FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Lab 5: Red-Black Trees

1 Introduction

Binary search trees are a commonly used data structure. However, these search trees have a major disadvantage as they will become unbalanced in certain scenarios. The efficiency of search queries depends on the depth of the tree. A solution to this problem are Red-Black trees that contain mechanisms to limit the height.

To gain a better understanding of the Red-Black tree, the following visualization can be used: https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

2 Startcode

For this implementation there are 2 classes given in rzwknoop.h and rzwboom.h.

Before you start, thoroughly analyze these 2 files. What is the relationship between RZWboom, RZWknoop and unique_ptr? How do you request data from an RZWboom object from the underlying RZWknoop? What is the function geefBoomBovenKnoop for?

If necessary, refresh the concepts of unique_ptr and how the functions make_unique and move work before starting this lab.

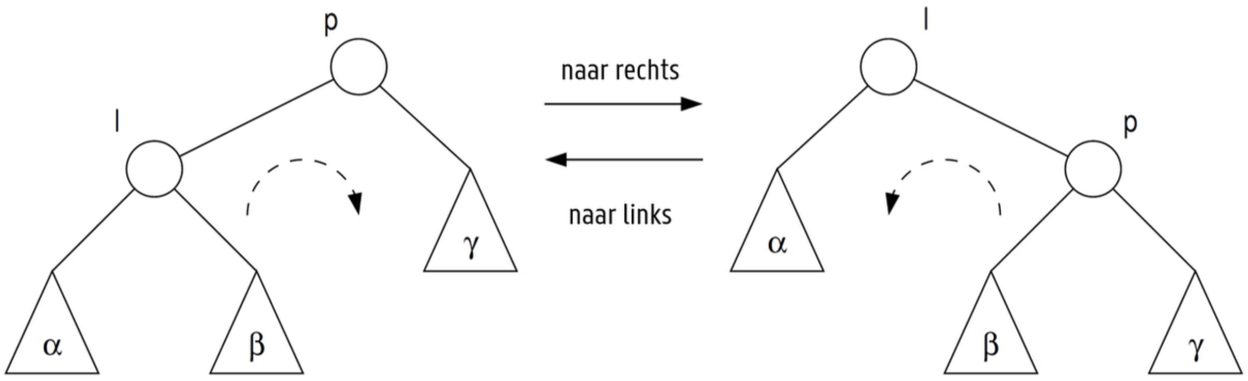
Tip: You can export and view the trees during this lab via the tekenAlsBinaireBoom function and display them in VSCode using the extension Graphviz (dot) language support for Visual Studio Code.

3 Basic Red-Black tree operations

First, some basic operations will have to be added to the RZWBoom class:

- In the class RZWBoom, implement the search function that can look up a key in the tree.
- After adding a new node to the tree you will have to restore the red-black balance of the tree by rotating the tree. It's best to write the rotate function now. To find out which pointers need to be adjusted, use the following figure:
- Now add the nodes in a "bottom-up" manner to the red-black tree, via the function add.

  Tip: use the search and rotate functions here.



4 Application

A library plans to create a digital search catalog of all the words in their books and texts. Since there is no control over the order of adding words and they want to limit the number of operations per search, they choose to create a red-black tree per book.

- We test whether this approach will work properly with Ovid's text. When creating the tree, we read in all words, see if they are already in the tree and add non-duplicate keys. In how many steps can it be determined afterwards whether a certain word is in the text?
- Bonus assignment: We also want to remove words from the book's search tree. Implement a delete functionality that makes sure that the tree is still balanced.
- Bonus assignment: The library receives a lot of questions from customers on how often some words appear in a text. Adjust the template, so that you can store additional data in the nodes in addition to the key.