A performance comparison of search trees

Technical Analysis

Przemysław Rosiński
Piotr Janaszek

Faculty of Mathematics and Information Science
Warsaw University of Technology

21 November 2011

DOCUMENT METRIC

Project:	A performance comparison of search trees			Company:	WUT
Name:	Technical Analysis				
Topics:	Technical analysis, system architecture, used solutions				
Author:	Przemysław Rosiński, Piotr Janaszek				
File:	SearchTreesPerformance_TA.pdf				
Version no:	1.1	Status:	Working	Opening date:	2.11.2011
Summary:	Brief description of technical base of the application				
Authorized by:				Last modification date:	21.11.2011

HISTORY OF CHANGES

Version	Date	Author	Description
0.0.1	2.11.2011	Przemysław Rosiński	Created document
0.2	2.11.2011	Przemysław Rosiński, Piotr Janaszek	Created Preface For Management and Application Overview
0.4	4.11.2011	Przemysław Rosiński, Piotr Janaszek	Created Used Solutions
0.5	4.11.2011	Piotr Janaszek	Added Development model to Used Solutions
0.9	6.11.2011	Przemysław Rosiński, Piotr Janaszek	Created System Architecture and User Interface
1.0	6.11.2011	Przemysław Rosiński	Created Postface and fixed mistakes
1.1	21.11.2011	Przemysław Rosiński, Piotr Janaszek	Changed System Architecture

TABLE OF CONTENTS

PREFACE FOR MANAGEMENT

This document is presenting the technical aspects of the application. It is said to present analysis of used solutions and provided functionalities.

The aim of the project is to develop a generic C# library of some search trees algorithms. The library should be standardized in order to be utilizable in other applications.

The developers are also said to provide a sample application showing capabilities of this library. In order to do so the program should be able to retrieve necessary data about the tree manually by user input via GUI (Graphical User Interface) and/or automatically from external file. Data in external file can be from different distributions generated in R package.

The application is supposed to be user-friendly. Hence the user interface should be clear and intuitive in use. Also system architecture should be designed to give maximum reliability for the user needs and future update purposes.

In the Application Overview section there are main functionalities of the application described and simple activity graph shown.

The following sections are going deeper and are showing to the reader the application from the inside, mainly used technologies, algorithms and solutions.

APPLICATION OVERVIEW

In this section a reader can find short and brief description of the application from a distance without going deep into the technical details of the program. It will help the end-user to easily understand next sections of this document and get to know how the whole application is working.

Main functionalities

The application functionality consists of several simple tasks:

- Import data.
- Create data structure.
- Search for given element and show graphically the searching result.
- Prepare performance results and export it into external file.

The functionalities listed above are the most important parts in the application. All other features will come directly from them or will be added in order to develop the useful and reliable program.

Application activity

Almost all main tasks given to the program must be done sequentially, because of a step-by-step manner of activity. One action can be taken only if the previous one was finished. There are activities done in the loops which for user can seem to be parallel, in fact there are not. In one loop after retrieving some data application adds it to the tree until all the data is used. The second loop is similar – after making a step in traversal of the tree we add this information into performance results set.

The easy and convenient way of showing the scheme of activities is the activity diagram.

The figure on the next page simply shows the activity of the application in time. It is clearly visible that retrieving data is done in a loop until no more data will be considered during the execution. However, every time some data is gathered it is added to the structure. As aforementioned, searching for a given element is done in parallel with preparing performance results.

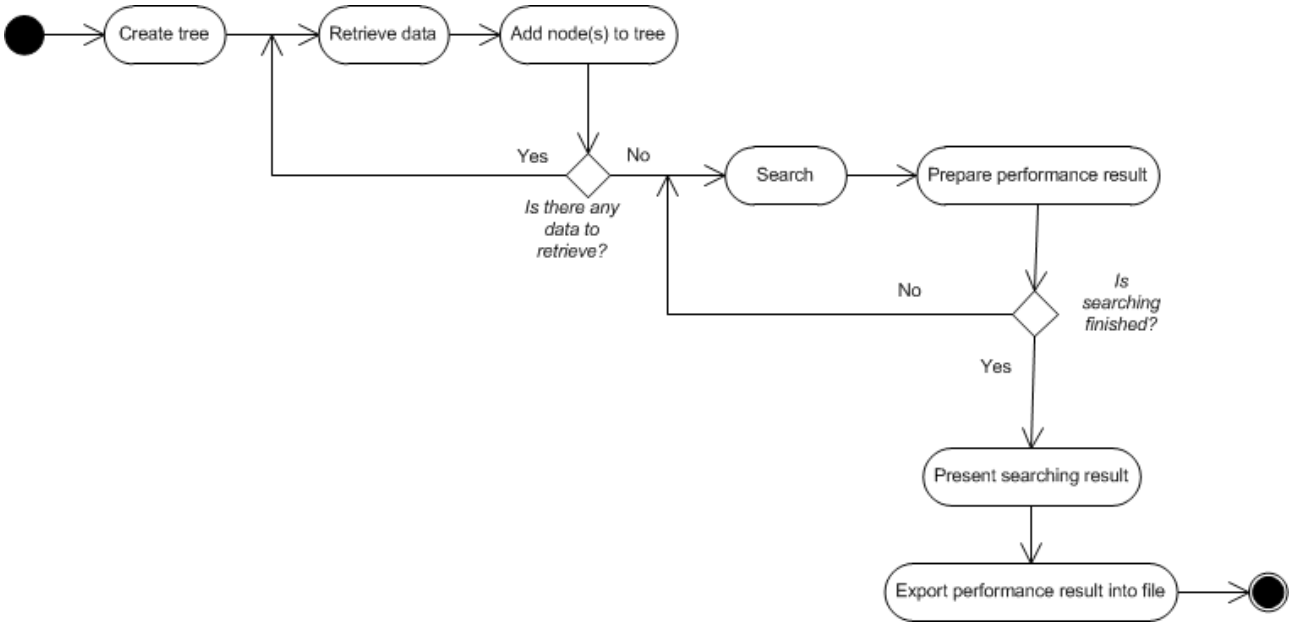

Figure 1: Activity diagram of the application.

USED SOLUTIONS

Library

The library of trees is said to be universal and easy to use in other applications. One would like to have standardized and popular format, while the other needs library effortless in linking and simple when importing data. The obvious choice is DLL (Dynamic-Link Library).[1]

External data

Both, input and output files must be compliant with R package. It is agreed that the application must be able to work with CSV (Comma-Separated Values) file format.

The definition of the CSV format is given in RFC 4180 by Internet Society organization.[2]

The most important assumptions of the standard are:

- Each record is located on a separate line, delimited by a line break (CRLF).

  For example:

  ```
  aaa,bbb,ccc CRLF
  zzz,yyy,xxx CRLF
  ```

- The last record in the file may or may not have an ending line break.

  For example:

  ```
  aaa,bbb,ccc CRLF
  zzz,yyy,xxx
  ```

- There may be an optional header line appearing as the first line of the file with the same format as normal record lines.

  For example:

  ```
  field_name,field_name,field_name CRLF
  aaa,bbb,ccc CRLF
  zzz,yyy,xxx CRLF
  ```

- Each line should contain the same number of fields throughout the file.

- Spaces are considered part of a field and should not be ignored.

- The last field in the record must not be followed by a comma.

Input file

Since one of the assumptions is “Each line should contain the same number of fields throughout the file”, number of trees described in one CSV input file must be limited to one. The justification is that the user may like to inspect trees of different number of nodes, which in turn enforces such approach. All the values from all the lines in the input file will generate one tree.

[1] Source: http://msdn.microsoft.com/en-us/library/ms682589.aspx
[2] Source: http://www.ietf.org/rfc/rfc4180.txt

Output file

To the output file performance results set will be exported. Number of fields and attributes of performance results will be the same for all investigated trees (type of the tree, data type, time of execution, number of nodes visited, etc.). Moreover, this information will be utilized in R package for comparing the results. Hence, in one file every line will represent one tree and every entry in the line will be a value corresponding to the titles listed in the header respectively.

Search algorithms

The most important part of the application is searching the tree itself. Required types of trees are: BST, AVL, red-black, 2-3, 2-3-4, B, B+ and Splay. All those contain sorted data what makes the computation of searching algorithms faster.

In searching algorithms there is pre-order traversal strategy implemented. It means that when searching for an element, first the parent node is compared with searched value and the child nodes are checked after it.

Searching algorithms' complexity depends on number of data entries in single node and the height of the tree. In self-balancing trees the height of the tree is always optimized in order to make searching most efficient.[3]

Below, there is presented table with searching time complexity for given tree types:

Tree type:	Average	Worst case
BST [4]	O(log n)	O(n)
AVL [5]	O(log n)	O(log n)
Red-black [6]	O(log n)	O(log n)
2-3 [7]	O(log n)	O(log n)
2-3-4 [8]	O(log n)	O(log n)
B [9]	O(log n)	O(log n)
B+ [10]	O(log n)	O(log n)
Splay [11]	O(log n)	O(n)

Table 1: Time complexity in big O notation.

[3] Source: http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree
[4] Source: http://en.wikipedia.org/wiki/Binary_search_tree
[5] Source: http://en.wikipedia.org/wiki/AVL_tree
[6] Source: http://en.wikipedia.org/wiki/Red_black_trees
[7] Source: http://en.wikipedia.org/wiki/2-3_tree
[8] Source: http://en.wikipedia.org/wiki/2-3-4_tree
[9] Source: http://en.wikipedia.org/wiki/B_tree
[10] Source: http://en.wikipedia.org/wiki/B%2B_tree
[11] Source: http://en.wikipedia.org/wiki/Splay_tree

Environment

The application together with the library are based on Microsoft solutions. Technology upon which it will be developed is Windows Forms and C# programming language. Chosen Integrated Development Environment is Microsoft Visual Studio 2010 since it provides all necessary features and functionality for creating modern and reliable applications and unified solutions.

Development model

In evolutionary development model there exists a division of the development cycle into smaller, incremental waterfall models in which a user is able to see the working product at the end of each cycle. The benefit of creating small pieces of code that works correctly is elimination of errors at very early stages. It puts emphasis on testing what profits in small number of high cost and time consuming failures at final stages of program development.

High quality code in conjunction with the evolutionary model will hopefully lead to reliable and functional application.

SYSTEM ARCHITECTURE

System architecture consists of three parts: trees library and presentation and logic layer of the application. Library is the universal package of trees. Presentation layer of the program is responsible for communication between the application and the user. Logic layer is processing the trees (building, searching) and creating the output.

Library

The library was said to be disjoint part of the application. In this package there are defined tree types. They all are inherited from the abstract class Tree. Also, all the methods and attributes are inherited. Since data entries can have their own attributes and flags, class Node is also in the library.

Application

Presentation layer

This layer as mentioned above is responsible for communication between the application and the user. In practice, it means that the program should handle the information transferred from user to the application and vice versa. Due to that the user interface including:

- tabs for switching between modes
- buttons for launching some methods
- drop-down lists for selecting items and options
- image box for plotting
- labels for presenting information

must lie in the presentation layer. But all the functionality triggered by using it should lie in another layer, mainly logic layer.

These assumptions bring the reliability to the interface. But also the program should have an intuitive interface for better communication. The reader can read more about that part in User Interface section on pages 12 and 13.

Logic layer

Logic layer is responsible for covering all the functionality that is not visible to the user. It is very important for the project, because the presentation of the results even in the most pretty way makes no sense when the results are incorrect.

For example, the input file must be loaded correctly with no errors. It must be loaded from given directory, in proper format, as one whole piece and without making any changes in the content.

Logic layer must be divided from the presentation layer for better reliability.

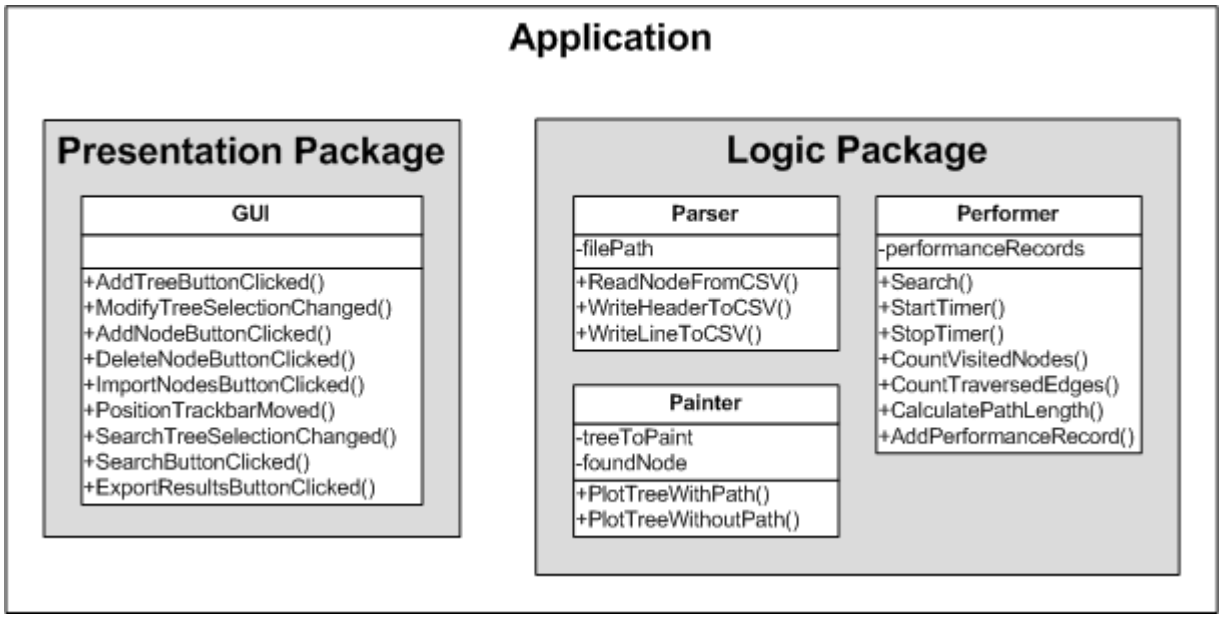

Figure 2: Class diagram of the application.

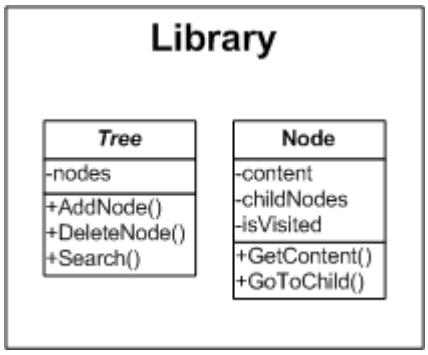

Figure 3: Class diagram of the library.

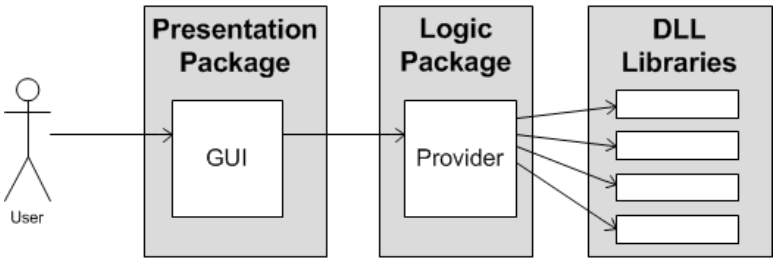

Figure 4: Data flow diagram.

USER INTERFACE

User interface consists of two main components, mainly Create and Search tabs. Both have in common that on the right side of the window there is a preview of the tree. The tabs differ in the left panel which is designated for user input.

The former serves for creation and modification of trees. *Add tree* button is used for building new tree and *Select tree* drop-down list is for selecting data structure under investigation. *Add node* and *Delete node* buttons are used for manual modification of the tree, while *Import nodes* button is used for automatic addition of data from external file.

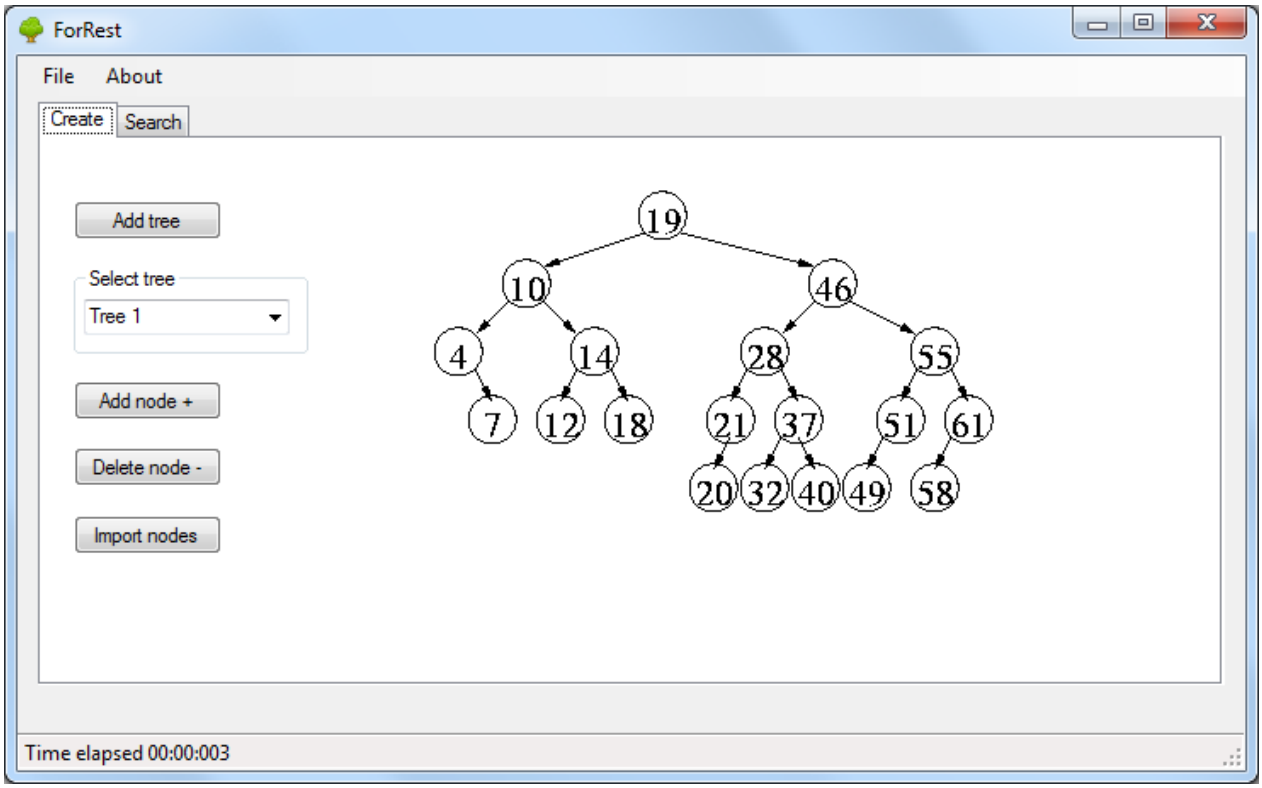

Figure 5: Application layout sketch.

The second tab is used for performing search operations. In *Select tree* drop-down list the user selects data structure on which searching algorithm will be executed. In *Search for* textbox there is the place for entering data the user wants to find. *Search* button triggers execution of searching process. Pressing *Export results* button cause exporting performance results set into external file.

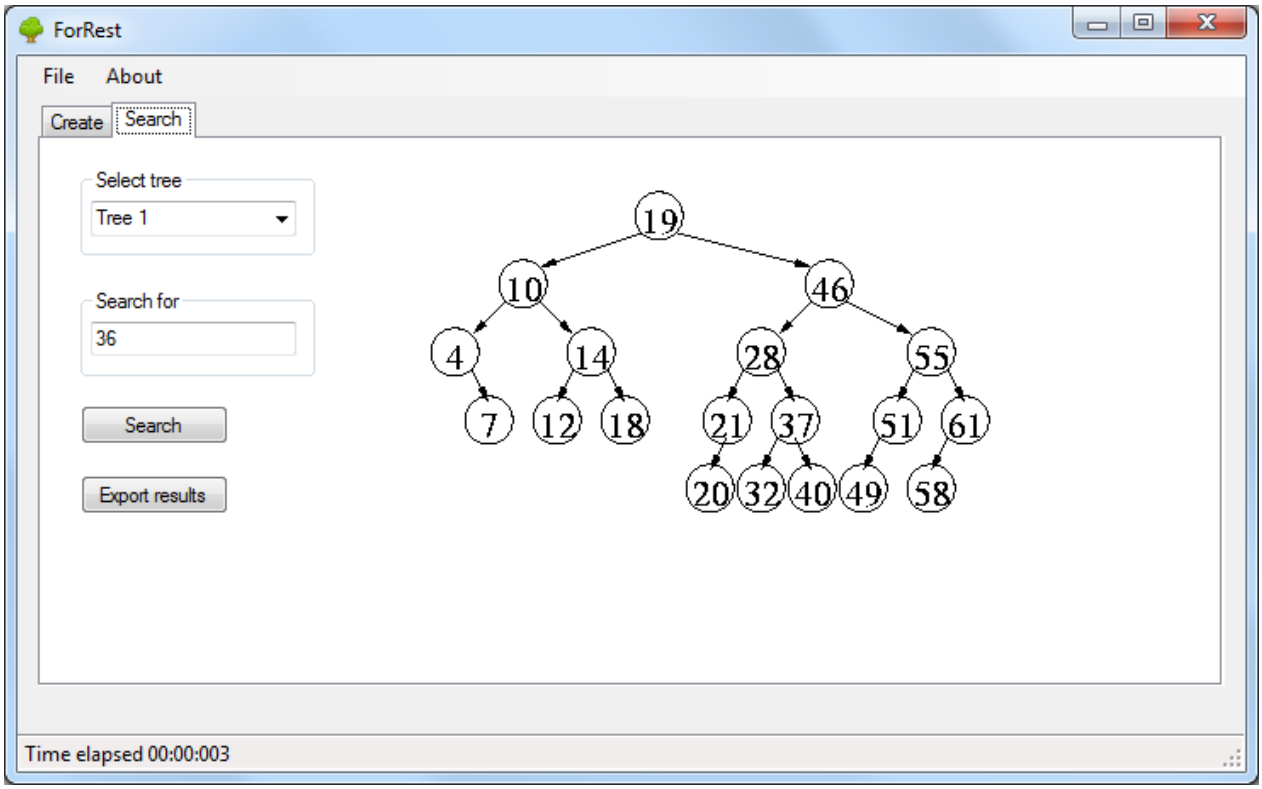

Figure 6: Application layout sketch.

POSTFACE

The application functionality is rather simple and it is designed to execute searching algorithms in different data structures. Program job should allow the user to perform the comparison between received results by storing sufficient information in standardized file.

The application also shows the user the way of storing data in the search trees.

Solutions mentioned in this document were chosen deliberately in order to make the library universally utilizable and to show the possibilities lying inside tree data structures.