Practicum 3 — Trees

Dit practicum gaat over Binary Search Trees. De structuur van de geleverde bestanden is als volgt:

De klasse `BinTree` in `bintree.py` implementeert een Binary Search tree. Het meeste werk wordt echter gedaan door de klasse `BinNode` welke de meeste operaties uitvoert.

De klasse `TestTree` in `testtree.py` is de tester die controleert of de implementaties naar behoren functioneren. Deze maakt gebruik van `TreeVerifier` om te controleren of de gegeven bomen aan de binaire zoekboom eigenschappen. Verder maakt deze gebruik van `AsciiTreeVisualizer`om een ASCII weergave van de boom te geven. (De code van al deze bestanden is niet zo interessant, daar men vooral met testfiles zal werken. Dit zijn bestanden die instructies aan `TestTree` geven voor wat voor operaties er gedaan moeten worden. Zie `testfile` voor een beschrijving van de syntax.)

De `test*` bestanden zijn bedoeld om verschillende aspecten van de bomen te testen.

Ook in dit practicum wordt gebruik gemaakt van het natural ordening principe welke in het vorige practicum uitgelegd is.

BinTree In deze opgave gaan we `BinTree` implementeren. Veel code is al gegeven en u wordt aangeraden niet teveel tijd te besteden aan alle code onderin het bestand (er wordt aangegeven waar). Zie de bijgeleverde documentatie (`doc/index.html`) voor de details over de methodes die daar staan.

Het is belangrijk om te onthouden dat elke node een link naar zijn parent heeft, en deze link moet te allen tijde correct zijn.

Wrappers De klasse `BinTree` heeft een redelijk aantal wrapperfuncties welke gebruikt kunnen worden bij de implementatie:
`attach_left(node)` en `attach_right(node)` van `BinNode` kunnen worden gebruikt om snel nodes aan de linkerkant of rechterkant van een node te plaatsen. Deze zetten zowel de parent- als de child-links correct. Deze werken ook om een kind te verwijderen door ze aan te roepen met `None`.
`replace_child(old_child, new_child)` van `BinTree` kan worden gebruikt om een node te vervangen. Voorwaarde is dat `old_child` een geldige parent-child link heeft. Deze werkt ook om de rootnode te vervangen.

Voorbeeld van gebruik in een methode van `BinNode`:

```
self.attach_left(new_node)
self.attach_right(None)
self.tree.replace_child(old_child, new_child)
```

Testen Om uw implementatie te testen kunt u het volgende commando gebruiken: `python3 testtree.py <testfile>`.

3.1.1 Implementeer de methode `BinNode.add(element)`. Deze methode moet (recursief) de juiste locatie voor de toe te voegen node uitzoeken en deze daar plaatsen. *Het is belangrijk dat NIET direct de constructor van `BinNode` aangeroepen wordt. Gebruik daarvoor in de plaats `self.tree.create_node(element, parent_node)`.*

Testen kan met de testfile `testadd`.

3.1.2 Implementeer de methodes `rotate_left()` van `BinNode` en `rotate_right()` van `BinNode`. Deze roteren de boom respectievelijk linksom en rechtsom. Let er hierbij op dat de fysieke nodes geherstructureerd moeten worden, niet alleen de waardes.

Testen kan met de testfile `testrot`.