Universität Innsbruck - Institut für Informatik
Prof. Clemens Ballarin, Robert Binna, Friedrich Neurauter, Francois Scharffe and Sarah Winkler

27 May 2008

Proseminar Algorithmen und Datenstrukturen

Exercise Sheet 9

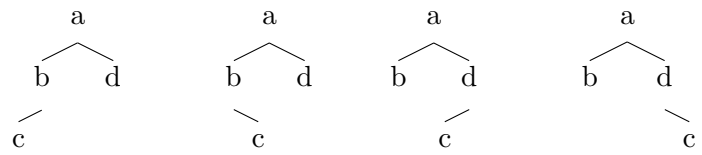
Exercise 1 (AVL)

a) Give AVL trees of height h ($h = 0, \ldots, 4$) with minimal number of nodes.

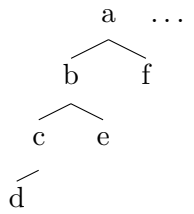
h=0

h=1

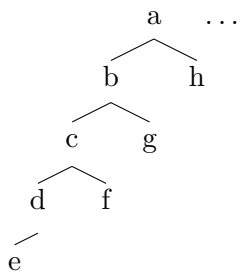
h=2



h=3



h=4



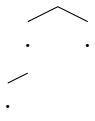
b) Give a construction that for any height h yields an AVL tree T_h with minimal number of nodes. Argue that your construction indeed yields a tree where the number of nodes is minimal.

An inductive construction of AVL trees is done as follows:

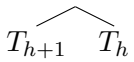
T_0:

T_1:

T_2:



T_{h+2}:



A minimal AVL tree of height $h+2$ is constructed by adding minimal AVL trees of height $h+1$ as the left child and of height h as the right child of a new root.

c) Give a closed form for the number N_h of nodes in T_h. Hint: from your construction, derive a recursive definition of N_h. Then express N_h in terms of the Fibonacci sequence F_i.

$N_h = N_{h-1} + N_{h-2} + 1$
This is equivalent to:

$$N_h + 1 = (N_{h-1} + 1) + (N_{h-2} + 1) \tag{1}$$

By defining $U_h := N_h + 1$ with initializers $U_0 = N_0 + 1 = 2$ and $U_1 = 3$, we get

$$U_h = U_{h-1} + U_{h-2} \tag{2}$$

which satisfies the definition of the Fibonacci sequence:

$$F_i = F_{i-1} + F_{i-2} \quad F_0 = 0, F_1 = 1 \tag{3}$$

However, due to the different initializers for U and F, U is not exactly the Fibonacci sequence but rather $U_h = F_{h+3}$ (easy!). Hence, we can express N_h in terms of the Fibonacci sequence as follows:

$$N_h = F_{h+3} - 1$$

Knowing the closed form

$$F_h = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^h - \left(\frac{1-\sqrt{5}}{2} \right)^h \right]$$

for the Fibonacci sequence, we immediately get a closed form for N_h:

$$N_h = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right] - 1 \tag{4}$$

Exercise 2 (Bucket sort)

a) Give the pseudo code for the bucket sort algorithm data structure and useful operations. You will consider numbers having k digits or less, digit values $\in 0, \ldots, b-1$.

b) Give the pseudo code for a recursive bucket sort algorithm. You will start to sort the most significant digit first.

The numbers are of base b and can can have up to k digits. A bucket is represented as a list of integers:

Listing 1 bucket data structure

```
1: bucket is declared as array 0..b − 1 of lists of integer;
```

The needed operations are thus the classical list operations: add, size, append, getElement. A *sig* variable represent the significant digit for the current bucket.

c) How many buckets does your algorithm store simultaneously at a given time? Is it possible to do better? Explain how. It is possible to reduce the number of buckets by starting to sort on the less significant digit.

Exercise 3 (Insertion and Selection algorithms)

a) Implement in C the insertion and selection sorting algorithms. Develop a user interface in order to select the size and randomly generate values for an array, print the array, check if the array is ordered and order the array using one of the algorithms. See the program `sort.c`.

b) Stable algorithms does not change relative position of elements with equal values. Are the selection and insertion algorithms stable ? Depends on the implementation you realized, they are stable in our implementation.

Listing 2 bucketSort

Input: the list to be sorted *list*, the significant digit *sig*, the sorted list *sorted*.
Output: the sorted list *sorted* updated with *list*.

```
 1: begin
 2:    i := 0;
 3:    while i < b do
 4:       bucket[i] := empty;
 5:       i := i + 1;
 6:    i := 0;
 7:    if sig > 0 then
 8:       while i <size(list) do
 9:          add(bucket[(list(i)/b^(sig−1))%b],getElement(list,i));
10:          i := i + 1;
11:       i := 0;
12:       while i < b do
13:          if size(bucket[i])> 1 then
14:             bucketSort(bucket[i],sig − 1,sorted);
15:          else
16:             append(sorted,bucket[i]);
17:          i := i + 1;
18:    else
19:       append(sorted,list);
20: return(sorted);
21: end
```