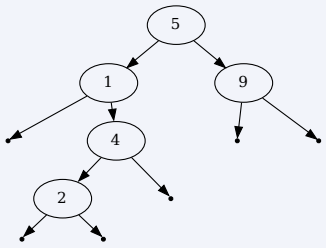
Übung 9

Abgabe: 26. Juni 2023, 18:00 Uhr

Aufgabe 9.1: oben-unten [10 Punkte, Link zur Abgabe] In dieser Aufgabe suchen wir einen Algorithmus, der einen binären Suchbaum ebenenweise ausgibt.

Listing 1: Beispiel

```
1 > cat example.in
2 5
3 1
4 4
5 2
6 9
```



Der vorgegebene Code fügt die Schlüssel in der Reihenfolge in einen binären Suchbaum ein, in der sie in der Eingabedatei gegeben sind.

Ihre Aufgabe ist es, den Suchbaum von oben nach unten Ebene für Ebene auszugeben. Jede Ebene soll von links nach rechts ausgegeben werden, jeder Knoten in einer Zeile.

Listing 2: Ausgabe

```
1 > ./levels
2 5
3 1
4 9
5 4
6 2
```

Das Repository enthält bereits einige Datenstrukturen: Wählen Sie eine oder mehrere Datenstrukturen für Ihre Implementierung aus – und zwar so, dass Sie eine möglichst gute asymptotische Worst-Case-Laufzeit erreichen. Dabei dürfen Sie die Datenstruktur(en) an Ihre Bedürfnisse anpassen. Verändern Sie dabei aber die grundlegende Funktionsweise der Datenstruktur(en) nicht.

Notieren Sie in `Frage.md`, welche Datenstruktur(en) Sie gewählt haben.

Hinweis: Sie müssen ggf. das Makefile anpassen.

Aufgabe 9.2: links-rechts [10 Punkte, Link zur Abgabe] In dieser Aufgabe erweitern wir einen Binären Suchbaum um eine Links- und eine Rechtsrotation, wie wir sie aus der Aldat kennen, vgl. Kapitel 4.3.1 im Skript.

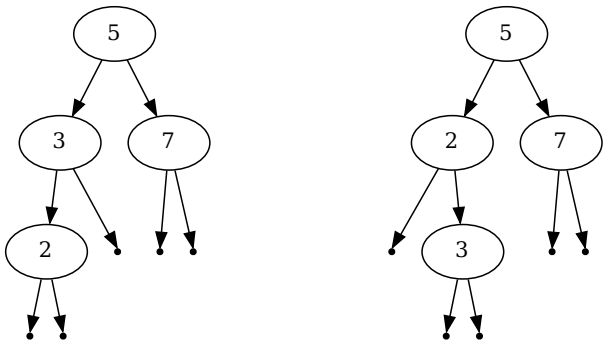
Im Repository finden Sie eine Implementierung eines binären Suchbaumes. Implementieren Sie die markierten Methoden für eine Links- und eine Rechtsrotation.

Die Eingabedatei beschreibt, wie der binäre Suchbaum aufgebaut und verändert werden soll:

Listing 3: Beispieleingabe example.in

```
1 a 5
2 a 3
3 a 7
4 a 2
5 p
6 r 3
7 p
```

In diesem Beispiel werden die Knoten 5, 3, 7 und 2 eingefügt. Anschließend wird der Baum ausgegeben, eine Rechtsrotation um Knoten 3 durchgeführt und der Baum dann erneut ausgegeben.



Das vorgegebene Programm liest bereits die Eingabedatei ein und ruft die passenden Methoden auf. Das Programm erzeugt Ausgabe im *graphviz-DOT*-Format[1].

Listing 4: Umwandlung der Ausgabe in PDFs

```
1 > ./treetest < example.in > example.dot
2 > dot -O -Tpdf example.dot > example.pdf
```

Sie erhalten ein PDF für jeden Baum in `example.dot`. Dazu muss `graphviz` installiert sein. Wenn Sie `eval.py --pdf` ausführen, geschieht diese Umwandlung automatisch.

[1] https://graphviz.org/doc/info/lang.html

Aufgabe 9.3: tree-queues [10 Punkte, Link zur Abgabe] In dieser Aufgabe ist das Ziel, einen binären Baum zu verwenden, um die Schnittstelle einer Prioritätswarteschlange zu implementieren.

Im Repository finden Sie die Implementierung eines binären Baumes sowie eine Headerdatei `prioqueue.h` mit einer Schnittstellendefinition einer Prioritätswarteschlange. Implementieren Sie die Methoden der Prioritätswarteschlange in der Datei `prioqueue.c`.

- Ergänzen Sie als ersten Schritt den Binärbaum um eine Methode `minimum` und eine Methode `maximum`, die den kleinsten bzw. größten Schlüssel im Baum zurückgeben.
- Verwenden Sie dann den Binärbaum, um die Prioritätswarteschlange zu implementieren. Greifen Sie dabei nicht direkt auf die Attribute des Binärbaums zu, sondern rufen Sie nur die Methoden des Baumes auf.
- Achten Sie auf sauberes Speichermanagement.
- Beantworten Sie außerdem die Fragen in `Frage1.md`, `Frage2.md` und `Frage3.md`.