# Contents

1 Introduction ........................................ 1
   1.1 Motivation ....................................... 5
      1.1.1 Comprehension of algorithms ............... 5
      1.1.2 Manipulation of algorithms ............... 7
   1.2 Proposal .......................................... 7
   1.3 Goals ............................................ 9
      1.3.1 Main Goal .................................... 9
      1.3.2 Specific Goals ............................... 9
   1.4 Thesis Outline ................................... 10

2 Related Work ........................................ 12
   2.1 Algorithm Visualization ......................... 12
      2.1.1 Pedagogical benefit ......................... 12
      2.1.2 Relationship between Implementation and Visualization 15
      2.1.3 Algorithm manipulation ..................... 18
   2.2 Concept Keyboards ............................... 18
   2.3 eXtensible Markup Language (XML) ............... 21

3 Requirements ........................................ 24
   3.1 Use multiple scenarios ......................... 24
   3.2 Use existent implementations .................... 25
   3.3 Generation of the Concept Keyboard ............. 27
4 Concept Keyboard Software

4.1 Configuration of the Concept Keyboard

4.1.1 Generation of Data XML file

4.1.2 Generation of Keyboard XML file

4.2 Algorithm Visualization

5 Implementation

5.1 Data XML file

5.2 Keyboard XML file

5.3 Abstract class GraphObject

5.4 Main application

5.4.1 Configure Application

5.4.2 Concept Keyboard Application

6 Algorithms

6.1 Algorithm Classification

6.1.1 Graph Algorithms

6.1.2 Tree structures

6.1.3 Sorting algorithms

6.1.4 Search in text

6.2 Algorithms implemented

6.2.1 Graph algorithms

6.2.2 Tree algorithms

6.2.3 Sorting algorithms

7 Tests and results

7.1 Tests in Santiago, Chile

7.1.1 Testers

7.1.2 Algorithms tested

7.1.3 Steps

7.1.4 Questionnaires

7.1.5 Exercises
CONTENTS

7.1.6 Test Results ................................. 90
7.1.7 Analysis ................................... 95
7.2 Improvements to the Software .................... 96
7.3 Tests in Druisburg, Germany ..................... 97
  7.3.1 Testers ................................... 97
  7.3.2 Algorithms tested ......................... 98
  7.3.3 Steps ..................................... 98
  7.3.4 Questionnaires ......................... 99
  7.3.5 Test Results ......................... 100
  7.3.6 Analysis .................................. 101

8 Discussion ..................................... 102
  8.1 Summary .................................... 102
  8.2 Future Work .................................. 104
  8.3 Conclusion .................................. 105
Chapter 1

Introduction

There is a growing interest in algorithm visualization (AV) to help instructors to explain and learners to understand algorithms. A certain algorithm animation shows relevant parameters and variables, the current state and a visual representation of the objects being manipulated, and often an animated formal description of the algorithm. Complex model structures are simplified in order to highlight only the important aspects by omitting non-relevant details. To achieve better comprehension, the designer scales down data to coarser structures and slows down algorithms that process data. Smooth transitions between different states of moving objects, for example, elements inside an array while performing a sorting algorithm, can help to follow the way it works on graphic representations of data structures.

In most cases algorithms work only locally on data structures in a serial way. So learners can run the algorithm step by step forward and backward selecting different input parameters, step size and execution speed while exploring interactively on their own a configurable animation of algorithms.
CHAPTER 1. INTRODUCTION

The intention of using this kind of interactive software is to gain insight in the dynamic behavior of the algorithm.

Conventional algorithm visualization systems give the user a limited control over the visualization of the algorithm or data structure. Usually they give the user the possibility to initialize the visualization by specifying the type of visualization, size of the probe, speed of the animation and granularity of the executed steps. During the execution of the animation, the system allows the user to reset the algorithm, go forward and backwards. For example, in a visualization of the Kruskal algorithm (**)(figure 1.1), there are three available buttons. The “Solve” button executes the entire algorithm and shows the user the final result for determining the minimum spanning tree. The “Step Solve” button executes the next step in the algorithm showing the user the new state that will be reached by the data structure. Finally, the last available button is “New Problem” which allows the user to start with a new set of data.

Another way of presenting the algorithm visualization is by giving the user a wider range of available actions, or functionalities, that can be executed over the algorithm or data structure. Instead of only allowing the user to move step-by-step over the execution of the algorithm, the algorithm visualization software may give the user the possibility to manipulate the data structure and alter the execution of the algorithm. The actions that the user can execute correspond to the functionalities that are available in the algorithm. In the general form, they correspond to the methods (functions) that exist in the corresponding implementation of the algorithm or data structure.

For example, in an AVL tree algorithm, the possible actions could be “simple rotation to the left”, “simple rotation to the right”, “move to left
Figure 1.1: Example of a step-by-step interface. The only possible actions are to solve the entire problem ("Solve" button), solve it step-by-step ("Step Solve" button), or show a new problem ("New Problem" button)
child”, “move to right child”, etc. This approach gives the user the possibility to explore, through experimentation, the characteristics of the algorithm or data structure giving him/her a better understanding of it. The exploration allows the student to get involved with the execution of the data structure and become an active participant in the development of the algorithm and not just an observer.

We propose that by giving the user the tools for experimenting and manipulating the algorithm or data structure, he/she will achieve a better comprehension of the algorithm in study.

The interaction between the user and the algorithm can be done with the use of a traditional keyboard present on every computer. The problem with these kind of keyboards is that the keys does not give the user the meaning of the functionality that will be executed by pressing them. Each key has a label that lets the user to differentiate them, and can determine which key must be pressed, but the problem of this approach is that there is no relationship between the key and the action that will be executed as a consequence of pressing it. This is one of the reasons for introducing the use of the concept keyboard. On a concept keyboard, each key has associated a special meaning (concept) instead of just a label. Usually a concept keyboard has only the necessary keys, depending on the use that will be given to the keyboard, and can be customized with additional information to the user. For example, instead of just a label, drawings can be associated, or special textures. Figure 1.2 shows an example of a concept keyboard and some layouts that can be used over the keyboard.
Figure 1.2: Example of a Concept Keyboard. The concept keyboards can use “overlays” to enable using different layouts of the keys.

1.1 Motivation

“I hear and I forget, I see and I remember, I do and I understand.”
Confucius

1.1.1 Comprehension of algorithms

Algorithms and data structures are the building blocks of computer-based systems, and essential in any computer science’s study plan. Since algorithms and data structures have abstract concepts that are difficult for a student to comprehend, there has been developed algorithm visualization systems that represent, in a more concrete way, the characteristics of algorithms and data structures.

An algorithm can be described as a set of rules that help to determine
what decision must be made in the next iteration, in order to solve a specific problem. Current algorithm visualization systems consists on presenting a graphical representation of the algorithm or data structure and allow the user to go step-by-step over the execution of the algorithm. This approach helps the student to understand what is the following step (based on the “rules”) that must be executed, considering the current status of the data structure.

One of the main difficulties that a student has when he/she is studying an algorithm, is to understand how an algorithm works, and not so much how it is implemented or what are the set of “rules” (steps) that are necessary to follow in order to get the correct result. Studies made related to the pedagogical benefit of algorithm visualizations, show that better understanding of algorithms and data structure can be achieved by allowing the student to get involved in the execution of the algorithm. Hundhausen et al. (***) conclude that “what learners do, not what they see, may have the greatest impact on learning”. The results obtained by Grissom et al. (***) in the study of the performance of similar group of students with different engagement levels with the algorithm visualization show that better comprehension is achieved by incrementing the level of engagement.

To help the student understand how an algorithm works, it is necessary to get them involved in the execution of the algorithm. He/she must be able of exploring the algorithm or data structure, and experiment with the operations available. By allowing the student to navigate through the data structure and explore the possible actions available, he/she will gain better comprehension of how the algorithm or data structure works.
1.1.2 Manipulation of algorithms

The interaction between the student and the algorithm or data structure is an important issue to consider in the manipulation of the algorithm. It seems interesting to enhance the way the student controls the visualization by creating a direct relationship between the operation that will be executed and the user interface. Using a concept keyboard instead of conventional keyboards seems to be useful in the process of understanding the corresponding algorithm or data structure. The ability of using multiple overlays representing different configuration of keyboards, or with graphical elements describing the related operation, can help the student to focus on understanding how the algorithm works.

The use of a concept keyboard makes possible to extend this work, and allow the use of algorithm visualization software to people with sensory disabilities, by enhancing the way the student interacts with the visualizer. This issue will not be covered on the present work, but future research can be done in this area.

1.2 Proposal

The proposal presented on this thesis is that the student can achieve a better comprehension of the algorithm or data structure by navigating through the data structure. This exploration, by experimenting with the available operations the algorithm has, can help the student to understand how the algorithm works.

Every algorithm or data structure has different operations available, and
CHAPTER 1. INTRODUCTION

the student must be presented with different keyboards, depending on the characteristics of the algorithm and on the focus that the instructor gave to the study plan. The generation of the concept keyboard must be an automatic process based on the characteristics of the algorithm. The configuration of the keys can be customized by the instructor, that acts as a designer, determining the arrangement of the keys, considering usability tests, and relationship between the concepts. The operations available have concepts related to them, and therefore, keys must be closer together to the keys that have a “similar” concept associated.

When the keyboard is configured and customized by the instructor, the student can visualize the algorithm, or data structure, and interact directly with it. This interaction consists on executing freely the operations (actions) that are available, according to the algorithm or data structure in study, trying to solve the problem presented. The student can navigate through the data structure, and execute any of the available operations. This liberty for executing the actions in the order he/she considers necessary permits him/her to understand better why executing certain operations can lead to the correct, or incorrect result. The student has the possibility to make mistakes, since the order in which the actions are executed is determined by his/her decision, allowing the student to have a better comprehension on the characteristics of the algorithm. We propose that the possibility of deciding the actions to be executed must be accompanied with the possibility of making “undo”, so the student can return to a previous state and use a different action.
1.3 Goals

1.3.1 Main Goal

The main objective of this thesis is to determine if the students will obtain a better comprehension, of the algorithm or data structure in study, by allowing him/her to explore and manipulate the data structure or algorithm. Enhancing the way the student controls the visualization of the algorithm or data structure will allow him/her to have better insight of the characteristics of the algorithm.

1.3.2 Specific Goals

In order to accomplish the main goal of this thesis, it is necessary to complete the following specific objectives:

1. Determine the requirements and characteristics of an algorithm visualizer that permits the student to manipulate directly the algorithm and data structure.

2. Design and develop a software that allows the instructor to:
   
   (a) Adapt an existent algorithm visualization software.
   
   (b) Create and customize the corresponding concept keyboard.

3. Design and develop a software for visualizing an algorithm that allows the student to:
   
   (a) Visualize an algorithm or data structure.
(b) Interact with the algorithm or data structure with the help of a concept keyboard.

4. Design and execute tests for determining the benefits achieved by using the new algorithm visualization software, in comparison with the tools that exists nowadays.

1.4 Thesis Outline

The previous sections present an overview of the contents of this thesis. This section outlines the structure of the thesis and gives a brief description of each chapter.

Chapter 2 presents the current state-of-art of the work related to this thesis. A description of the work other authors have done in relation to *Algorithm Visualization*, *Concept Keyboards*, and *eXtensible Markup Language* is presented.

Chapter 3 presents the characteristics and requirements that must be satisfied in order to prove the main goal of the thesis.

Chapter 4 describes the characteristics of the software developed. This software is used to test the proposed approach for visualizing algorithms, allowing the instructor to create and configure the concept keyboard, and the student to interact with the algorithm or data structure.

Chapter 5 presents the details of the design and implementation of the software described on chapter 4.
Chapter 6 presents a description of the algorithms that were used, as well as a description of the proposed concept keyboard associated to each of the algorithms.

Chapter 7 outlines the tests that were conducted in Santiago, Chile, as well as the tests conducted in Drusiburg, Germany.

Chapter 8 presents the conclusion of the current work. This conclusions are based on the results obtained from the tests that were executed, and from the observations made during the development of the thesis project. It is also presented the future work that can be executed, based on the present work.
Chapter 2

Related Work

2.1 Algorithm Visualization

Many researchs has been done in order to analyze the advantages and disadvantages of algorithm visualization as well as how to connect the visualization with the algorithms implemented. The study of the related work to algorithm visualization that has been done will be separated in three subjects: pedagogical benefit, relationship between implementation and visualization, and algorithm manipulation.

2.1.1 Pedagogical benefit

Algorithm visualization started to be used in late 1980’s in order to create and interactively explore graphical representations of computer science concepts. Since then, there has been the discussion whether algorithm vi-
visualizations provide a pedagogical benefit to the student or not. Most of the researchers agree that algorithm visualization software can improve the comprehension level of the students, but some researchers argue that there are no substantial results that can validate this statement.

Hundhausen and Douglas (*1*) classify in two categories the experiments that have been done in order to test the pedagogical benefit of using AVs. The first category analyzed the representational characteristics of the study material. They considered the study of text versus animation, text-first or animation-first, and different graphical attributes of animations. The second group varied the level of learner involvement in the visualization of the algorithm. These studies allowed more involvement of the student by allowing them to construct their own data sets, answer specific questions about the algorithm, make predictions of the future behavior of the algorithm, or program the algorithm or data structure. Based on the results of these experiments, Hundhausen observed that the experiments that manipulate the level of involvement of the students have significantly better results than the experiments that manipulated the graphical visualization. He concludes that algorithm visualization software do improve the comprehension level of the students, and specifically “what learners do, not what they see, may have the greatest impact on learning”.

The observations made to the experiments support our hypothesis that in order to achieve a better comprehension, it is necessary that the student must be able of having an active participation with the algorithm. The classical ways of visualizing algorithms give the user a limited control over the algorithm or data structure, and do not give the user the necessary level of involvement with the execution of the algorithm.

In order to analyze the pedagogical benefit of different algorithm ani-
CHAPTER 2. RELATED WORK

In algorithm visualization systems, the Working Group on Improving the Educational Impact of Algorithm Visualization, defined an *engagement taxonomy* of the level of engagement of different AV (*7*). This taxonomy defines six levels of engagement:

**No viewing** Instruction without using any AV.

**Viewing** The most passive level of visualization. It only allows the student to observe the visualization.

**Responding** It presents the students with a set of questions that must be answered during the visualization of the algorithm or data structure.

**Changing** Considers modifying the visualization by the student. An example is the modification of the input data of the algorithm in study.

**Constructing** Learners can construct their own visualization of the algorithm or data structure in study.

**Presenting** Entails the student to make a presentation of the visualization to an audience for observations and discussion.

The Working Group propose to use this classification in the analysis of existent algorithm visualization systems, in order to determine the comprehension achieved by the learners.

Based on this taxonomy, Grissom et al. (*8*) present a study that compares the performance of three treatment groups that have different level of engagement with algorithm visualization. They focused on the “No viewing”, “Viewing”, and “Responding” levels presented on the taxonomy. To determine the improvement on the performance of the students, they compared the results obtained on a pre-test with the results of a post-tests. The
three group of students attended a one class session lecture that presented bubblesort, selection sort, and insertion sort algorithms. In order to have a similar background, each professor used the same lecture slides to present the algorithms and students were given a printed study material. After one week, the students had to answer the post-test. After analysing the results, they observed that the three groups presented improvement on the results obtained between the pre- and post-tests. The relevant observation was that the difference of improvement between not viewing a visualization and interacting with one was significantly better on the last group. They concluded that learning improves as the level of student engagement with algorithm visualization increases.

The results obtained on the Grissom studies agree with the purpose of the present work. Better comprehension is achieved by giving the student more than a simple visualization of the algorithm or data structure. Besides an introductory presentation of the algorithms, students can obtain better understanding on the way the algorithm works by complementing the classes with an interactive visualization of the algorithm or data structure. The visualization must allow the student to manipulate the data structure and modify the input data used for the algorithm.

2.1.2 Relationship between Implementation and Visualization

On every algorithm visualization system, there exists the problem of determining how to connect the visualization with the actual implementation of the algorithm or data structure. In the work of Demetrescu et al. (*2*), the authors present a comparison between the two most common approaches for linking the implementation with the visualization. They compare the
Event-driven approach versus the Data-driven approach. The first approach consists on identifying interesting events in the implementation code that correspond to relevant actions in the execution of the algorithm, and map these events with a graphical representation. For example, in a sorting algorithm, the action of swapping two elements can be considered as an interesting event. The second approach consists on observing how the relevant variables of the algorithm change, and create a graphical representation that describes the state of the algorithm. An example of this approach are the conventional debuggers, where it is possible to understand the state of a program by observing how the variables change over time. They specified the visualization of the Bubblesort algorithm, with each of these approaches and they were able of identifying relevant differences between these two approaches. The event-driven approach seemed very intuitive and useful for highly customized animations, but they require developers to write several extra lines of code, and do not allow code ignorance. On the other hand, the data-driven approach require developers to write few lines of extra code, and little knowledge of the underlying code is needed to specify the animation. A problem with this last approach is that it has a steeper learning curve, and seems to be less flexible.

There have been developed several systems based on specifying interesting events. One of the first systems that used this approach was BALSA (**), and then it successor BALSA II (**), extended it by allowing to specify steps and break points. Recent work, as the framework Ganimal (**), use the event-driven approach to specify the relevant events associated to an algorithm or data structure. After specifying the interesting events, it presents a visualization of the algorithm or data structure. To identify the interesting events, it uses an animation description language, Ganila, which extends the Java programming language. These annotations are then compiled with the use of a modified Java compiler and linked with the existent
implementation. There exists a graphical user interface that helps the specification of these settings. An example of using the Ganimal framework is presented on Kerren’s research (*5*) where he used Ganimal to visualize animations of complex computational models for compiler design.

In the present work, it will be used the event-driven approach for connecting the visualization with the implementation of the algorithm or data structure. Each key of the concept keyboard will be mapped to the execution of an existent method available in the algorithm implementation. In order to choose the interesting events, the designer has a simple graphical user interface that displays the available actions (methods), and allows him/her to select the relevant actions. Since, one of the intentions of the work is to allow the use of existent implementations, we propose that the selection of the interesting events should be done without modifying the implementation of the algorithm.

A disadvantage of using the approach used by Ganimal, is that it requires additional software (the Ganila compiler) besides the Java Runtime Environment. Another disadvantage is that the specification of the interesting events is based on writing extra lines of code, using a new programming language (Ganila), making more difficult to create new animations. The main advantage of Ganimal is the wide range of features that are available. Besides specifying the interesting events, it allows to specify alternative code blocks, alternative interesting events and runtime modification of the animation.
2.1.3 Algorithm manipulation

Faltin (*4*) presents a system for visualizing algorithms and data structures using an approach similar to the one presented in this work. He proposes that the instructor should decompose the algorithm into many small functions, and present a call graph for the functions. The student is presented with a description of every function, and a visualization of the data structure. Then, the student task is to discover, by experimenting, the correct steps in order to execute the algorithm. He states that it is necessary to have a mechanism for backtracing the steps executed in case the user realizes that he/she has made a mistake. Besides the algorithm visualization, the student can query a tutorial text which explains the right solution and can watch an animation of the algorithm.

In our work, we give the user the capability of executing different actions in a free way. Each of these actions correspond to the execution of a method of the algorithm or data structure. In case the student identifies that he/she has performed an incorrect decision, he/she is able of making an “undo” and execute a different action. Since there can be many interpretations of the intention of the algorithm and the problem to which it is intended to solve, it is important to give the user an explanation of the characteristics of the algorithm or data structure.

2.2 Concept Keyboards

The Concept Keyboard consists on a flat touch-sensitive surface with 128 or 256 cells arranged in a rectangular array. These cells can be grouped in order to create associated regions that act as a single cell. Each of these cells,
or group of cells, can be described as a key in the keyboard, and associate an action that must be executed when they are pressed. For example, the keys can be linked to a pre recorded sound that plays when the key is pressed, or a particular animation is presented when a portion of the keyboard is selected.

A useful feature of the concept keyboards is the possibility to superpose special boards (overlays) over the concept keyboard. These overlays allow the user to relate the action that will be executed with a particular image and/or a special kind of surface.

The VICTAR (Visual Impairment Centre for Teaching And Research) center of the University of Birmingham developed the “Sound Book” software (**). This software uses a concept keyboard so when particular areas of the keyboard are pressed, a digitized sound is played. They use tactile overlays that are placed over the keyboard and children can move their fingers over the overlay and press special areas of the keyboard marked as “nodes”. At this center they have designed several activities that allow the student to listen to stories and interact with the development of it. Other activities are related to determining the corresponding animal or transportation system, based on the sound that is played. Since they work with visually impaired children, the objective is to help in the learning of tactual reading by making the student to move his/her fingers from left to right along a tactile path.

The TIM (Tactile Interactive Multimedia) project (**), developed at the Universit du Havre and directed by Archambault and Burger, is dedicated to create and adapt computer games for blind children. The main purpose of the project is to allow the children with visual impairment to use the computer in an autonomous way. In this project, they use a concept keyboard with different overlays that allow the children to interact with the
computer game. These overlays have pictures that are carved in relief, or words written in Braille. The children can play the games by pressing different areas of the concept keyboard, depending on the desired action they want to execute. They tested the games with children of 3-6 years old, and from the results it was possible to extract the personal appreciations of the children and their parents, in relationship to the concept keyboard:

“[…] The Concept Keyboard happens to be of particularly well adapted use for children who have not learned to use a keyboard yet.”

“The Concept Keyboard has a big advantage: it allows an entirely independent use of the software, and learning it is quite easy even for children who cannot read […] The equipment is very interesting and can bring much to the learning for the young blind child.”

These results confirms our proposal of using a concept keyboard to allow better interaction between the user and the software. By using different overlays on the keyboard it is possible to give the user new ways of interacting with the algorithm or data structure. In the Sound Book project, they use different layouts depending on the story the software is presenting. In a similar way, it is possible to display different layouts of keyboards, with different actions associated to each key, of the algorithms and permit the user to navigate through the data structure.

A problem presented in the Sound Book project is the time that the teacher must dedicate for designing and creating new interfaces. This is a problem can be generalized to any generation of a new concept keyboard.
As part of the present work, it will be presented a semi-automatic way of generating and configuring the corresponding concept keyboard for each algorithm or data structure. The possibility of having multiple keyboards to each algorithm allows the teacher and the student to analyze the concepts that are present on the execution of an algorithm.

### 2.3 eXtensible Markup Language (XML)

The *Extensible Markup Language* (XML) is a simple and very flexible text format. It is used as a metalanguage, allowing the user to specify a set of customized tags that define the particular data. This flexibility permits to exchange information in a standard way between several entities. The specification and recommendations regarding the use of XML is directed by the World Wide Web Consortium (W3C).

The W3C presents a description, in ten points, of the characteristics of XML (**):

1. XML is for structuring data.
2. XML looks a bit like HTML.
3. XML is text, but isn’t meant to be read.
4. XML is verbose by design.
5. XML is a family of technologies.
6. XML is new, but not that new.
7. XML leads HTML to XHTML.
8. XML is modular.

9. XML is the basis for RDF and the Semantic Web.

10. XML is license-free, platform-independent and well-supported.

On the present thesis, it will be used XML files for describing the algorithms and data structures in study, as well as their corresponding concept keyboards. The reasons for using XML files instead of other forms for storing information in files, can be summarized by explaining the points (1), (3), (4) and (10) from the previous list.

**XML is for structuring data.** On the present work it is necessary to have a way of describing the characteristics of the algorithms or data structures. The characteristics of every algorithm or data structures (e.g., class name, methods, parameters, etc) can be specified by a small set of identifiers. Since the algorithms are written using the same programming paradigm (in this case Object Oriented), it is possible to define group of rules that describe an algorithm. Similarly, the characteristics of the generated concept keyboards will be common to every concept keyboard created.

**XML is text, but isn’t meant to be read.** The XML files are human-readable files that can be examined easily in order to check the characteristics of the description files. Although the processing of the XML file will be done by a software, it enables the teacher to read the file.

**XML is verbose by design.** XML files can become long files, containing multiple nested tags, just to describe a small set of characteristics. Since the processing of the XML files will be in charge of a computer software, and the file will be used mainly for storing information, the fact of having long files is not a problem.
Finally, XML is license-free, platform-independent and well-supported. The software that will be developed in order to prove the hypothesis presented in this work will be written completely in Java. One of the main features of the Java programming language is being platform-independent, so it is mandatory to have a platform-independent way of storing the required information.
Chapter 3

Requirements

In this chapter, it will be presented the requirements that the software must meet in order to achieve the objectives presented on previous chapter.

3.1 Use multiple scenarios

One of the basic concepts on which this thesis is based is that by allowing the student to experiment by themselves how the algorithms or data structure works, will permit the student to achieve a better comprehension of the algorithm in study. Besides experimenting with different algorithms, it is necessary to experiment with different scenarios and understand by exercising how the algorithm works in different situations. Some algorithms have better performance with some kind of initial states, and not so good performance with other initial states. These differences must be accessible for the student in order that he is able of determining the pros and cons of
each algorithm and be able of deciding which situations are better for using certain algorithm.

This is the reason why the system must be capable of adapting to algorithms that implement the functionality of using a file that describes the initial state of the algorithm or data structure. For example, in the implementation of an algorithm for sorting a set of data, it could be useful that the student can interact with the algorithm in cases where the data is completely in disorder, in ascending order, in descending order, etc.

To generalize this idea, we introduce the concept of startup file. This is a file that describes the initial state of the algorithm or data structure, and can contain additional information particular to the implementation. The structure of this startup file is left free to the criteria of the person that implements the algorithm or data structure, since it will be used exclusively by the algorithm.

### 3.2 Use existent implementations

The system must be able to use existent implementations of algorithm or data structure visualization, with only few modifications to the way they were implemented. To achieve this, it is necessary that all the implementations of algorithm visualization have some common criterias in the way of writing the code. It was identified four restrictions that must be satisfied.

The first consideration that must be made to the way the algorithm is implemented is that methods should be implemented considering the object they reference as an intrinsic parameter. Instead of using methods that receive the object on which they operate as an argument, eg. “rotate(x)”,
CHAPTER 3. REQUIREMENTS

the methods should be implemented as “rotateLeft()” and “rotateRight()”. These last methods operate over the object to which is sent the message, i.e. the this object. This restriction may force to modify the way the algorithm was implemented, but it is not a radical change in the way of implementing the algorithm.

The second restriction that must be satisfied is that each object used as an argument for the available methods must have a constructor that receives a String as an argument. This restriction allows the value of the elements of the data structure to be set in a more general way. An exception for this restriction is when the method receives variables whose type is one of the primitive types that exist in Java. Every primitive type has a wrapper class that permits the construction of an object using a String, and then convert this object into a value of primitive type. This exception makes easier the conversion of an implemented class into a class that can be used for the visualization with a concept keyboard.

The third consideration is that every implementation must have a method in charge of performing the graphical representation of the algorithm or data structure. Since the algorithm visualization software is based on having a graphical representation which is extended with the use of a concept keyboard, it is essential that the implementation is capable of displaying in a graphical way the representation of the algorithm. The way of accessing this method must be common to every implementation in order to be able of making calls to this method every time the system determines it is necessary to refresh the representation. This restriction can be satisfied by defining an abstract class that must be extended by the class implementing the algorithm, in which there is the definition of the way of accessing the graphical representation.
The last consideration is that it is necessary to standardize the way the algorithm receives the startup file that was selected by the user. This restriction is only applicable to the algorithms that implement the use of a startup file. To standardize this, it is possible to include this restriction in the abstract class mentioned on the previous restriction.

3.3 Generation of the Concept Keyboard

The system must allow the designer to generate in a semi-automatic way, the corresponding concept keyboard of an algorithm or data structure. The generated concept keyboard must allow the final user to traverse the data structure and participate in the execution of the algorithm. The ideal requirement would be to generate the concept keyboard directly from the source code that implements the algorithm or data structure. This approach has the problem that it implies that the developer must implement the algorithm focused on the generation of the concept keyboard, making the implementation less “cleaner”. It is necessary to describe the class that implements the algorithm in a more abstract way than the source code itself. This description must include a specification of the relevant methods of the class that implements the algorithm or data structure by specifying the name of the class, its parameters and its return value. Besides the formal description of the class, it is required that the designer can complement the description with a brief explanation of the functionalities every method has.

It was decided to use an interface file for describing the class and its methods. This interface file was designed as an XML file in order to have a structured way of presenting the information and, at the same time, in a human-readable format. In the following chapters, the file that describes the data structure will be named as the Data XML file.
CHAPTER 3. REQUIREMENTS

With the use of the Data XML file it is possible to execute the methods that the algorithm or data structure has implemented, by using the concept of reflection that is available in Java. The software can be able to interpret the Data XML file and execute the methods described, determine the parameters required, and the return value of each method.

Based on the Data XML file, it must be possible to generate the corresponding concept keyboard. The actions specified on the Data XML file must be mapped to a key on the keyboard, in order that every time the user “touches” a key, the corresponding action is executed. The layout of the keyboard must also be customizable, i.e. the distribution of the keys on the keyboard can be arranged according to the characteristics of the corresponding functionality and the way it relates to the other actions. Also, every key must have a sound related, in order that the interaction between the user and the algorithm or data structure is not only based on the visual channel, but also uses the auditive channel.

The generation of the concept keyboard with all the characteristics described before, must be assisted by the software. The software should inspect the corresponding class that implements the algorithm, and assist the designer in the process of generation and customization of the concept keyboard.

After generating the concept keyboard, it is described using another interface XML file, known as the Keyboard XML file. This interface file describes the main characteristics of the concept keyboard, the class that references and the amount of keys it posses, as well as the description of every key with its corresponding action, label and sounds.
Chapter 4

Concept Keyboard Software

In this chapter, it will be presented the software developed as part of the thesis project. The objective of this software is to help the teacher in the process of semi-automatic generation of the concept keyboard, as well as a algorithm visualization tool for the final user, ie. the student.

The software is divided in two sections: Configuration of the Concept Keyboard and Algorithm Visualization (figure 4.1). The objective of the Configuration of the Concept Keyboard section is to allow the user to generate the interface files, based on an existent AV implementation. These interface files, as described in the previous chapter, will contain a description of the data structure (named “Data XML file”), as well as a description of the generated concept keyboard (“Keyboard XML file”). The second section, Algorithm Visualization, is intended to give the user a graphical visualization of the algorithm, were he can traverse the data structure using the generated concept keyboard.
4.1 Configuration of the Concept Keyboard

During the configuration stage, the user acts as a designer who determines the actions that will be available to the final user, and customizes the concept keyboard that will be generated. The process of generation of the interface files is divided in seven steps:

1. Determine the class to be used
2. Choose the methods
3. Enter description
4. Save generated XML file (Data XML)
5. Arrange the keys of the keyboard
6. Customize each key (label and sound)

7. Save generated XML file (Keyboard XML)

The first four steps are used to create the “Data XML” file that describes the data structure, and the functionalities that will be available for generating the concept keyboard. Each of these functionalities will be mapped to a key in the concept keyboard, allowing the user to interact with the data structure. The next three steps are used in the process of creation of the “Keyboard XML” interface file used to characterize the corresponding concept keyboard.

To have a common graphical user interface, it was designed a basic structure containing the common information and available actions that will be available during the configuration of the concept keyboard (figure ??). At the left side of the window, it is presented the seven steps, and the current step that is being executed is highlighted. The right side of the window is used by every step of the configuration to display and interact with the user. At the bottom of the window, there are the navigation buttons. These buttons permit the user to go forward and backwards in the configuration process, and cancel the configuration at any time.

4.1.1 Generation of Data XML file

As mentioned before, one of the main objectives of this software is to take advantage of existent implementations with no, or few, modifications. In the process of creation of the concept keyboard it is necessary to construct the Data XML file.

To generate the Data XML file, the first step is to choose the class that
CHAPTER 4. CONCEPT KEYBOARD SOFTWARE

Figure 4.2: Main configuration window. At the left hand side, the user is presented with a list of the configuration steps (A). The right side is used to display the information of each step (B). The navigation buttons (C) allow the user to go backwards and forward in the configuration steps, as well as to stop the configuration at any time.
CHAPTER 4. CONCEPT KEYBOARD SOFTWARE

33

implements the visualization of the algorithm (figure 4.3).

Figure 4.3: Choose class. The designer must enter the name of the class (A) implementing the algorithm or data structure.

After choosing the class, the designer must determine which methods (actions) must be available for the user to execute (figure 4.4), in order to achieve a certain granularity in the manipulation of the data structure. Depending on the characteristics of the final users and the planification of the course, the actions required can be different. The system presents a list of the methods available in the selected implementation, and the designer must choose the required methods. In order to allow a certain level of abstraction between the implementation and the specification levels, the designer can give a brief description to each of the selected methods. This description
will be useful in the process of generation of the concept keyboard, since the
designer can relate a concept to each action, rather than just the name of
the method that implements the functionality.

![Image of diagram]

**Figure 4.4:** Choose method of the class. The designer can select from the list of
available methods (A) the relevant ones, by selecting the corresponding checkbox (B). It is possible to add a brief description (C) for every method selected.

Some algorithms can have different implementations, depending on the
interpretation the author gave to the problem that must be solved. To help
the student to understand better the problem to be solved, the designer can
enter a description of the current implementation of the algorithm (figure 4.5). This description will be displayed to the student when he works in the
Algorithm Visualization section of the software.

Figure 4.5: Enter description. The designer can add a description (A) of the special characteristics of the implementation of the algorithm.

With the information recollected in these three steps, the system generates a XML file that describes the data structures and the actions that can be used. The designer can read the generated XML file in order to check the correctness of it and return to a previous step if it is necessary to make a modification. This XML file, known as the “Data XML” file, must be saved in order to use it in the visualization section (figure 4.6).
Figure 4.6: Save generated Data XML file. The designer must enter a name for this new file in the area A, or browse for a special directory (B) in the computer. After saving the file (C), the designer can continue with the next steps.
4.1.2 Generation of Keyboard XML file

To generate the “Keyboard XML” interface file, the designer must determine the position of the selected actions in the keyboard. Each action is represented by a key, that will allow the final user to interact with the data structure and algorithm. The user drags and drops each action to a position in the keyboard (figure 4.7). The keyboard is represented by a 5x5 grid, which can be configured to any MxN grid, where each position in the grid is named after the column and the row where it belongs. The designer can determine the distribution of the keys based on the results of usability tests, that can help him understand which actions are closely related, so the keys must be closer together.

After determining the position of the keys in the grid that represents the final keyboard, the user can customize the properties of each key. The user can change the label that will be used to identify the key, as well as the sound associated to it (figure 4.8). Each key can have two different sounds associated to it: Over sound will be played when the user “passes over” the key, and Click sound will be played when the key is pressed. The system permits the designer to browse for a file in the computer and play the corresponding sound.

Furthermore, it is possible to specify if the algorithm or data structure has the possibility of using a startup file depending if the class implementing the algorithm has available this feature. This startup file can be used to specify the initial values of the data structure. For example, it can be used to specify the nodes in a graph, or the elements in a tree, so the student can experiment with different scenarios.

When the keyboard is configured, the system generates the second in-
**Figure 4.7:** Arrange the keys of the keyboard. The designer can drag and drop from the list of available actions (A) to the 5x5 grid (B). The designer can also move the actions inside the grid to a different position. To have a better understanding of the meaning of every action, the system displays a description (C) of every action by moving the mouse over them.
Figure 4.8: Customize the keys. Every key in the keyboard can be customized by modifying the label associated to it (A). To complement the visual channel, the designer can associate an “over” sound (B) that will be played when the user “passes over” the key, and a “click” sound (C) that will be played when the user “presses” a key. To find the file, it is possible to browse in the computer (D) and play the file (E) to test if the selected file corresponds to the desired sound. The combo box (F) allows the designer to determine if a startup file will be used.
interface file (figure 4.9) and must be saved, in order to use it during the visualization stage.

![Image of Configure Concept Keyboard interface]

**Figure 4.9:** Save generated Keyboard XML file. The designer must enter a name for this new file in the area A, or browse for a special directory (B) in the computer. After saving the file (C), the designer can continue with the next steps.

### 4.2 Algorithm Visualization

The second section of the system (*Algorithm Visualization* section) consists on the application for visualizing the algorithm complemented with the
use of a concept keyboard.

In order to visualize an algorithm, it is necessary to specify the interface files that will be used to describe the data structure and the concept keyboard. The selection of these files is essential for the visualization since they are in charge of fully describing the characteristics of the concept keyboard and the corresponding actions that must be executed. Besides the selection of these two files, it is necessary to specify the startup file that must be used, if the algorithm supports the use of a startup file.

To obtain these information, it was implemented two kind of graphical interfaces. The first graphical interface implemented is focused to the expert user that would like to choose freely the XML files to be used (see figure 4.10), as well as the startup file that describes the initial state. The second graphical user interface was designed to allow the teacher to create a group of specific concept keyboards and startup file. By defining a text file with the name of the XML files and the startup file that must be used, the student can select from a list of available configurations, the one that he/she wants to work (figure 4.11).

After determining the interface XML files and the startup file, the user is presented with the visualization of the algorithm or data structure (figure 4.12). The visualization is based on four windows (figure 4.13): concept keyboard, data input, draw area, and description.

**Concept Keyboard** shows the corresponding concept keyboard. Each key represents a functionality that allows the user to manipulate the algorithm or data structure. This keyboard is based on the “Keyboard XML” interface file generated in the configuration section.

**Data Input** allows the user to input data for the actions that may require
Figure 4.10: Choose XML files and startup file (expert user). The user must determine the name of the Data XML file (A) and the name of the Keyboard XML file (B). After choosing these two files, the user must enter the name of the startup file that will be used (C).

Figure 4.11: Choose XML files and startup file (normal user). The user can choose from one of the available set of files (A) the one he wants to use.
Figure 4.12: Algorithm Visualization interface.
Figure 4.13: Algorithm Visualization windows. (a) Draw area, (b) Description, (c) Concept Keyboard, (d) Data Input.
more interaction with the user (e.g., the value of a new element in a tree). This window is also used to give the user the feedback of the results of the different actions executed.

**Draw Area** displays the graphical output of the algorithm, or the graphical representation of the data structure.

**Description** shows a description of the current algorithm. This is used to help the student have a better understanding of the objective of the algorithm, and in some cases, describe some special considerations of the current implementation.

During the visualization of the algorithm, the user can load different algorithms, and their corresponding keyboards. Each algorithm can have multiple concept keyboards depending on the possible actions that the user can execute. Furthermore, the system allows the user to load different layouts of the same concept keyboard, for each of the algorithms. Figure 4.14 presents two layouts of the same concept keyboard. This permits the designer to create different concept keyboards depending on the characteristics of the group of students that will use the system, or based on the results of usability tests.

Depending on the purpose of the study, the teacher might want to create a concept keyboard with only a subset of all the available actions, and another keyboard with a different subset of actions. For example, if the purpose of the study is to understand how to build a binary tree, the teacher might add the actions “insert node”, “remove node”, and “get height”. On the other hand, if the purpose of the study is to understand how to navigate through the tree, the selected actions could be “move upper node”, “move left node”, “move right node”, and “move to root”.
Figure 4.14: Different layouts. Both of these concept keyboards have the same possible actions, but the layout of the keys is different.

It is also possible to load a startup file (if the algorithm allows it), to change the initial values of the graph, tree, etc. so the user can have different problems to solve. This feature allows the student to analyze different possible scenarios were an algorithm can be executed. For example, to understand how an algorithm works in the worst-case, best-case, average-case scenarios.
Chapter 5

Implementation

5.1 Data XML file

The algorithm or data structure is described using a XML file. This XML file allows the designer to make an abstraction of the real implementation and just describe the characteristics of the algorithm or data structure. The main structure of this file can be described by the DTD (Document Type Definition) presented on listing 5.1.

To understand how this file describes the class that implements the algorithm, it is necessary to understand the structure of the DTD of the Data XML file. In the following table, it is presented a description of every tag in the file.

**classname** Describes the name of the class implementing the algorithm or data structure. It is written in the format `package.class`.
Listing 5.1: DTD of the Data XML file.

**action** Represents each of the relevant methods of the class. The attribute
*id* is a unique identifier for every method and will be used when de-
scribing the concept keyboard.

**description** Used by the designer to describe better the functionalities of
the method.

**methodName** Indicates the name of the method as presented on the source
code.

**methodArgument** Encapsulates each of the arguments of the method.

**type** Indicates the Java variable type of the argument.

**name** A description of the argument, used to explain the user what repre-
sents every parameter in the execution of certain method (action).

**methodReturn** The variable type of the return value of the method.
An example of a Data XML file, describing the implementation of the AVL tree, is presented on listing 5.2. This example is only a fragment of the interface file, but is used to understand better the structure of the file.

```xml
<?xml version="1.0">
<dataStructure>
  <className>avltree.AvlTree</className>
  <action id="0">
    <description>Insert an element</description>
    <methodName>insert</methodName>
    <methodArgument>
      <type>int</type>
      <name>Element to insert</name>
    </methodArgument>
    <methodReturn>void</methodReturn>
  </action>
  <action id="1">
    ...
    ...
  </action>
</dataStructure>
```

Listing 5.2: Data XML interface file example

5.2 Keyboard XML file

After describing the algorithm or data structure, it is necessary to describe the concept keyboard that will be used. To model the characteristics and actions related to every key, it is used the Keyboard XML file. The
CHAPTER 5. IMPLEMENTATION

XML file can be described by the DTD presented on listing 5.3.

```xml
<?xml version="1.0" encoding="UTF-8">
<!ELEMENT keyboard (className, startup, dimension, keys)>
<!ELEMENT className (#PCDATA)>
<!ATTLIST startup type CDATA #REQUIRED>
<!ELEMENT dimension (rows, columns)>
<!ELEMENT rows (#PCDATA)>
<!ELEMENT columns (#PCDATA)>
<!ELEMENT keys (key+)>
<!ELEMENT key (methodId, location, label, sound+)>
<!ELEMENT methodId (#PCDATA)>
<!ELEMENT location (row, col)>
<!ELEMENT row (#PCDATA)>
<!ELEMENT col (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT sound (filename)>
<!ATTLIST sound type CDATA #REQUIRED>
<!ELEMENT filename (#PCDATA)>
```

Listing 5.3: DTD of the Keyboard XML file

The next table presents a description of the tags that form the Keyboard XML file.

- **classname** Describes the name of the class implementing the algorithm or data structure. It is written in the format `package.class`.
- **startup** Is used to determine if the class uses or not a startup file. The value of the `type` attribute indicates if the use of a startup file is
not implemented (type="0"), is optional (type="1"), or is required (type="2").

**dimension** Groups the elements related to the dimension of the generated concept keyboard (rows and columns tags).

**rows** The amount of rows of the grid that represents the keyboard.

**columns** The number of columns of the grid that represents the keyboard.

**keys** Groups a set of keys that belong to the keyboard.

**key** Represents each key on the concept keyboard. Every key has to have associated an action that will be executed when the user presses the key.

**methodId** A unique identifier that describes the method that must be executed when the key is pressed. It corresponds to the identifier assigned in the Data XML file.

**location** Groups the tags that identify the position of the key in the (rows x columns) grid.

**row** Indicates the row where the key must be placed.

**col** Indicates the column where the key must be placed.

**label** Indicates the label used on the key. This label has to describe the functionality of the corresponding action.

**sound** Used to specify the sounds associated to the key. If the attribute type is “over”, the file will be played when the user “passes over” the key. If the attribute has value “click”, then the file will be played when the key is pressed. This approach allows to add more sounds, by extending the definition of the type attribute.

**filename** Indicates the WAV file that will be played.
To understand better the way the interface file is used, on listing 5.4 it is presented a fragment of a real Keyboard XML file used to describe the concept keyboard generated for the AVL tree data structure.

### 5.3 Abstract class `GraphObject`

One of the main requirements in this work is that the system must be able of handling existent implementations of algorithm visualization. In order to use these implementations, it was identified two requirements related to the way of programming the algorithm or data structure: standarize the way of accessing the graphical representation, and the way of using the startup file.

To solve these problems, we propose that each algorithm must implement the abstract class `GraphObject` (listing 5.5). This class defines a common name for the method in charge of drawing the elements, as well as a common constructor used by the system. To handle the startup file, it is used the constructor that receives a `String` as the argument.

To understand better how to implement an algorithm, listing 5.6 presents the structure of the implementation of the Quicksort algorithm. The main difference with the common implementations is that the class must extend the `GraphObject` and implement the method `draw(Graphics)`, and the constructor with a `String`. 
CHAPTER 5. IMPLEMENTATION

Listing 5.4: Keyboard XML interface file example
public abstract class GraphObject {
    protected int width, height;

    // Implicit constructor
    public GraphObject() {}

    // Used in loading a startup file.
    public GraphObject(String filename) {
        this();
    }

    // Method in charge of drawing
    public abstract void draw(Graphics graphics);
}

Listing 5.5: Abstract class GraphObject. The constructor that receives a String is used to load the startup file. The method draw(Graphics graphics) is in charge of displaying the graphical representation of the algorithm.
package quicksort;

import java.awt.Graphics;
import common.*;
import java.io.*;

public class Quicksort extends GraphObject {
    protected Node root;
    protected Node current;
    protected String filename;

    // methods that must be implemented
    public Quicksort() { ... }
    public Quicksort(String filename) { ... }
    public void draw(Graphics graphics) { ... }

    // methods for performing actions
    public boolean moveLeft() { ... }
    public boolean moveRight() { ... }
    public boolean moveToBigger() { ... }
    public boolean moveToSmaller() { ... }
    public void selectPivot() { ... }
    public boolean splitElements() { ... }
    public boolean changeSibling() { ... }
    public boolean joinElements() { ... }
}

Listing 5.6: Example of Implementation: Quicksort
5.4 Main application

The software can be conceptually divided in two applications that work independent from each other: the Configure Application and the Concept Keyboard Application. As mentioned before, the first application allows the user to create and configure the interface XML files that will be used during the visualization of the algorithm. The second application is the one in charge of presenting to the final user the visualization of the algorithm, and enables the interaction with the concept keyboard.

5.4.1 Configure Application

The Configure Application was designed, based on the seven steps that are necessary to completely describe the data structure and the concept keyboard: determine the class to be used, choose the methods, enter description, save generated XML file (Data XML), arrange the keys of the keyboard, customize each key (label and sound), and save generated XML file (Keyboard XML). To execute each of these steps, it is necessary to complete the prior ones, since the result of each step is used as input for the next one. The designer must be able to navigate through the steps, in order to go forward and backward through them so he can modify the options selected on previous steps.

During the design of the Configure Application, two problems were observed: control data communication between each of the steps, and control the displaying of each of the windows that allow the interaction with the designer. To solve these problems, the system was designed using the Mediator design pattern (see appendix XXXX). This pattern uses a “director” in charge of controlling the communication between multiple “colleagues”.
CHAPTER 5. IMPLEMENTATION

Each of the windows (panels) that present the configuration steps, act as the “colleagues” and the class ConfigurePanel is the “director” class (see figure 5.1). The panels notify the director class to advance to the next or previous step, depending on the selection made by the user. Figure 5.2 presents an example of the interaction between the panels and the ConfigurePanel class. This allows to remove or add steps in the configuration process, only by modifying the ConfigurePanel class, since none of the steps controls explicitly the next or previous step. This centralized processing also makes simpler the data communication between each of the steps. The ConfigurePanel class creates an instance of the Data Manager class that is used as a repository of the common data structures, eg. the keyboard description, class description, etc. Every panel created by the ConfigurePanel receives a reference to the DataManager class, so each step can have access to the information of the previous steps without the need of passing particular data to every step.

**Figure 5.1:** Configure Application class diagram
Figure 5.2: Interaction between the panels (“colleagues”) and the ConfigurePanel class (“director”). While the Panel_1 is displayed, the user presses the button ‘Next’. Panel_1 sends the message to the ConfigurePanel class, and as a response, the director class hides Panel_1 and sets visible the Panel_2. Similarly, if the user presses the button ‘Back’, Panel_2 sends the message to the director, and it executes the corresponding actions.
5.4.2 Concept Keyboard Application

The Concept Keyboard Application presents the user with the graphical representation of the data structure and algorithm, and allows the interaction through the concept keyboard. This application presents four windows: the concept keyboard, the input area, draw area and the algorithm description. To control the interaction between these four windows and the data structure, this application was also designed using the Mediator design pattern. The “director” was implemented by the ActionManager class who is in charge of controlling the messages sent between the concept keyboard (Panel.Keyboard), the data input area (Panel.Input), the draw area (Panel.Graph), and the description area (Panel.Description). The ActionManager is also in charge of creating an instance of the data structure, or algorithm, and execute the actions selected by the user, as well as creating the corresponding concept keyboard. It reads the Data XML file and, through reflection, is able of loading the class and then execute the methods mapped to every key in the keyboard. In order to display the concept keyboard, it reads the Keyboard XML file, and arrange the keys according to the specification made by the designer, and associates to every key the corresponding sound and label. The keyboard is represented by the Keyboard class that consists on a set of keys.

This architecture allows the integration of new implementations, since every interaction is described in the XML files and the fact of extending the GraphObject class, permits to standarize the calls to the constructor and the method in charge of displaying the algorithm or data structure.

Another advantage of this architecture is that every component can be modified independently. For example, instead of using a “virtual” keyboard displayed on the screen, it is possible to map the keyboard on a “physical”
tablet, or use other input devices as a joystick, by only modifying the class 
the Panel.Keyboard. Also, new ways of visualizing the algorithms can be 
tested, by extending the class in charge of the display of the algorithm.
Chapter 6

Algorithms

In this chapter it will be presented a brief description of how algorithms are classified, and what are the main problems present in the study of new algorithms. Afterwards, it will be described the algorithms that were implemented, and some examples of the concept keyboards generated.

6.1 Algorithm Classification

Algorithms can be classified according to the type of problem to which they were designed, as well as the type of data structure that is used in the execution of the algorithm. The most common algorithms that are present on the fields of study related to science are those related to graphs, trees, sorting data, and searching in text.
6.1.1 Graph Algorithms

Graphs represent a set of nodes (vertices) which are connected by a group of arcs (edges). The graphs can be classified in two groups: undirected and directed graphs. The nodes in an undirected graph are connected by arcs that do not have associated a direction, i.e. the arcs only differ by the two nodes they connect and not by their starting and ending nodes. On the other hand, the nodes in a directed graphs are connected by arcs that can be differentiated by the nodes they connect and by their starting and ending node.

Usually, the arcs in a graph have a weight associated that helps to model different kinds of problems. For example, if every node in the graph represent a city, the weight of the arcs can represent the distance between the cities, the cost of building the road, or the cost of accessing it.

The most common problems that are presented in the introductory courses to the graph data structure and graph algorithms, are related to route problems. For example, finding the minimum spanning tree in an undirected graph, or the shortest path between two nodes in a directed graph. In this work we tested the algorithm visualization software with the implementations of Kruskal algorithm and Prim algorithm for determining the minimum spanning tree, and Dijkstra algorithm for determining the shortest path between two given nodes.

6.1.2 Tree structures

A tree consists on a set of nodes, where each node has a “parent” node (with the exception of the root node) and zero-or-more “child” nodes. There
exists many algorithms related to the tree structures. Most of these algorithms try to solve the problem of inserting, deleting, or searching nodes in the tree efficiently, hopefully in logarithmic time. From these algorithms, we chose to implement two of them: AVL trees and Binary Heap trees.

6.1.3 Sorting algorithms

Sorting a set of data, in ascending or descending order, is of great importance in every day problems as well as in theoretical studies. There exist different approaches for solving this problem. Some of these approaches use the concept of divide and conquer, in order to divide the main problem in “smaller” and “simpler” problems, that can be easily solved and then reconstruct the main problem. To implement this kind of solutions, recursive algorithms are used, which are very difficult to understand for students that are new in algorithms studies. This is one of the reasons why we decided to implement the Quicksort algorithm, and test the comprehension obtained by different students.

6.1.4 Search in text

Another kind of algorithms are those related to searching a particular pattern inside a text. Many studies have been done in this field, in order to find a particular text, in an efficient way. In general, these kind of algorithms are based on pre-processing the text were the pattern must be searched, in order to have an index of the occurrence of some letters or words. After the pre-processing of the text, the algorithm consists on finding a match and iterating over the text according to the rules specified in the algorithm. In these kind of algorithms, the most difficult part for the student to understand
is how to pre-process the text and why certain pre-processing improves the future searches.

These kind of algorithms is not very suitable to be visualized, as it is presented in this work, since the most important part of the algorithm is done “before” the algorithm starts iterating. This is the reason why, we did not implement an algorithm for searching in text.

### 6.2 Algorithms implemented

In the following we will present several well-known examples for animated algorithms. The examples are classified under themes like sorting, searching, graph algorithms, string processing, image processing and others. We have selected those algorithms which have been favorably evaluated by users. Then an interface file was generated based on the structure of the algorithm. In this XML-file the names of classes and methods play an important role. The number of keys derived depends on the granularity of the algorithm. To illustrate this fact we will give several examples.

#### 6.2.1 Graph algorithms

**Shortest Path (Dijkstra algorithm)**

To solve the problem of determining the shortest path, the most popular algorithm was the one presented by Edsger W. Dijkstra, known as “Dijkstra’s algorithm”. There exists many interpretations of the problem to be solved by Dijkstra algorithm, since some authors present it as “finding all the shortest
paths that connect any two nodes of the graph”, or “find the shortest path between given starting and ending nodes”. Our implementation considers the second approach, since it is simpler to understand, and the problem of finding all the paths is just an extension of the simpler problem.

Based on this implementation, we designed a concept keyboard that will allow the user to navigate through the arcs and nodes of the graph, and determine which node should be selected, according to the algorithm presented by Dijkstra. The actions that are available are the following:

selectNextEdge() moves the current position (selected edge) to the next available edge.

selectPreviousEdge() moves the current position to the previous available edge.

addNextNode() adds the selected node to the set of “reached nodes”, and update the total cost of the reached nodes.

removePreviousNode() allow the user to remove the last node that was appended.

showPath() shows the current path, based on the nodes selected by the user.

showShortestPath() shows the corresponding shortest path, using the Dijkstra algorithm.

testShortestPath() it tests if each of the steps taken by the user when he/she selected the nodes satisfy the conditions stated in the Dijkstra algorithm. This is useful to test if the user really used the algorithm, or just chose the nodes by inspection.
Figure 6.1 presents the concept keyboard proposed for this algorithm. The keys that have similar functionalities are grouped together. For example, the keys that allow the user to iterate through the available edges ("Next Edge" and "Previous Edge" keys) are grouped together, as well as the keys that let the user to select and deselect nodes ("Add Node" and "Remove Node") are also grouped together. The third group of keys consist on the keys that allow the user to test if the algorithm has finished, or obtain information related to what is the solution to this problem.

![Concept Keyboard](image)

**Figure 6.1:** Concept keyboard for Dijkstra Algorithm. Groups that are conceptually related are closer together.

Examples of different scenarios that were presented to the user can be observed on figure 6.2.
Figure 6.2: Visualization of Dijkstra Algorithm
Minimum Spanning tree

Another common problem is the one of finding the minimum spanning tree, i.e. finding a set of arcs, where the sum of their cost is minimum, which connects all the nodes, without creating cycles (loops between nodes). To solve this, we decided to implement two algorithms that can be used to find this tree, the Kruskal algorithm and the Prim algorithm. Both of them find the tree with minimum cost, but using different approaches.

The Kruskal algorithm starts with a forest of \( n \) trees, where each tree consists of each of the nodes of the graph. In each iteration the algorithm picks the edge that has the smallest cost and that does not create a cycle. Then, it connects the two small trees with the selected edge, to create a bigger one. At the end of the process, the algorithm ends with one tree where each of the nodes of the graph is connected, without forming any cycle.

The Prim algorithm begins with the edge with least cost, and marks the two nodes it connects as “settled”. Then, in each iteration, the algorithm adds the edge with least cost, where one of the ends is marked as “settled”, and the other end is not. This assures not to create cycles, while picking the edges with least cost. The process ends when all the nodes in the graph are marked as settled.

Although both algorithms are implemented differently, the actions associated to them are very similar. To both of them, the actions that were defined were the following:

\[ \text{selectNextEdge()} \] allows the user to navigate through the edges of the graph.
CHAPTER 6. ALGORITHMS

selectPreviousEdge() allows the user to navigate through the edges of the graph.

addCurrentEdge() adds the current edge, and as a consequence, marks the nodes, at the ends of the edge, as “settled”.

removeLastEdge() removes the last edge that was appended so its edges are unmarked as “settled”.

test() it tests if the algorithm (Kruskal or Prim) was used during the selection of the edges and nodes. This allows the user to test if he/she has used in a correct way the algorithm.

Using these two similar algorithms permitted to use the same concept keyboard and study how much the students understood each of the algorithms, and the differences between the ways of processing the graph. The concept keyboard proposed during the testing of the algorithms is shown on figure 6.3.

Figure 6.3: Prim and Kruskal Concept Keyboard
On figure 6.4 and 6.5 different screen captures of each algorithm during the processing of the graphs are shown.

### 6.2.2 Tree algorithms

#### AVL Trees

An **AVL-tree** is a balanced binary tree proposed by Adelson-Velskii and Landis: it is not perfectly balanced, but the pairs of sub-trees differ in height by at most one. Insertion, deletion and search take logarithmic time. Implementations of AVL-tree insertion may be found in many textbooks and are a little bit tricky. An extra attribute (the difference between the heights of its left and right sub-trees) for each node indicates whether the tree is left-heavy, balanced or right-heavy. If the balance is destroyed by an insertion, we check the nodes for violation ascending from leaves to the root and focus the first node with height different than -1,0,1.

Depending on the structure of the sub-trees, different standard rotations are performed to correct the balance: a simple rotation to the left-hand or right-hand side or double rotations equalize their height by cutting a sub-tree and moving it across to become the child of the upper node.

To do this work and to navigate on the tree we have configured the concept keyboard in figure 6.6. We used a standard JAVA-coded implementation of AVL-trees with the AVL-class and all relevant methods. This implementation defines a set of actions that can be executed over the AVL-tree. Although the implementation considers many methods, the ones that are considered useful for the learner are the following:
Figure 6.4: Visualization of Kruskal Algorithm
Figure 6.5: Visualization of Prim Algorithm
**CHAPTER 6. ALGORITHMS**

**insertAVL(x)** inserts an element in the tree, keeping the AVL condition of the subtrees.

**insert(x)** inserts an element in the tree, without the restriction of the AVL tree.

**moveUp()** moves focus to the parent node.

**moveLeft()** moves focus to the left child node.

**moveRight()** moves focus to the right child node.

**isAVL()** shows if current tree is an AVL tree.

**rotateRight()** rotates current node to the right.

**rotateLeft()** rotates current node to the left.

**Figure 6.6:** Concept Keyboard for AVL algorithm

The following figure shows a real situation after the insertion of several nodes and the execution of a few rotations.
Figure 6.7: Real situation in executing the AVL algorithm
CHAPTER 6. ALGORITHMS

Binary Heap Trees

A heap sort stores \( n \) entries in a binary tree. Then, beginning at the end and going on from right to left, the heap structure is established by comparing and exchanging nodes. Arriving at the root of the tree the first element of the sorted list is found and exchanged for the last element. The next steps are repeated until the complete list is sorted. The root element sinks to this right position, the heap structure is again reached and the new root exchanged. Several authors provide only two methods: “Build heap” and “Move root to sorted list and reheap”. For the purpose of this work, we configure the concept keyboard presented in figure 6.8.

![Concept Keyboard](image.png)

**Figure 6.8:** Binary Heap concept keyboard

Figure 6.9 presents examples of different binary heaps used during the process of learning the algorithm.
Figure 6.9: Real situation in executing the Binary Heap algorithm
6.2.3 Sorting algorithms

Quicksort

The Quicksort algorithm was invented by C.A.R. Hoare in the mid-sixties, and it is still one of the most efficient sorting algorithms. The algorithm can be described as follows, supposing that the elements are stored in an array:

1. First, one of the elements of the array is selected as the pivot. Some implementations select the pivot randomly, and other implementations choose the first element. Our implementation allows the user to determine which element to use as pivot.

2. Then, each of the elements is “moved” inside the array, in order to have the smaller elements at the left of the pivot, and the bigger elements at the right.

3. Finally, each of the sub-arrays that were created (elements smaller and bigger than the pivot) are sorted recursively.

4. This recursive process stops when each of the sub-arrays has zero or one element.

In our work, the implementation we made of the quicksort algorithm is based on displaying an array of elements, and the user can navigate through the elements by moving to the left and right. The user can select the pivot, and move the elements. The actions that the user can execute are the following:
**moveLeft()** moves current position of the cursor to the left. This allows the user to navigate through the elements.

**moveRight()** moves the actual position to the right.

**selectPivot()** selects the current element as the pivot. This permits the user to select any of the elements of the array as the pivot, so the user can study how the algorithm works, depending on the pivot that was chosen.

**moveToBigger()** moves the current element to the right of the pivot, in order to have the elements that are bigger than the pivot at the right.

**moveToSmaller()** moves the current element to the left of the pivot, with the elements that are smaller than the pivot.

**splitElements()** this is used when all the elements were moved so the condition of having the smaller elements at the left and the bigger elements at the right is accomplished. Two smaller arrays are created, showing how the bigger problem was divided in simpler ones.

**changeSibling()** allows the user to move from one sub-array to the other sub-array.

**joinElements()** is used in the process of making the process backwards, with the sub-arrays already sorted.

Based on these actions, we propose the concept keyboard presented in figure 6.10. This concept keyboard allows the user to move the elements of the array, and simulate the process of dividing the main problem into smaller ones.

Figure 6.11 shows an example of the visualization of the Quicksort algorithm.
Figure 6.10: Quicksort Concept Keyboard
Figure 6.11: Visualization of the Quicksort algorithm
Chapter 7

Tests and results

The objective of the testing was to determine the comprehension achieved by using the Concept Keyboard Application and using a step-by-step algorithm visualization.

The testing was divided in two phases. The first one was done in Santiago, Chile during December 2003 with students of the Universidad de Chile. The second phase was done in January 2004 at Drusburg, Germany with students of the Compother Graphics, Image Processing and Scientific Computing Group of the Universitat Duisburg-Essen, directed by Dr. Wolfram Luther. After examining the results obtained from the tests done in Santiago, it was possible to determine the need to make some modifications to the structure of the tests as well as some modifications to the Concept Keyboard Visualization software.

In both cases, the tests consisted on a set of questionnaires that had to be answered by each student, on-line work with the algorithm visualization...
software, and solving some exercises related to the algorithms and data structures studied. The design of the tests was done with the help of Dr. Henning Breuer.

A characteristic of the Concept Keyboard Application is that it allows the teacher to analyze different aspects of the visualization of algorithms. The teacher can use different layouts of the same concept keyboard, in order to test the usability of different configuration of the keys. It is also possible to test different concept keyboards for the same algorithm, or different graphical visualization of the algorithm or data structure. To be able to analyze the results it is necessary to determine some common characteristics that will be present in every test and focus the study in only one aspect of the concept keyboard.

In the following tests, it was decided to focus on studying the different comprehension achieved by modifying the way the user interacts and controls the algorithm or data structure. For each algorithm, every student used the same generated concept keyboard (actions and layout) as well as the same visualization of the algorithm.

7.1 Tests in Santiago, Chile

Since, nowadays, Internet is one of the main sources of information, it was decided to compare the comprehension obtained between using the Concept Keyboard Application and step-by-step algorithm visualization available in Internet. There exists a large amount of on line AV that are distributed freely for the students. From these AVs it was selected at least two for each of the algorithms that will be included in the testing phase.
Figures 7.1 and 7.2 presents a screenshot of each of the web visualization software used.

7.1.1 Testers

For carrying out the tests, it was necessary to determine the characteristics of the final users of the algorithm visualization software. The students of Civil Engineer at the Universidad de Chile can be separated according to their level of knowledge in algorithms.

The first group consists on the first years students (students still in Plan Común study program). These students only have a basic knowledge on algorithms, mainly on sorting algorithms like Quicksort or searching in simple binary trees. The software can be used to allow a better understanding of these algorithms, but it is of not much use as testing, since the amount of algorithms that can be tested is limited.

The second group is composed by the students who are taking the first courses of Computer Science (third year students). They possess more knowledge on algorithms since they are currently studying the course Algorithms and Data Structures (CC30A – Algoritmos y Estructuras de Datos) or have recently taken the course. In this course, the students learn different kinds of algorithms such as sorting, search in text, tree and graphs algorithms.

The third group are the last year students and postgraduate students on Computer Science. They have more experience working and implementing algorithms than the students of the previous groups. Most of them must have done courses on analysis of algorithms, and must have studied more
CHAPTER 7. TESTS AND RESULTS

Figure 7.1: Screenshots of the Web Visualization used for each algorithm. (a),(b): Kruskal; (c): Prim; (d): AVL tree; (e),(f): Dijkstra
Figure 7.2: Screenshots of the Web Visualization used for each algorithm. (g),(h): Heapsort; (i),(j): Quicksort
advanced algorithms.

The students of the second and third group are more suitable for performing the tests since they have a much wider area of knowledge than the first year students.

The testing was divided in two sessions. The first session was done with students that were currently assisting the Algorithms and Data Structures course (CC30A), dictated by Prof. Juán Alvarez. The class had approximately 25 students, and 11 students volunteered for performing the tests in a 90 minute class. The second session was done with students that were currently in fifth or sixth year, and students enrolled in the master or doctorate program. In this group, six people volunteered (figure 7.3).

![Figure 7.3: Testers. (a) A picture of the students of the first session (third year students), during the execution of the tests. (b) Picture of students during the second session of tests.](image-url)
7.1.2 Algorithms tested

The tests consisted on working with algorithms for determining the minimum spanning tree: Prim and Kruskal; determine shortest path between two nodes in a directed graph: Dijkstra algorithm; sorting: Quicksort; binary trees: Heapsort and AVL.

In the session done with the third year students, it was used only the Prim, Kruskal and Dijkstra algorithms, since the last part of the course was focused on these algorithms, and they needed to reinforce these algorithms.

7.1.3 Steps

The test was divided in the following six steps:

1. Students must answer the “General Questionnaire”, which is used to extract general information related to the user.

2. The group of students is divided in two groups:
   (a) Group A: students will work with the Concept Keyboard Application.
   (b) Group B: students will work with the algorithm visualization available in Internet.

3. Students must answer a specific questionnaire for each group.

4. Solve a set of exercises.

5. The groups are swapped. Group A will work with the Web visualization, and group B will work with the Concept Keyboard Application.
6. Answer a comparison questionnaire

7.1.4 Questionnaires

During the tests, the students must answer a set of questionnaires that will help to understand the comprehension achieved as well as to obtain the suggestions that the students could make in relation to the software. Each student had to answer three questionnaires during the execution of the tests (see appendix ?? for the real questionnaires used). The results obtained from these questionnaires were subjective, but useful to determine possible improvements to future tests, and allows to study the personal appreciation of the students.

The objective of the first questionnaire, known as the General Questionnaire, is to obtain general information of the user (sex, study program, etc), and to determine the comprehension level of each of the algorithms in study based on the personal appreciation of the student.

For each type of algorithm visualization used (Concept Keyboard Application or Web Visualization), there exists a particular questionnaire that the student must answer. The objective of these questionnaires is to obtain the observations the students have in relation to each algorithm visualization they used.

The questions presented in the Concept Keyboard Application questionnaire were the following:

- What do you like of the \textit{Visualization with keyboard software}?
- What do you \textit{not} like of the software?
• Do you think it is useful for a better understanding of the algorithm?

For the Web Visualization, the questions presented were the following:

• Do you think it is necessary to use algorithm visualization software as a complement to what was learned in class? (yes/no answer)

• What improvements would you do to the algorithm visualization software you have used?

The third type of questionnaire is the Comparison Questionnaire and its objective is to determine which algorithm visualization they prefer (ie. they like more), independent of the objective results they obtain by solving the real exercises. In this last questionnaires, the student received an explanation of the main difference between both visualizations, and based on that, they must comment which one they preferred and why. This questionnaire consisted on the following questions, common to each of the groups.

• Of the two systems used (Web visualization and Visualization with Keyboard), which one do you like more?

• What advantages do you find in the system you liked more (the one you chose in the previous question)?

• The main difference between the two systems, is that the Web Visualization software is focused on showing the algorithms as an animation. On the other hand, the Visualization with Keyboard is focused on a direct manipulation of the algorithm. What advantages or disadvantages do you find on the Visualization with Keyboard?
7.1.5 Exercises

After working with one type of algorithm visualization software, the users must solve a set of exercises. Since the students of the first session (CC30A course students) only worked with Prim, Kruskal and Dijkstra algorithms, they solved only the exercises related to these algorithms.

For each algorithm, the students had to solve two exercises. There was an exception with the Kruskal and Prim algorithms were the students had to solve three exercises, since these algorithms are much easier to solve than the other ones. The following figures show the exercises presented for each algorithm.

7.1.6 Test Results

From the questionnaires that every student had to answer, it was possible to obtain subjective results. Appendix ?? shows in detail all the answers to the questions present on the questionnaire. The most relevant answers to each of the questions are the following:

- What do you like of the Visualization with keyboard software?
  - “Distribution of the elements, nice colors and friendly.”
  - “It is possible to interact and work in the development of the algorithm.”
  - “It’s simple to use, and it is a good tool for the study of the algorithms, because of the way it teaches.”
  - “Nice GUI. It is easy to understand the algorithms, and high level of interaction with the user.”
Figure 7.4: Kruskal algorithm exercises
Figure 7.5: Prim algorithm exercises

Figure 7.6: Dijkstra algorithm exercises
CHAPTER 7. TESTS AND RESULTS

Figure 7.7: AVL algorithm exercises

Figure 7.8: Heapsort algorithm exercises

Figure 7.9: Quicksort algorithm exercises
• What do you not like of the software?
  – “The arcs could be chosen by clicking them.” (3 users)
  – “Fonts of the letters, missing help.”
  – “The lack of a guide for describing the objectives of each algorithm.”

• Of the two systems used (Web visualization and Visualization with Keyboard), which one do you like more?
  – All but one preferred to work with the Visualization with Keyboard.

• What advantages do you find in the system you liked more (the one you chose in the previous question)?
  – “(Visualization with Keyboard) More friendly, understandable, interactive. Allows that we solve the problem, and not just visualize the algorithm.”
  – “(Visualization with Keyboard) Allows the user to make the path by yourself. The other [AV] just gives the correct answer.”
  – “(Web Visualization) Both of them have advantages depending on if it is used to practice or for learning. In the case of learning a new algorithm (which is my case) I prefer the [web] visualization.”
  – “(Visualization with Keyboard) It exists the possibility that the user can choose the actions to take, it is not just a step-by-step solving.”

• What advantages or disadvantages do you find on the Visualization with Keyboard?
Advantage: The student can view the results of the modifications made so he/she can test if he/she understood the algorithm. It allows to study specific cases in which he has doubts, instead of only using the pre-made examples available on the web.

Advantage: It has more interaction [with the user], and allows to learn from the mistakes made. Disadvantages: Navigation with keys is less natural than with the mouse.

Advantage: It allows the student to practice more. As a suggestion, it would be nice to add some hints of the algorithm.

The exercises solved by the testers allowed to obtain more objective results. The students that used the Visualization with Keyboard type of algorithm visualization before solving the exercises, obtained a 95% of correct answers. On the other hand, the users that worked with the Web Visualization before solving the exercises, obtained a 90% of correct answers.

7.1.7 Analysis

The students that worked on the tests liked the level of interaction obtained with the Concept Keyboard Software. When they had doubts, it allowed them to explore for the correct answer, giving a deeper understanding of the algorithms.

Some users disliked that the concept keyboard and algorithm visualization are on the same screen. The fact of having them on the same screen does not allow to take advantage of the concept keyboard. Using a physical concept keyboard, and not a representation on the screen, makes it possible to separate these two concepts.
There was not a considerable difference on the results obtained from the solving of the exercises. In both cases, most of the students answered correctly.

By analyzing the answers, it is possible to extract some improvements for future tests:

1. It is necessary to give the user a description of the objectives of the algorithm, and explanation of the actions that are available.
2. The concept keyboard must be presented as a separate device. The graphical representation does not allow to take full advantage of benefits of using a concept keyboard.

The difference between the graphical user interface used on the Concept Keyboard Software and the Web Visualizations may cause differences on the results obtained. Since it was used visualizations obtained from Internet, each of them had different colors and buttons, adding an extra variable in the analysis of the results.

7.2 Improvements to the Software

After executing the tests in Santiago, it was possible to identify improvements that could be done to the Concept Keyboard Software, before making the tests in Germany.

The first modification was to allow the designer to include a description of the algorithm during the configuration stage. This description can include
the characteristics of the algorithm or data structure, the main objectives, and/or a description of the possible actions that the user can execute.

In order to have more accurate results, it is necessary to use similar GUI between the visualization with a concept keyboard and the step-by-step visualization.

7.3 Tests in Druisburg, Germany

On the tests done in Germany, the objectives were similar to the tests done in Santiago: compare the comprehension achieved by using an algorithm visualization with Concept Keyboard, and a step-by-step interface. The difference with the other group of tests, was that it was decided to use a similar user interface in both kind of algorithm visualization (figure 7.10).

The algorithm visualization used on the Concept Keyboard Software was adapted in order to permit the usage of a step-by-step interface. The algorithms implemented were modified so besides the actions used with the concept keyboard, it was also available the actions present on a step-by-step interface.

7.3.1 Testers

In the tests executed in Germany, 18 people volunteered to make the tests. The volunteers were enrolled on different study plans, and in contrast to the tests executed in Santiago, it also participated a group of teachers. The exact distribution was the following: 1 students was in an undergraduate
7.3.2 Algorithms tested

The tests used the Kruskal algorithm for determining the minimum spanning tree; Quicksort for sorting a set of data; AVL trees for studying a binary tree; and Dijkstra algorithm to determine the shortest path between two nodes.

7.3.3 Steps

The tests had the same structure used on the tests done in Santiago. The difference between them was that in the tests executed in Gernant, the students worked with Quicksort and Kruskal using the step-by-step interface, and used the AVL and Dijkstra algorithm with the Concept Keyboard application.
CHAPTER 7. TESTS AND RESULTS

7.3.4 Questionnaires

The questionnaires used on the tests in Santiago were modified, by adding more questions (see appendix ??). These new questions had a new format. Instead of being open questions as most of the questions in the first version of the questionnaires, these new questions asked the user to qualify a set of statements. The users had to give a grade indicating the level of agreement with each of the statements. The questions included were the following:

- Answer the following questions by a grade of agreement (−− agree not at all; ++ agree completely).

<table>
<thead>
<tr>
<th>Question</th>
<th>−−</th>
<th>−</th>
<th>0</th>
<th>+</th>
<th>++</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is necessary to use algorithm visualization software as a complement to what was learned in class or tutorial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>My knowledge concerning the proposed algorithms is now better</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The tool allows to have a better insight into the proposed algorithms and data structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I am now able to explain the data structures in a better way</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>After having used the tool it seems to be simpler to program the algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The handling of the keyboard is intuitive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The tool is easy of use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Are the following features of the visualization with keyboard important or not (−− not important; ++ very important)?
### 7.3.5 Test Results

The results obtained from the latest tests gave new lights on the way the concept keyboard influences the comprehension achieved by the students.

With the results obtained from the answers of the exercises included in the tests, it is possible to observe that the students that had lower results, preferred the step-by-step interface over the concept keyboard. On the other hand, using the visualization with the concept keyboard, lead to better results in problem solving than the use of step-by-step interface.

People like the existence of an extended explanation of the algorithm being studied. They like to have a description of the characteristics of the algorithm and how they work. This description should be visible during the execution of the algorithm.

In relation to the graphical user interface, students prefer to have larger

<table>
<thead>
<tr>
<th>Feature</th>
<th></th>
<th></th>
<th>0</th>
<th>+</th>
<th>++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide further algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Include introductory or help text</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choose the keys from a list of methods in the Java program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arrange the keys of the keyboard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customize the labels of the keys</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provide underlying sound when clicking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use a graphic tablet as input interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use a real alternative hardware keyboard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provide other forms of output like speech or text</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
buttons to control the actions.

7.3.6 Analysis

The step-by-step interface does not require that the user must understand the algorithm before using the algorithm visualization software. This approach leads the user to the correct answer, without the need of understanding how the algorithm works. This is the reason why the students that had lower results on the problem solving section preferred the step-by-step interface. The algorithm visualization with the concept keyboard requires that the user must have a theoretical background of the algorithm. With this basis, the student can take advantage of the functionalities available. This last statement agrees with the results showing that the use of the visualization with a concept keyboard allow the user to improve their comprehension level.

The possibility of displaying a description of the characteristics of the algorithm or data structure is of great importance during the study of the algorithms. Students need to have a reference of the objectives of the algorithm during the visualization of the algorithms.
Chapter 8

Discussion

8.1 Summary

The aim of this thesis was to determine if the student can obtain a better comprehension of the algorithm or data structure by enhancing the way the user interacts with the visualization. Conventional algorithm visualization systems allow the user to control the execution of the algorithm with a limited interface, enabling the user to iterate in a step-by-step sequence. We present an enhanced interaction by allowing the student to navigate through the data structure and execute the possible functionalities available in the algorithm. Usually, an algorithm can be associated to a problem that must be solved, so the student must have the possibility to explore the way the algorithm works by selecting the appropriate action that must be executed in order to solve the problem that is presented. To show the possible actions and give the student the control of the execution, it is used a configurable concept keyboard.
CHAPTER 8. DISCUSSION

The instructor has the ability to use an existent visualization, with few modifications, enhancing the way actual visualizations are used. In the present work, we present the requirements that an existent visualization must satisfy, in order to generate the corresponding concept keyboard and enable the manipulation using this special purpose keyboard. The process of creation and further configuration of the concept keyboard is controlled by the *Configure Application* created during the development of this thesis.

After configuring and generating the corresponding concept keyboards, the student can manipulate and navigate through the data structures with the use of the *Concept Keyboard Application*. For every algorithm, the system allows the instructor to generate different layouts of concept keyboards, as well as using different sets of initial data. These features make it possible for the instructor to present the student with different scenarios and analyze the characteristics of the algorithms and data structures with different conditions.

After the development of these applications, it was necessary to design and execute different sets of tests for evaluating the pedagogical benefit of using enhanced algorithm visualizations. The tests were executed in two stages. The first stage was executed in Santiago, Chile, with students of the Universidad de Chile, and the second stage was done in Duisburg, Germany, with students of Universitat Duisburg-Essen. The tests consisted on a set of questionnaires that had to be answered by the students. The objective of these questionnaires was to obtain the observations and subjective appreciations the users had of the software. To obtain objective results, it was also designed a set of tests that were used to compare the level of comprehension achieved by presenting the users with different exercises. The results obtained were compared between the students that used the enhanced visualization and the students that used conventional algorithm visualization.
systems. After the tests done in Santiago, a new interface was developed, in order to have similar graphical user interface between the step-by-step interface and the concept keyboard visualization, so the results obtained are more representative.

The results obtained are very promising. The group of students that used the enhanced algorithm visualization system obtained better, and in some cases, the same results as the students that used conventional step-by-step interfaces. The observations obtained from the users are also positive. Most of the users liked the use of an interface that enables him/her to interact freely with the algorithm and manipulate directly the data structure. The users liked the fact of being able of solving the problem “by themselves”, instead of just observing how the system gets the right answer.

On the other hand, a recurrent observation made by the users is the need of separating the presentation of the concept keyboard, from the visualization. Presenting both of them on the same window gives the user the idea of being able of interacting directly with the mouse over the visualization, and not only over the keyboard. This problem makes it more difficult to take advantage of the concept keyboard.

8.2 Future Work

The present work can be used as a basis for future research. One of the most clear extensions are related to separating the concept keyboard from the visualization. As explained before, having the concept keyboard separated to a physical device connected to the computer can help the student to separate the idea of visualizing the algorithm and manipulating it. This separation can help the student to understand that the visualization is an
abstract way of representing the problem but it is not necessary the real problem. Also, using a “real” concept keyboard, allows the instructor to complement the process of learning new algorithms by using specially designed overlays that can be placed over the keyboard. The use of overlays can help the study of algorithms to physical impaired people, since the keys can be complemented with special textures (eg. use of Braille) and pictures.

The software that was developed as part of this thesis can be used to make different types of tests. For example, the instructor can use a particular visualization of an algorithm, and test the comprehension achieved by the students by using different layouts of a concept keyboard. It is also possible to test the same concept keyboard with different visualizations of the same algorithm, and determine which visualization is better for a particular algorithm.

In other words, the system gives the instructor the possibility to make different tests by changing the layout of the concept keyboard, the keys included in a concept keyboard, the visualization used, or the set of initial data that describes the scenario being tested.

### 8.3 Conclusion

The work presented on this thesis helped to determine that an enhanced interface for interacting and visualizing an algorithm or data structure, allows the students to understand better the characteristics of the algorithm. When the user is able of manipulating the data structure and explore the algorithm by experimenting, it gives him/her a higher level of understanding of the algorithm or data structure.