Algorithms & Datastructures
Laboratory Exercise Sheet 8

Wolfgang Pausch <wolfgang.pausch@uibk.ac.at>
Heiko Studt <heiko.studt@uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>
Tomas Vitvar <tomas.vitvar@uibk.ac.at>

May 11th, to be discussed on May 18th

Exercise 1) B-trees The B-Tree is one of the major data structures in computer science, especially for databases. It is not optimized on comparison or swap complexity, but on block-read complexity as it is very costly to search for a block (up to multiple magnitudes against all other concerning operations), aswhile the subsequential read of a big size (a block) does not take much time.

k-B-Trees are similar to the brother trees in the lecture and consist of "lines" of keys/references as the nodes. The nodes contain $2 * k$ sorted keys and $2 * k + 1$ references to children. All leaves are at the same level. Though in general it can contain the same element twice, we are concerning only dictionaries with unique elements.

The size of k is chosen in that way, that a node fills (at best) exactly one (or a multiple of) blocks on the hard-disk. (It is one parameter to optimize databases.) Some of the interesting aspects are, the height is of small magnitude and there will be few (costly) re-balancing.

To summarize a k-B-Tree is defined as follows:

- The root may consist of 0 (sometimes 1) up to $2 * k$ keys
- All other nodes consist of at least k up to $2 * k$ keys
- All leaves are at the same level.

Insertion goes as follows: You insert at a leaf. If the leaf get size $2k + 1$, you divide it into two leafes of size k. The median is inserted into the parent node. This is recursed upwards, as the parent node could also reach size $2k + 1$. If the root exceeds its maximal size, you generate a new root of size 1.

Exercises:

1. Let $k = 2$. Do the following operations: (give graphs)
   `insert(3)`, `insert(9)`, `insert(12)`, `insert(55)`, `insert(1)`, `insert(3)`, `insert(42)`, `search(12)`, `delete(3)`, `delete(9)`, `delete(60)`
2. Proof by induction:
   k-B-Trees with height $h > 0$ (height(root) is 0) have at least $2 * (k + 1)^{(h-1)}$ leaves.

Exercise 2) Amortized costs

1. Do an amortized cost analysis for the lecture's ArrayStack implementation. (Push)
2. Why one could be interested in an amortized analysis? It is already in O(N), so what!?

Exercise 3) Hashing

1. Do the following in the given order both for (a) closed hashing (open addressing) with linear probing, (b) open hashing. Each have an array size 10 and are using the modulo operator for locating the actual array block to insert into.
   Assume the given objects have got the following hashes, and they are equal if and only if these hashes are the same. (why is this a special case?).

   - Insert: 50, 2, 34, 22, 48, 54, 5, 17, 26
   - Delete: 2, 33, 54
   - Insert: 12, 33

   At each step, state clearly the status of the datastructure.
2. There are open hash and closed hash data structures. List the differences.
3. What kind of hash (open/closed/linear probing/...) is used in Java (`java.util.HashMap<K,V>`, `java.util.HashTable<K,V>`, `java.util.LinkedHashMap<K,V>`)?
4. Hashing can be used in some unexpected situations. This is one used case.
   Write a program which saves a list of Strings into some files decided on its hash. The files are some kind of dictionaries, it is faster to have several files.

   (a) Is this open or closed hashing?

   (b) Design such a write-only data structure in Java.
   Notes: You don't need threads. For files, please have a look into PrintWriter (`http://www.java2s.com/Tutorial/Java/0180__File/WritelinesoftexttofileusingaPrintWriter.htm`).
   This time there is no skeleton given, because the task is to build up one.

   (c) Write several values into the files and explain what happens.

   Background: If you want to introduce thread parallelism in file-based dictionaries, you will need several files (because you need several so-called locks, in this case they are file-based). You can decide on which file to read/write by using a method like above.