Exercises for

Database Implementation

Elite Graduate Program Software Engineering

Florian Funke (florian.funke@in.tum.de)

Assignment 4

Excercise 1

Implement a B$^+$-Tree index for your database system on top of the segments. Your tree should ...

> ... support different (opaque) key[1] types. Parameterize the B$^+$-Tree with a key type and a comparator. You can assume that all key types have fixed length.
>
> ... offer the following **reentrant** operations
>
> - `insert` Inserts a new key/TID pair into the tree.
> - `erase` Deletes a specified key. You may simplify the logic by accepting underfull pages.
> - `lookup` Returns a TID or indicates that the key was not found.
> - `lookupRange` Returns an iterator that allows to iterate over the result set.
>
> ... support graphical output method `visualize`, e.g. via Graphviz/dot[2] (see note below).

Use the concurrency control techniques from the slides "Concurrent Access (2)" and "Concurrent Access (3)".

Note

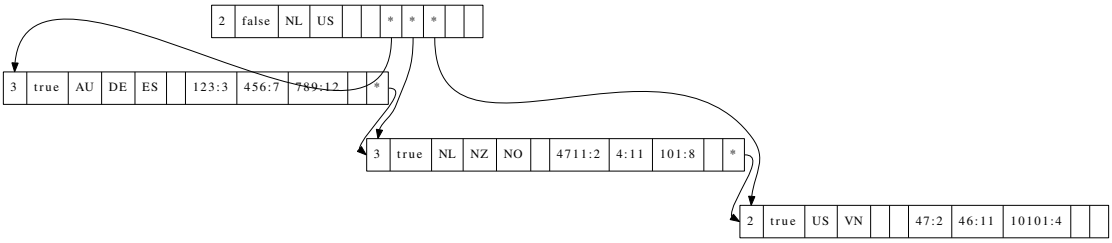
The following example Graphviz/dot code could be the output of a `visualize` method. It can then be rendered using the command `dot -Tpng tree.dot -o tree.png` (your program does not need to invoke the rendering automatically). You can either produce the output yourself, or employ a library.

```
digraph myBTree {
node [shape=record];
node0 [shape=record, label=
 "<count> 2 | <isLeaf> false | <key0> NL | <key1> US | <key2> | <key3> | <ptr0> * | <ptr1> * | <ptr2> * | <ptr3> | <ptr4>"];
leaf1 [shape=record, label=
 "<count> 3 | <isLeaf> true  | <key0> AU | <key1> DE | <key2> ES | <key3> | <tid0> 123:3 | <tid1> 456:7 | <tid2> 789:12 | <tid3> | <next> *"];
leaf2 [shape=record, label=
 "<count> 3 | <isLeaf> true  | <key0> NL | <key1> NZ | <key2> NO | <key3> | <tid0> 4711:2 | <tid1> 4:11 | <tid2> 101:8 | <tid3> | <next> *"];
leaf3 [shape=record, label=
 "<count> 2 | <isLeaf> true  | <key0> US | <key1> VN | <key2> | <key3> | <tid0> 47:2 | <tid1> 46:11 | <tid2> 10101:4 | <tid3> | <next>"];

node0:ptr0 -> leaf1:count;
node0:ptr1 -> leaf2:count;
node0:ptr2 -> leaf3:count;
leaf1:next -> leaf2:count;
leaf2:next -> leaf3:count;
}
```

[1] Your tree does not need to support non-unique entries.

[2] `www.graphviz.org`



In this example, the inner node (root) consists of the count (the number of entries), a flag indicating if it is a leaf or not (false), 4 key slots (two of which are used) and then 5 child-pointer slots (three of which are used). The leaves consist of the count, the flag, 4 key slots, 4 TID slots and a next pointer slot.