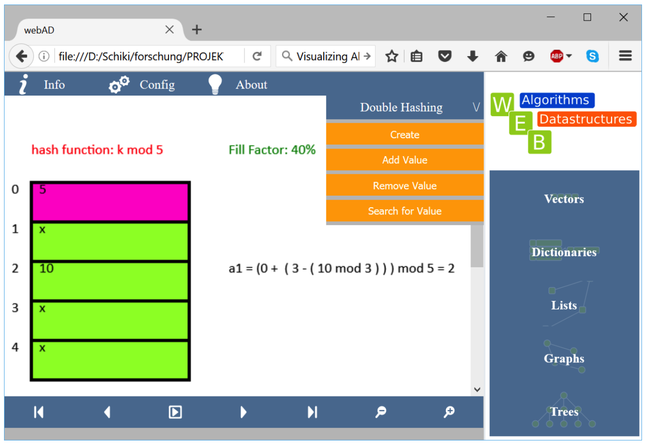


Figure 13: Double hashing.

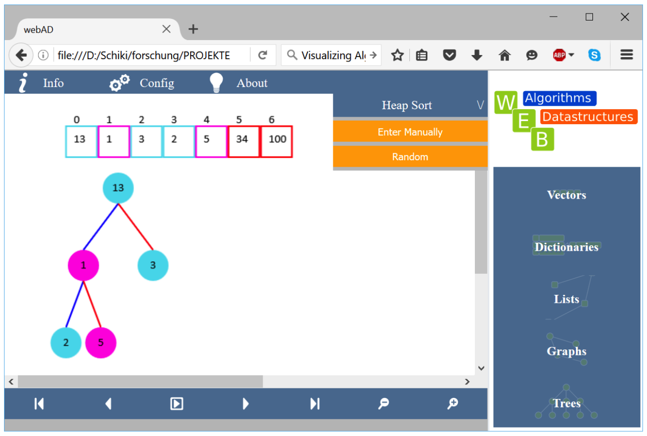


Figure 11: Heapsort.

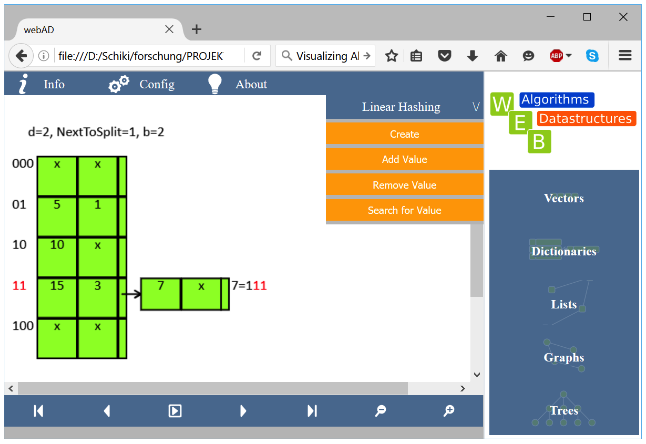


Figure 14: Linear hashing.

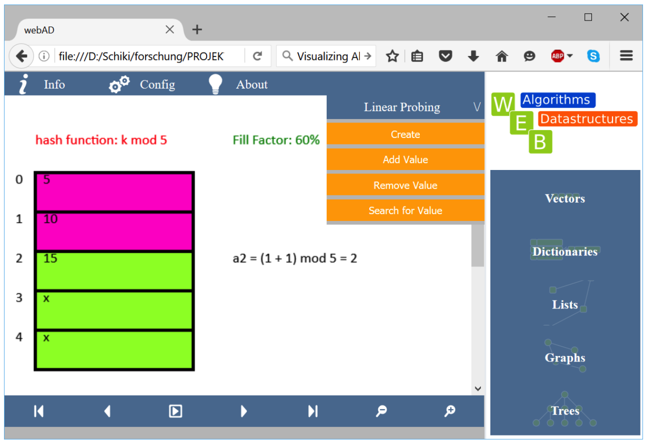


Figure 12: Linear probing hash.

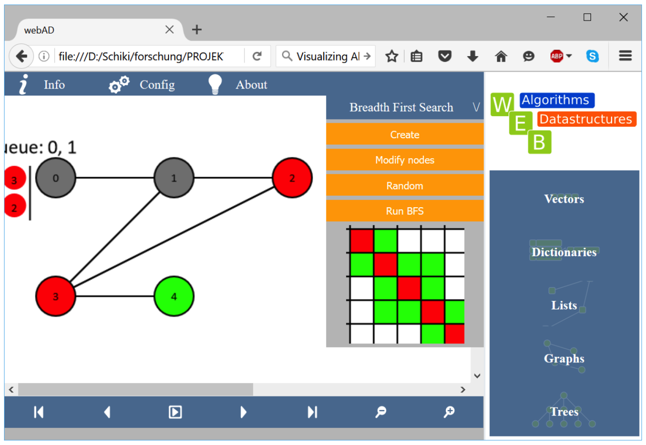


Figure 15: Breath first search.

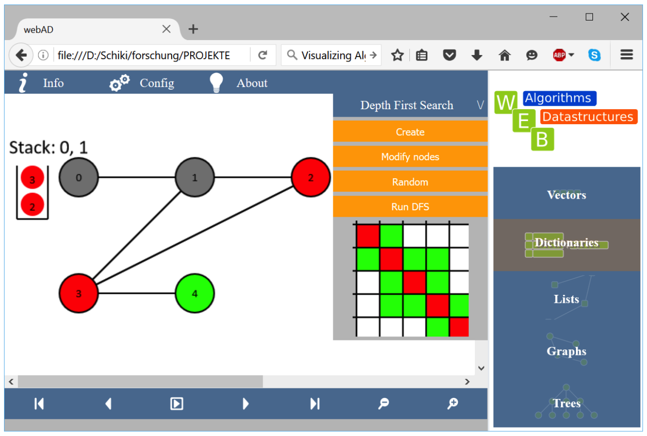


Figure 16: Depth first search.

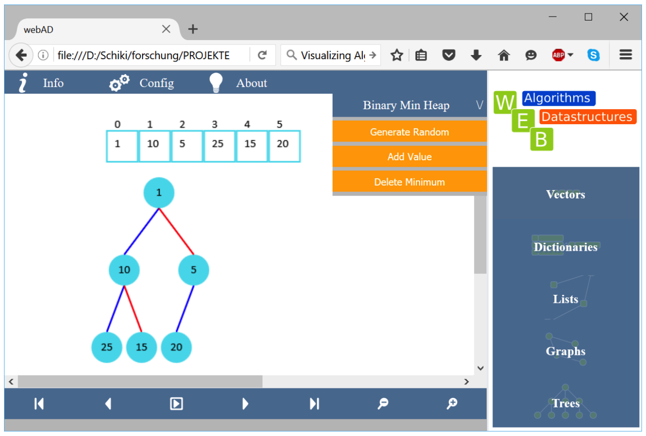


Figure 19: Binary minimum heap.

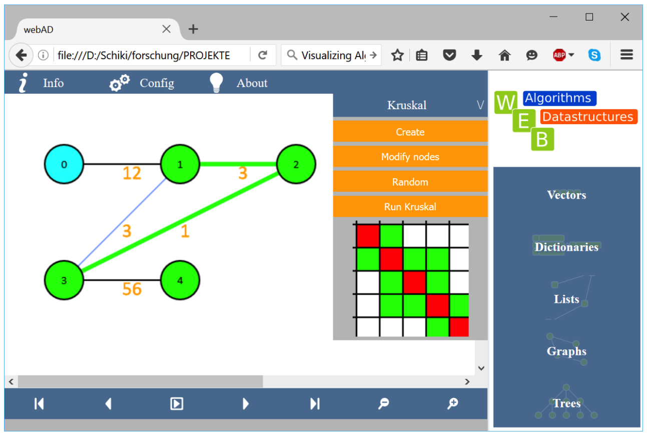


Figure 17: Kruskal minimum spanning tree algorithm.

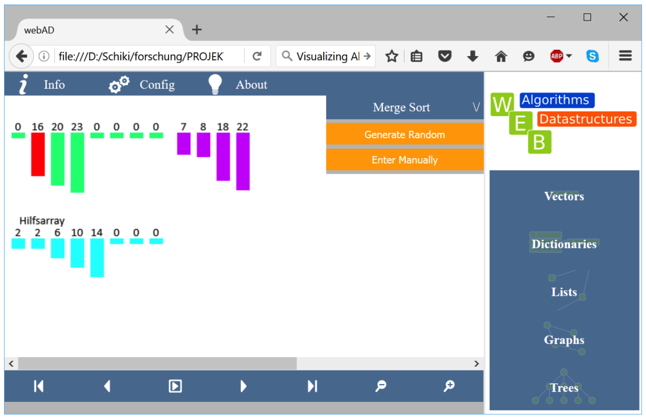


Figure 20: Mergesort.

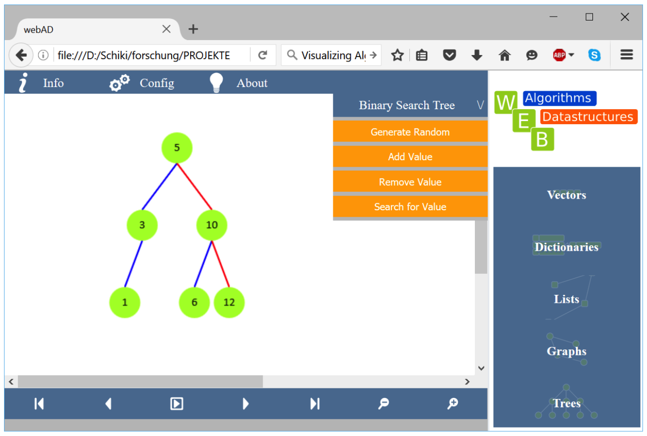


Figure 18: Binary search tree.

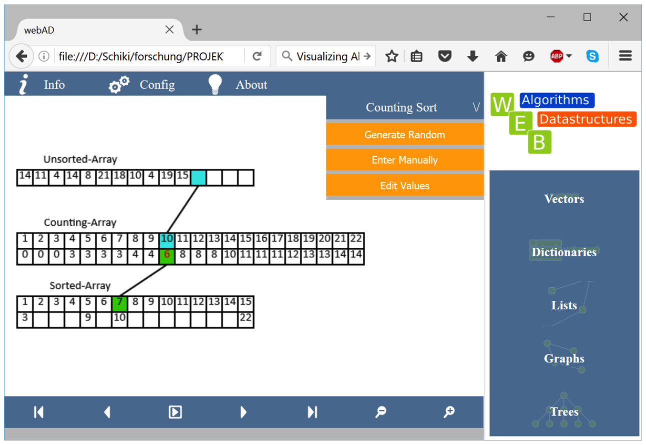


Figure 21: Countingsort.

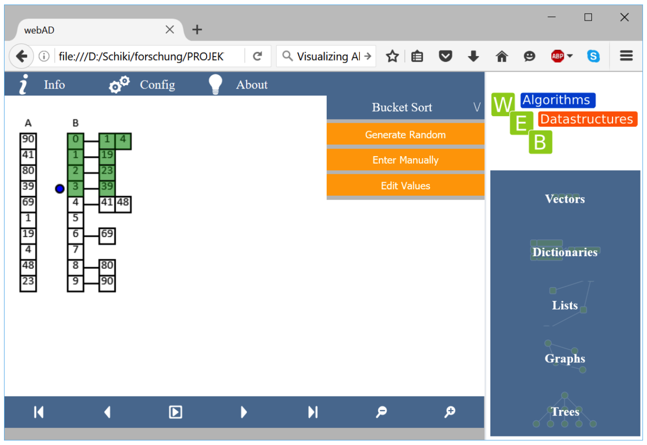


Figure 22: Bucketsort.

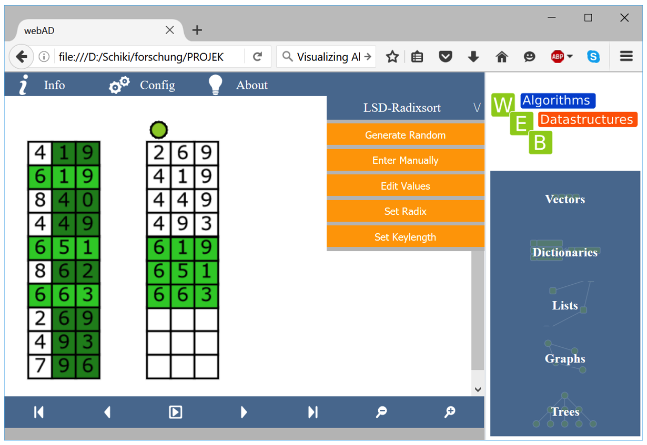


Figure 23: LSD Radixsort.

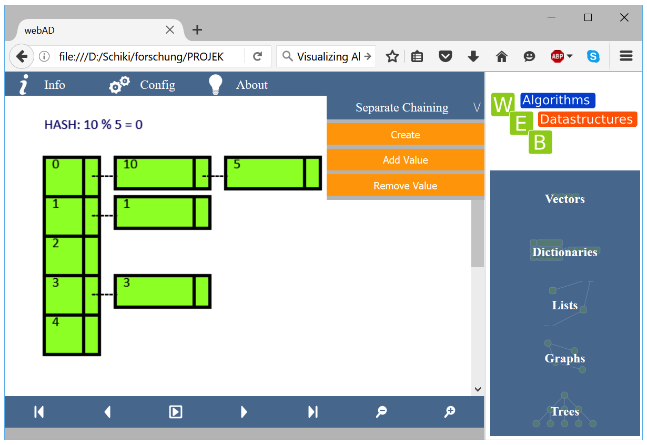


Figure 24: Separate Chaining Hash.

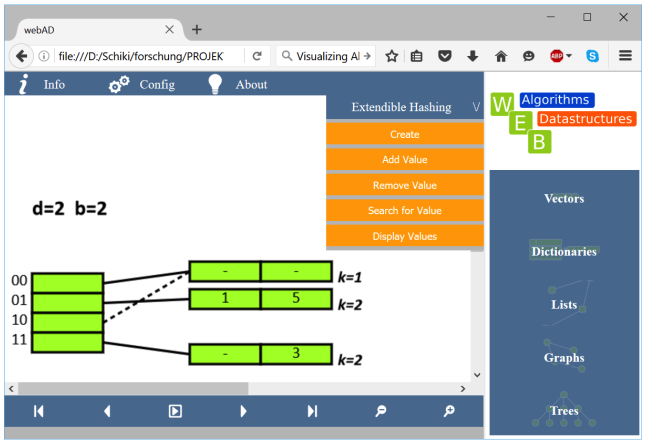


Figure 25: Extendible Hash.

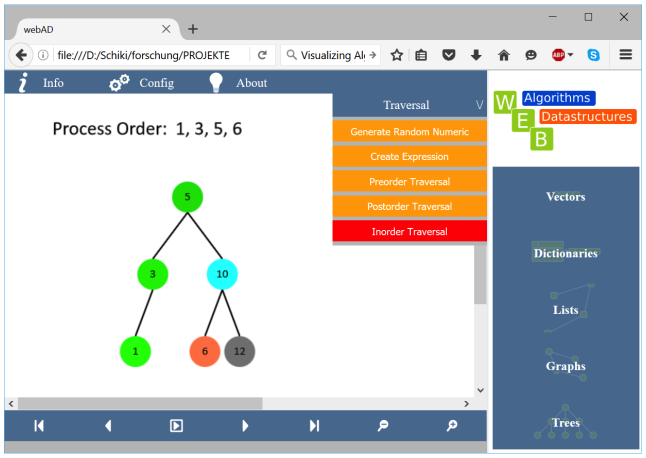


Figure 26: Tree Traversal.

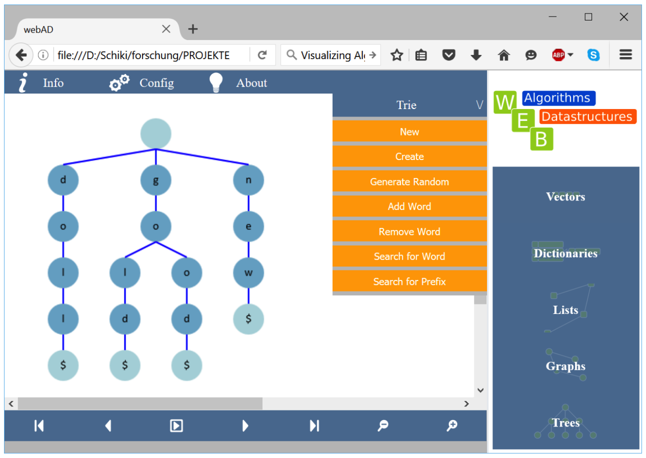


Figure 27: Trie.