Final Project - Inverted Index and Comparative Analysis

Arthur Rabello Oliveira[1], Gabrielle Mascarello, Eliane Moreira, Nícolas 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]Escola de Matemática Aplicada, Fundação Getúlio Vargas (FGV/EMAp), email: arthur.oliveira.1@fgv.edu.br

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. Complextity Analysis

3.2. AVL Tree

3.2.1. Algorithms

3.2.2. Complextity Analysis

3.3. Red-Black Tree (RBT)

3.3.1. Algorithms

3.3.2. Complextity Analysis

3.4. Inverted Index

3.4.1. Algorithms

3.4.2. Complextity 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:

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

8.2. Gabrielle Mascarello

Implemented and documented the following functions:

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

8.3. Eliane Moreira

Implemented and documented the following functions:

```cpp
InsertResult insert(BinaryTree* binary_tree, const std::string& word, int documentId); //bst.cpp
void updateHeightUp(Node* node); //tree_utils.cpp
```

8.4. Nícolas Spaniol

Implemented and documented the following functions:

```cpp
void test_one_insertion(); //bst_test.cpp
void test__left_tree_insertion(); //bst_test.cpp
void test_right_tree_insertion(); //bst_test.cpp
void test_generic_tree_insertion(); //bst_test.cpp
void test_numbers_and_words_tree_insertion(); //bst_test.cpp
void test_different_words_same_doc_tree_insertion(); //bst_test.cpp
void test_same_word_same_doc_tree_insertion(); //bst_test.cpp
void test_same_word_different_docs_insertion(); //bst_test.cpp
void test_similar_words_insertion();
```

```cpp
int main(); //bst_test.cpp
void test_createNode(); //test_tree_utils.cpp
void test_createTree(); //test_tree_utils.cpp
void test_calculateHeight(); //test_tree_utils.cpp
void test_updateHeightUp(); //test_tree_utils.cpp
int main(); //test_tree_utils.cpp
```

8.5. Gabriel Carneiro

Implemented and documented the following functions:

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