

Final Project - Inverted Index and Comparative Analysis

Arthur Rabello Oliveira¹, Gabrielle Mascarello, Eliane Moreira, Nicolas Spaniol, Gabriel Carneiro

Abstract

We present the implementation and comparative analysis of three fundamental data structures for indexing and searching textual documents: the Binary Search Tree (BST), the AVL Tree, and the Red-Black Tree (RBT). Each structure was implemented with its core operations, including insertion and search. Unit tests were developed to validate the correctness and performance of these implementations. We also provide a further comprehensive comparative study of the three trees based on their time complexity, balancing efficiency, and suitability for document indexing. The results demonstrate the trade-offs between implementation complexity and query performance, offering insights into the practical considerations for choosing appropriate search tree structures in information retrieval systems.

Contents

1. Introduction	3
1.1. Motivation	3
1.2. Problem Statement	3
1.3. Overview of Search Trees	3
2. Data Structures Overview	3
2.1. Binary Search Tree (BST)	3
2.2. AVL Tree	3
2.3. Red-Black Tree (RBT)	3
2.4. Inverted Index	3
2.5. Comparison of Properties	3
3. Implementations	3
3.1. Binary Search Tree (BST)	3
3.1.1. Algorithms	3
3.1.2. Complexity Analysis	3
3.2. AVL Tree	3
3.2.1. Algorithms	3
3.2.2. Complexity Analysis	3
3.3. Red-Black Tree (RBT)	3
3.3.1. Algorithms	3
3.3.2. Complexity Analysis	3
3.4. Inverted Index	3
3.4.1. Algorithms	3
3.4.2. Complexity Analysis	3
4. Testing and Validation	3
4.1. Unit Testing Method	3
4.1.1. Binary Search Tree (BST)	3
4.1.2. AVL Tree	3
4.1.3. Red-Black Tree (RBT)	3
5. Comparative Analysis	3

¹Escola de Matemática Aplicada, Fundação Getúlio Vargas (FGV/EMAp), email: arthur.oliveira.1@fgv.edu.br

5.1. The Experiment	4
5.2. Memory Usage	4
5.3. Time Complexity	4
6. Conclusion	4
6.1. Summary of Findings	4
7. Source code	4
8. Task Division (Required by the professor)	4
8.1. Arthur Rabello Oliveira	4
8.2. Gabrielle Mascarello	4
8.3. Eliane Moreira	4
8.4. Nicolas Spaniol	5
8.5. Gabriel Carneiro	5

1. Introduction

1.1. Motivation

1.2. Problem Statement

1.3. Overview of Search Trees

2. Data Structures Overview

2.1. Binary Search Tree (BST)

2.2. AVL Tree

2.3. Red-Black Tree (RBT)

2.4. Inverted Index

2.5. Comparison of Properties

3. Implementations

3.1. Binary Search Tree (BST)

3.1.1. Algorithms

3.1.2. Complexity Analysis

3.2. AVL Tree

3.2.1. Algorithms

3.2.2. Complexity Analysis

3.3. Red-Black Tree (RBT)

3.3.1. Algorithms

3.3.2. Complexity Analysis

3.4. Inverted Index

3.4.1. Algorithms

3.4.2. Complexity Analysis

4. Testing and Validation

4.1. Unit Testing Method

4.1.1. Binary Search Tree (BST)

4.1.2. AVL Tree

4.1.3. Red-Black Tree (RBT)

5. Comparative Analysis

5.1. The Experiment

5.2. Memory Usage

5.3. Time Complexity

6. Conclusion

6.1. Summary of Findings

7. Source code

(repository)

8. Task Division (Required by the professor)

8.1. Arthur Rabello Oliveira

Implemented and documented the following functions:

```
1 BinaryTree* createTree(); //bst.cpp
2 static std::string normalise(const std::string& w); //data.cpp
3 std::vector<std::string> list_files_txt_in_path(const std::string &dir_path);
4 SearchResult search(BinaryTree* binary_tree, const std::string& word); //
  tree_utils.cpp
```

8.2. Gabrielle Mascarello

Implemented and documented the following functions:

```
1 std::string read_file_content(const std::string& full_file_path); //
  data.cpp
2 int main(int argc, char *argv[]); //main_bst.cpp
```

8.3. Eliane Moreira

Implemented and documented the following functions:

```
1 InsertResult insert(BinaryTree* binary_tree, const std::string& word, int
  documentId); //bst.cpp
2 void updateHeightUp(Node* node); //tree_utils.cpp
3
4 void test_one_insertion(); //bst_test.cpp
5 void test__left_tree_insertion(); //bst_test.cpp
6 void test_right_tree_insertion(); //bst_test.cpp
7 void test_generic_tree_insertion(); //bst_test.cpp
8 void test_numbers_and_words_tree_insertion(); //bst_test.cpp
9 void test_different_words_same_doc_tree_insertion(); //bst_test.cpp
10 void test_same_word_same_doc_tree_insertion(); //bst_test.cpp
11 void test_same_word_different_docs_insertion(); //bst_test.cpp
12 void test_similar_words_insertion();
13 int main(); //bst_test.cpp
14 void test_createNode(); //test_tree_utils.cpp
```

```
15 void test_createTree(); //test_tree_utils.cpp
16 void test_calculateHeight(); //test_tree_utils.cpp
17 void test_updateHeightUp(); //test_tree_utils.cpp
18 int main(); //test_tree_utils.cpp
```

8.4. Nicolás Spaniol

8.5. Gabriel Carneiro

Implemented and documented the following functions:

```
1 Node* createNode(std::string word, std::vector<int>documentIds, int color
  = 0); //tree_utils.cpp
2 std::vector<std::string> tokenize(std::string filename); //data.cpp
3 void destroy(BinaryTree* binary_tree); //tree_utils.cpp
4 int calculateHeight(Node* root); //tree_utils.cpp
```

