Evgenia Lyjina
Report on final project for Scientific computation II

# Text analyzing

## 1. Introduction

The purpose of the program is to sort a text file's contents in alphanumerical order and to count the appearance of the encountered words. The words have to be transformed to a lower case according to the case study.

The sorting of the content was supposed to be done with a binary tree. which is why I found this problem interesting. I have been studying graphs and sorting algorithms but haven't implemented any larger problems before.

I chose to use red-black tree for this problem because it is a self balancing binary search tree. It’s privilege over regular binary search tree is it’s time complexity for all operations being always O(log(n)). In this case the only used operation is insert, but f.ex search could be also easily implemented.

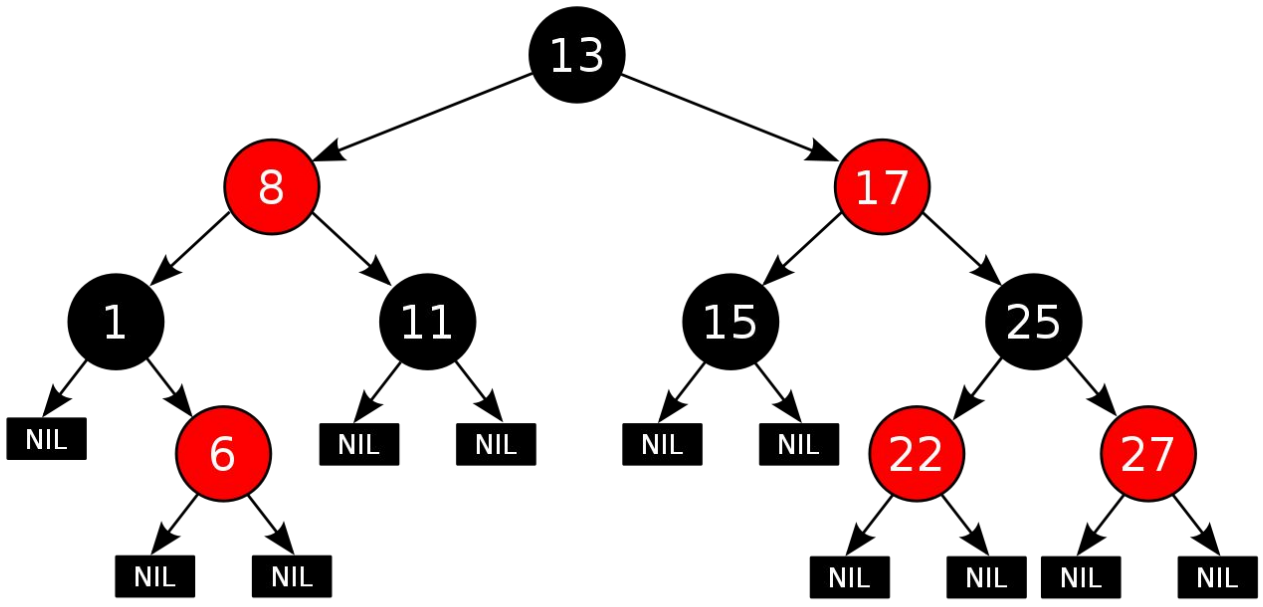
# 2. Methods

The main problem of the program is sorting strings by engineering a binary tree. Other problems are comparing strings, reading from a file and writing to a file. The last problems are more technical ones and will be explained in the next chapter.

## 2.1 Red-black binary tree

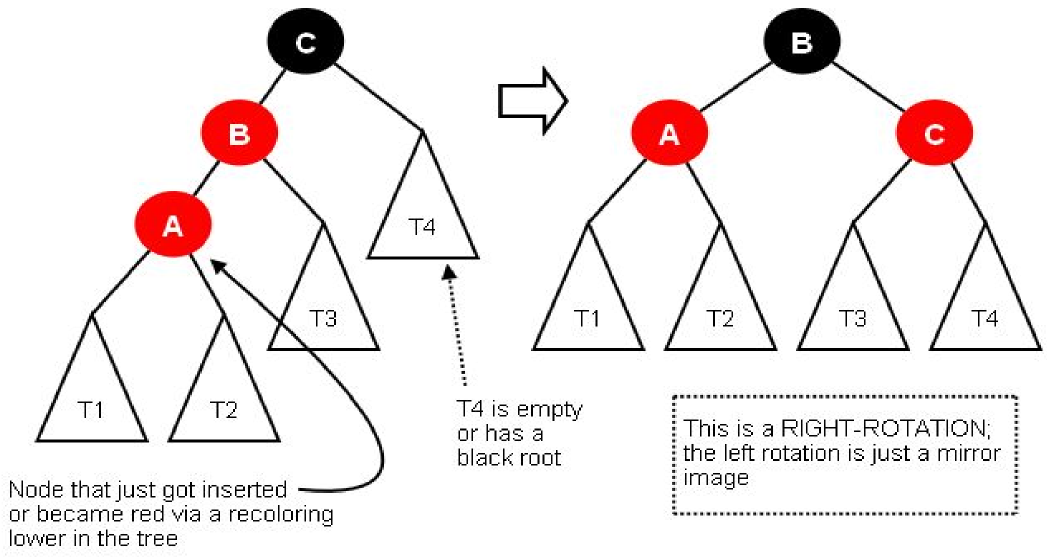
A binary tree is linked data structure linking data elements called nodes. They contain data and a reference to "left" and "right" data elements called “children” and in the case of the red-black tree also a reference to the previous node called “parent”. The top-most node is called the root.

The basic idea of a red-black tree is
1. Every node is either red or black.
2. Every “null” pointer is considered to be pointing to an imaginary black node.
3. If a node is red, then both its children are black.
4. Every direct path from the root to a leaf contains the same number of black nodes.



*Red-black tree (Image from Wikipedia)*

After inserting a new node there might appear conflicts with the rules mentioned above which means that the tree becomes unbalanced. In that case a balancing method(s) must be implemented. The methods are called rotations and depending on the situation either left, right or both rotations are taking place. To satisfy the red-black tree rules we want that the nodes references to children and parents are rewritten and nodes are recolored.


C
B
A
T1
T2
T3
T4
Node that just got inserted or became red via a recoloring lower in the tree
T4 is empty or has a black root
B
A
C
T1
T2
T3
T4
This is a RIGHT-ROTATION; the left rotation is just a mirror image

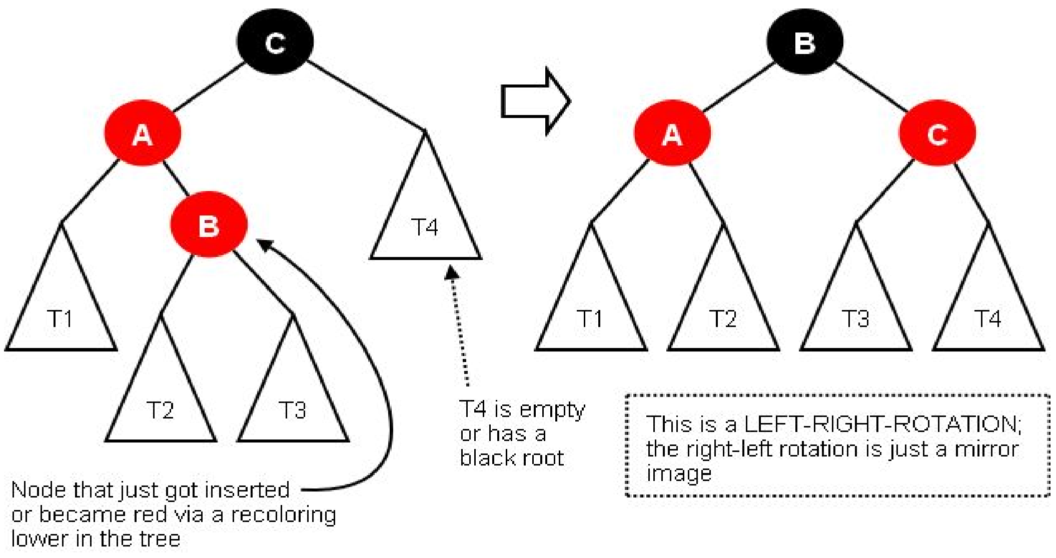

C
A
B
T1
T2
T3
T4
Node that just got inserted or became red via a recoloring lower in the tree
T4 is empty or has a black root
B
A
C
T1
T2
T3
T4
This is a LEFT-RIGHT-ROTATION; the right-left rotation is just a mirror image

*Image from http://cs.lmu.edu/~ray/*

When we follow these rules for building the red-black binary tree we can be sure that our tree is “sorted” in the sense that we have a method to traverse the tree in an alphabetic order.

## 2.2 Tree traversal

At any time of inserting nodes to the tree, the strings can be picked up in the alphabetic order by a tree traversal method. For this purpose an in-order depth-first traversal is implemented. This method traverses through the tree collecting nodes starting from the leftmost leaf, backs up to it’s parent and recurses into the sibling of the left child. The traversal is carried out until the rightmost leaf or node is collected.

The picture below shows the order of the node visitation.

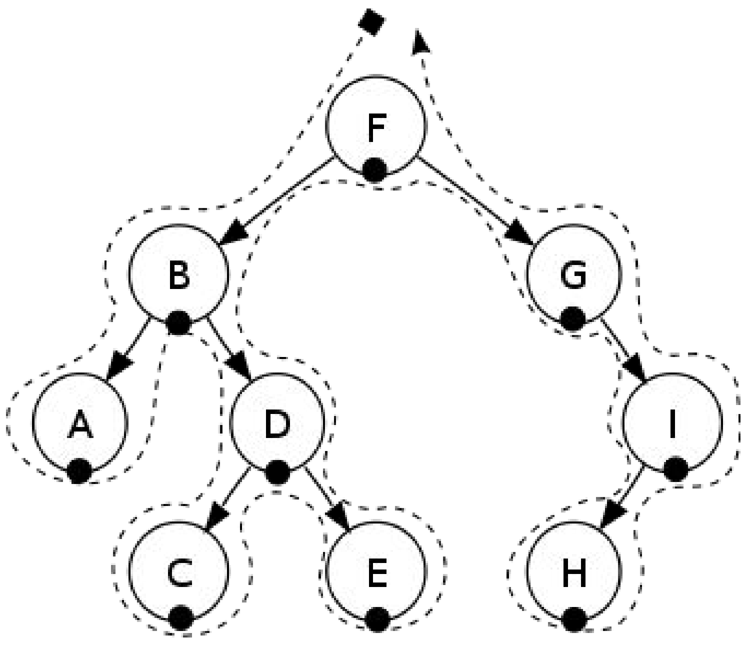


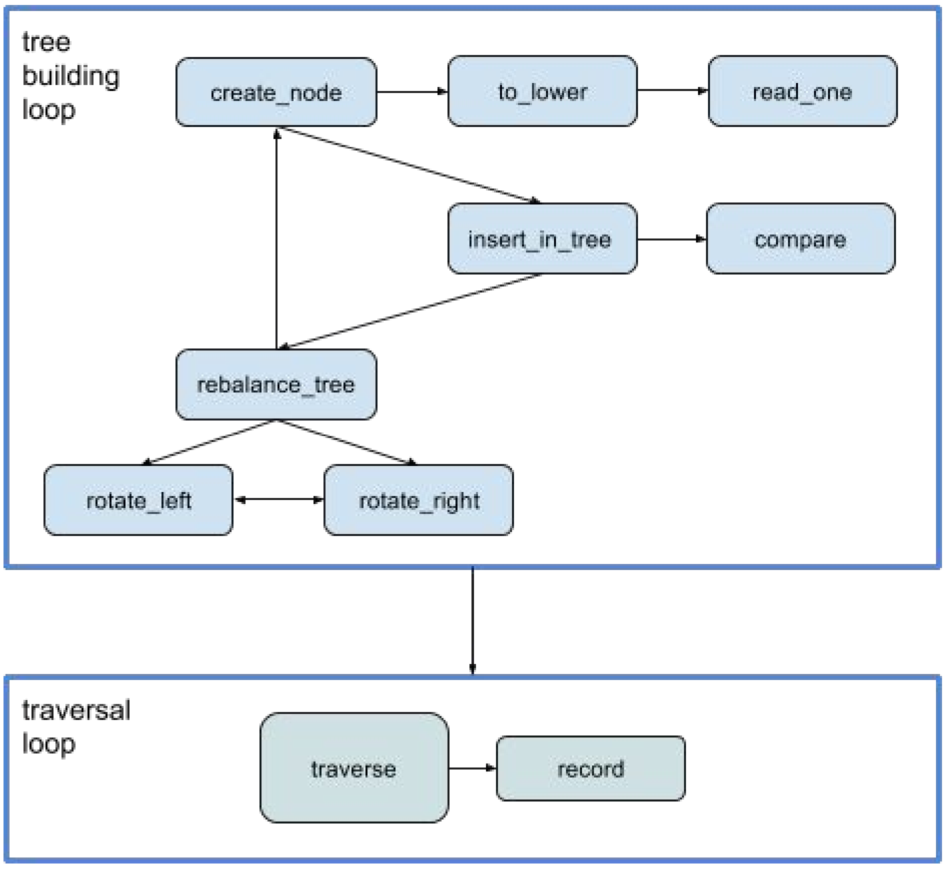
*Image from Wikipedia*

# 3. Implementation

The program consists of five modules and a main program.

The main program takes care of reading the program parameters, opening files, maintaining a loop for tree creation, printing the tree and finally calling the tree traversal method.

Image describing method invocation from Main program loop:



The tree building and traversal methods are strongly based on a program from the book "Algorithms and Data Structures in F and Fortran", by Robin A. Vowels. The modifications are made in handling the nodes’ data. The tree building methods create_node and insert_in_tree, invoke procedures to read and compare words.

The method for reading a file is implemented by picking one character at a time from the file. Characters not counted in a word and thus are considered as string separators are: space, tab, newline and a set of special characters:. , ! " ‘/ ( ) [ ] { } < >*| = ? : ;+^%. The word is considered complete once a separator is encountered.

Before adding the word to a node it is transformed to lowercase. The to_lower procedure has nordic alphabet in two strings, one being upper and the other lowercase. The input string is searched for a capital letter and replaced with the same letter from a lowercase string.

The auxiliary comparing method for inserting in tree uses Fortran's intrinsic compare functions llt() and lgt() to compare two strings and returns integer -1, 0 or 1 depending on the outcome.

All modules and their methods used in program are listed in appendix.

# 4. Compiling and running the program

Program is written and tested on Ubuntu but should work on any operating system with gfortran or any other Fortran 90 compiler. Instructions for Ubuntu and gfortran follow.

Text_analyzer.zip file contains a Makefile which takes care of compiling the program. To compile it you need gfortran installed. Gfortran is the GNU Fortran compiler which is a part of GCC. You can install the package by running:

```
sudo apt-get install gfortran
```

Download text_analyzer.zip to a directory you want to install the program in. Unzip file:

```
unzip text_analyzer.zip -d ./text_analyzer
```

cd to text_analyzer source directory and run `make`.

```
cd text_analyzer/src
make
```

If some other compiler is used, then the order of files must be:
node.f90 word_collection.f90 tree_collection.f90 print_tree_collection.f90 main.f90.

To run a program execute the exe-file with the file to be analyzed as parameter:

```
./text_analyzer.exe path_to_file
```

The output file will appear in the same directory as the original file.

# 5. Results

The program's product is a text file with a list of sorted words and their appearance count. A name of the produced file is a name of the file given as a parameter with _analyze.txt postfix: `somefile_analyze.txt`.

Program also produces Fort.100 file which contains text representation of the binary tree. In some cases also an image of the tree can be produced as a tree.png file in text_analyzer/src directory. For this purpose Graphviz package for linux has been used. Unfortunately with some input characters graphviz gives a syntax error, so this graphical presentation is used merely for debugging and is not a real feature of the program. To use it however the graphviz package needs to be installed and the code for printing in the src/main.f90 has to be uncommented. To install package from command line:

```
sudo apt-get install graphviz
```

## 5.1 Efficiency

The execution time of the program might not be optimal, because the program reads the file one character at the time. On the other hand handling buffers is not necessary which leads to minimal memory use. It would have been worth to test  time usage with different buffer sizes to determine the best efficiency for the program between used memory and time spent to run program.

The time spent inserting into the tree is however optimized by using the red-black tree which has time complexity log(n), where n is the number of words already in the tree.

# 6. Conclusions

Making of a text analyzer was a learning experience also in learning more about Fortran as well as getting to implement a binary search tree.

Future developments could be searching for a particular word, which could be implemented with a tree search algorithm. Another nice feature would be determining the characters not to be included in collection.

## APPENDIX

**Modules and their procedures:**

------------------------------------

node_def

contains the definition of type Node

------------------------------------

GLOBAL

global variables and parameters

------------------------------------

word_collection

| | |
|---|---|
| read_one() | Reads one character at a time from a file and appends it to a string until one of the string separator characters are encountered.<br>Characters which separates strings and thus are not read in are space, tab, newline and a set of special characters:<br>. , ! " '/ ( ) [ ] { } < >*\| = ? : ;+^%.<br>The trimmed string is returned. |
| to_lower(a) | Transforms upper case characters to lowercase. The alphabet is hardcoded as scandinavica keyboard layout.<br>Input is a type of character.<br>Returns string in lowercase. |
| compare(a,b) | Compares alphabetically two strings. Fortran's intrinsic functions llt(a,b) and lgt(a,b) have been used here.<br>Input is type of character.<br>Return integer -1 if a < b, 0 if a = b and 1 if a > b. |
| record(a) | Appends a string and it's appearance into a file.<br>Input is a pointer to a node which data is to be written to a file. |

------------------------------------

tree_collection

| | |
|---|---|
| create_node(a) | This procedure creates a node by reading data with read_one()<br>and setting pointers to children to null.<br>Calls the procedure insert_in_tree. |
| insert_in_tree(node_p) | Searches the right position for node in tree using procedure compare().<br>Input is a pointer to a node to be inserted. |

| | |
|---|---|
| rebalance_tree(node_p) | This procedure rebalances and re-colors the nodes of a tree following an insertion. Calls procedures rotate_left() and rotate_right().<br>Input is a pointer to a (red) node that has been inserted in the tree. |
| rotate_left(node_p) &<br>rotate_right(node_p) | Performs a left or right rotation of a branch of a Red-Black tree.<br>Input is a pointer to the node about which a branch of the tree is to be rotated. |
| traverse(node_p) | Recursive subroutine which searches the tree for sorted nodes.<br>Input is a pointer to a node on a path. |

-------------------------------------

print_tree_collection

Contains recursive subroutine print_tree() to produce a string for graphviz. Strings are found by pre-order depth-first traversal.