

Lab classes - Week 4

In the lab-classes this week we experiment with *binary search trees*.

Update the environment: In your existing directory CS-242-Algorithms:

```
git checkout master
git pull git://github.com/OKullmann/CS-242-Algorithms.git master
```

Or if this directory (repository) doesn't exist (anymore), resp. you want to create a new repository, create a new clone by

```
git clone git://github.com/OKullmann/CS-242-Algorithms.git
```

Using the second form, in the following weeks by `git pull` you will pull from OKullmann's Github repository. Now change to the subdirectory for week 4:

```
cd CS-242-Algorithms/200910/Week04/
```

Basic setup: Today (as well as in the following two weeks) no files are to be changed, but we

1. read the code
2. run the code on small examples, to understand the algorithms
3. perform some experiments.

Compilation is done by

```
Week04> make
```

The executables produced are

```
BinaryTrees_Output
BinaryTrees_Search
BinaryTrees_Statistics
```

First with `BinaryTrees_Output` hands-on experience with small examples is to be gained. Then via `BinaryTrees_Search` some feel for the runtime of larger examples shall be developed, while finally the height of "random" binary search trees is investigated in R using `BinaryTrees_Statistics`.

Comments on the code The easiest form of class `Tree` (implementing the functionality of a dynamic class according to the book) is in `BinaryTrees_simplified.hpp`. The main differences to Java are:

- In C/C++ *objects* and *pointers* are to be distinguished. Given a pointer `p` to some object with member, say, `key`, one uses `p->key`. While if `x` is some object with member `key`, then one writes `x.key`.
- Garbage collection is most of the time not used in C++, and instead many tools are available for the precise and efficient destruction of objects. For trees, we use the simplest model which destroys all nodes, using preorder traversal, upon destruction of the tree (this is done by the *destructor* “`~Tree`”).

Understanding the construction of binary search trees `BinaryTrees_Output` allows you to enter a sequence of keys, where then data on the tree is output, and finally search queries can be entered (here after cursor-return has been entered in the second line, via `Ctrl-D` the input of the input stream is signalled):

```
Week04> ./BinaryTrees_Output
1 4 2 3 8 5
The full tree:
0x804b008:          0          0 0x804b020          1          1
0x804b038: 0x804b020          0 0x804b050          2          3
0x804b050: 0x804b038          0          0          3          4
0x804b020: 0x804b008 0x804b038 0x804b068          4          2
0x804b080: 0x804b068          0          0          5          6
0x804b068: 0x804b020 0x804b080          0          8          5

Height = 3
Inorder walk:
(1, 1) (2, 3) (3, 4) (4, 2) (5, 6) (8, 5)
Minimum = 0x804b008
Maximum = 0x804b068

Searching for keys:
4
search for 4 yields 0x804b020
predecessor of 0x804b020 is 0x804b050, successor is 0x804b080
7
search for 7 yields 0
```

The output under “full tree” shows all nodes (one per line), in inorder traversal, with first the tree pointers `parent`, `left` and `right`, then the key, and then the satellite data, which here is the count of the item in the input stream. Tasks (**answers to be shown to the postgrads**):

1. Understand the above example, drawing the tree on paper.
2. Enter your own examples.

3. How to create the most extreme forms of degenerated trees?
4. In what order do we have to enter numbers from 1 to 15 so that the resulting binary search tree has minimum height 3?

Experiencing run times Next we consider application `BinaryTrees_Search`:

```
> ./BinaryTrees_Search 1000 0
User time usage: 0.00e+00s
Height = 999
```

Searching for keys:

```
1
User time usage: 0.00e+00s
count = 1
1001
User time usage: 0.00e+00s
key not found
```

```
> ./BinaryTrees_Search 10000 1
User time usage: 0.00e+00s
Height = 31
```

Searching for keys:

```
100
User time usage: 0.00e+00s
key not found
101
User time usage: 0.00e+00s
key not found
102
User time usage: 0.00e+00s
key not found
103
User time usage: 0.00e+00s
key not found
104
User time usage: 0.00e+00s
count = 8020
```

The first parameter is the number N of nodes in the tree automatically generated, the second parameter is either 0 or 1, where in the first case the keys generated are $1 \dots 1000$, while in the second case random integers from 0 to $N - 1$ are created. The task here is to get a feeling for the behaviour, and perhaps to guess the asymptotic behaviour of the run time in these two cases (though there are some complications here).

Running Experiments Finally, by `BinaryTrees_Statistics` the behaviour of the height for “random” binary search trees is to be explored. The two parameters of the program are the the

maximum size of search trees, and how many search trees per size are to be produced (recall, these things are random); for example:

```
> ./BinaryTrees_Statistics 100000 1 > data_100000_1  
> ./BinaryTrees_Statistics 100000 5 > data_100000_5
```

Such data you read into R by

```
E = read.table(``data_100000_1``')
```

while you plot it by `plot(E)`. **Your task is to find out the order of the growth rate** by plotting additional curves

```
lines(E$n, f(E$n))
```

for appropriate functions f . Show your best approximating function to the postgrads.