Algorithms & Datastructures
Laboratory Exercise Sheet 9

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 25th, to be discussed on June 1st

Exercise 1) Questions and Answers Answer the following questions (true or false).

- ☐ Quick sort has a worst-case running time of $\Theta(n^2)$.
- ☐ Omitting the pre-processing step of the Quicksort given in the lecture has no impact.
- ☐ Radix sort sorts numbers on their most significant digits.
- ☐ Loop invariants help us understand complexity of algorithms.
- ☐ For the loop invariant we state a condition that must be true before the first iteration of the loop while it remains true before subsequent iteration, and a condition when the loop terminates.
- ☐ Pre-order binary tree traversal, which prints a number of a node key in each step, prints the lowest number first.
- ☐ Binary tree conditions is $p.left.key < p.key > p.right.key$.
- ☐ A minimum $min(p)$ of node p is in a leftmost leaf of the $p.left$ subtree.
- ☐ Successor of a node p in a binary tree can always be determined using $min(p.right)$
- ☐ In an AVL tree, for each node x, the heights of the left and right subtrees of x always differ by at most 1.
- ☐ Removing a node from an AVL tree always requires rotation.
- ☐ Rotation in an AVL tree always changes the height of the tree.
- ☐ In a B-tree, an internal node x that contains $x.n$ keys always has $x.n+1$ children.
- ☐ All leaves in a B-tree are in the same depth.
- ☐ Every node in a B-tree with the minimum degree of 2 must have at least 1 key and may contain at most 4 keys (therefore the internal node may have at most 4 children).

Exercise 2) Hashing Consider the following two classes for triples of arbitrary types, and lists of integers.

```
public class Triple<X,Y,Z> {

   X x;
```

```
    Y y;
    Z z;

    public boolean equals(Object other) {
        if (other instanceof Triple) {
            Triple<X,Y,Z> t = (Triple<X,Y,Z>)other;
            return this.x.equals(t.x) && this.y.equals(t.y) && this.z.equals(t
                .z);
        } else {
            return false;
        }
    }

}
```

```
public class List {
    Element root;

    public boolean equals(Object other) {
        if (other instanceof List) {
            Element one = this.root;
            Element two = ((List)other).root;
            while (one != null && two != null) {
                if (one.x != two.x) {
                    return false;
                }
                one = one.next;
                two = two.next;
            }
            return (one == null && two == null);
        } else {
            return false;
        }
    }
}


class Element {
    int x;
    Element next;

    Element(int x, Element next) {
        this.x = x;
        this.next = next;
    }
}
```

Here, equality is defined in the standard way, i.e., two Triple-Objects are identified iff they represent the same triple, and similarly for lists.

1. Determine whether universal hashing is applicable or not, using triples or lists as type of the keys. Give a short explanation of your answer.
2. Implement for both classes universal hashing (if possible), or some other sensible hash-function, otherwise.