3 Trees: traversal and analysis of standard search trees

Summer Term 2011

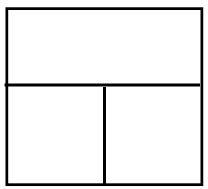
Jan-Georg Smaus

Albert-Ludwigs-Universität Freiburg

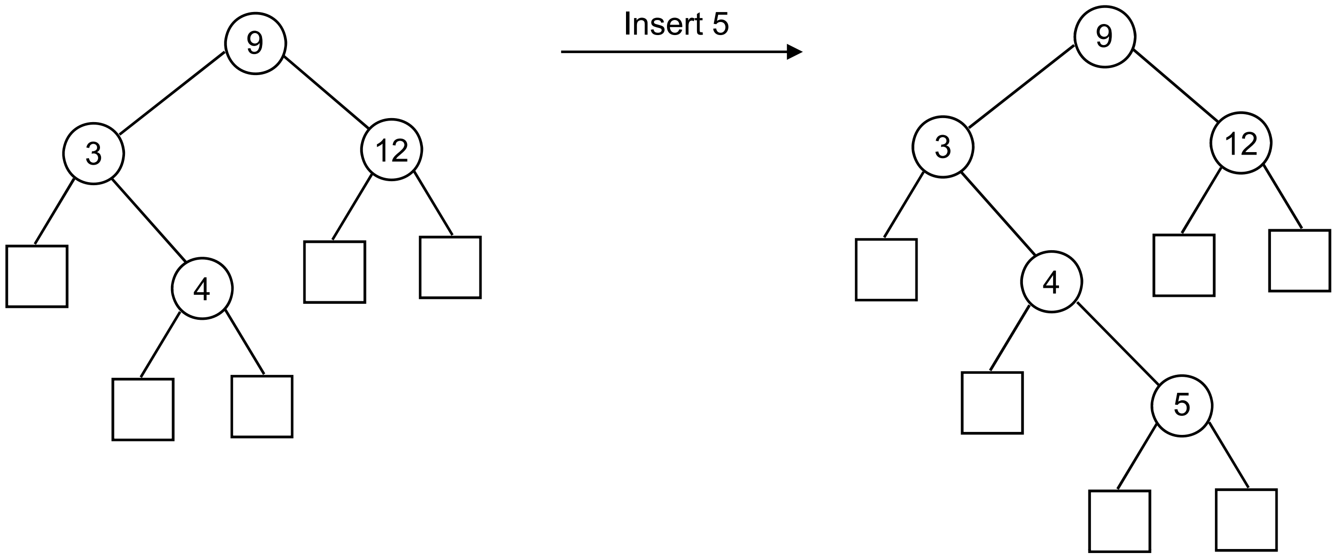
Binary Search Trees

- Binary trees for storing sets of keys (in the internal nodes of trees), such that the operations
  - find
  - insert
  - delete (remove)

  are supported.
- Search tree property: All keys in the left subtree of a node *p* are smaller than the key of *p*, and the key of *p* is smaller than all keys in the right subtree of *p*.
- Implementation:



Standard binary search trees (8)



- Tree structure depends on the order of insertions into the initially empty tree
- Height can increase linearly, but it can also be in O(log n), more precisely $\lceil \log 2\,(n+1) \rceil$.

Traversal of trees

Traversal of the nodes of a tree

- for output
- for calculating the sum, average, number of keys ...
- for changing the structure

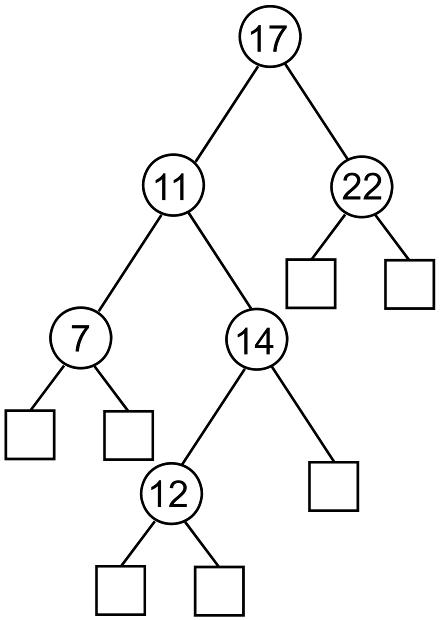
Most important traversal orders:

1. Preorder = NLR (Node-Left-Right)
   first visit the root, then recursively the left and right subtree (if existent)
2. Postorder = LRN
3. Inorder = LNR
4. The mirror image versions of 1-3

Preorder

Preorder traversal is recursively defined as follows: Let p be the root.

- Visit *p*,
- traverse the left subtree of *p* in preorder,
- traverse the right subtree of *p* in preorder.



Preorder implementation

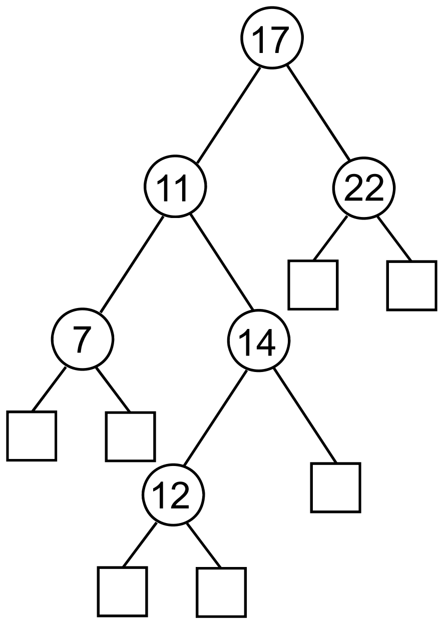
```
// Preorder Node-Left-Right
   void preOrder (){
       preOrder(root);
       System.out.println ();
   }
   void preOrder(SearchNode n){
       if (n == null) return;
       System.out.print (n.content+" ");
       preOrder(n.left);
       preOrder(n.right);
   }

   // Postorder Left-Right-Node
   void postOrder(){
       postOrder(root);
       System.out.println ();
   }
   // ...
```

Inorder

- The traversal order is: first the left subtree, then the root, then the right subtree:

```
 // Inorder Left-Node-Right
void inOrder(){
    inOrder(root);
    System.out.println ();
}
// ...
```

Example



Preorder:

17, 11, 7, 14, 12, 22

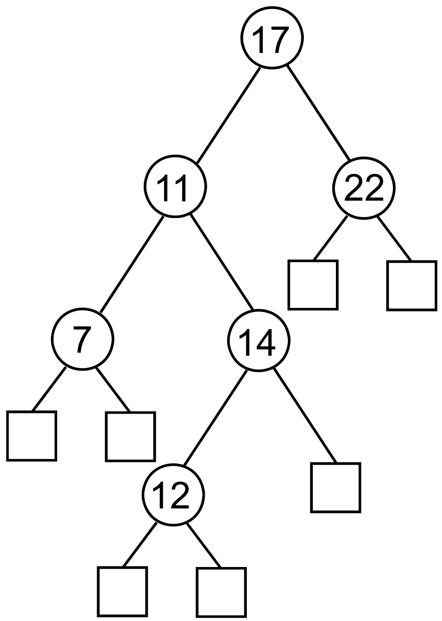
Postorder:

7, 12, 14, 11, 22, 17

Inorder:

7, 11, 12, 14, 17, 22

Example for Search, Insertion, Deletion



Sorting with standard search trees

Idea: Create a search tree for the input sequence and output the keys by an inorder traversal.

Remark: Depending on the input sequence, the search tree may degenerate.

Complexity: Depends on internal path length

Worst case: Sorted input: $\Rightarrow \Omega(n^2)$ steps.

Best case: We get a complete search tree of minimal height of about log *n*. Then *n* insertions and outputs are possible in time $O(n \log n)$.

Average case: Later ...

Analysis of search trees

Two possible approaches to determine the internal path length:

1. Random tree analysis, i.e. average over all possible permutations of keys to be inserted (into the initially empty tree).

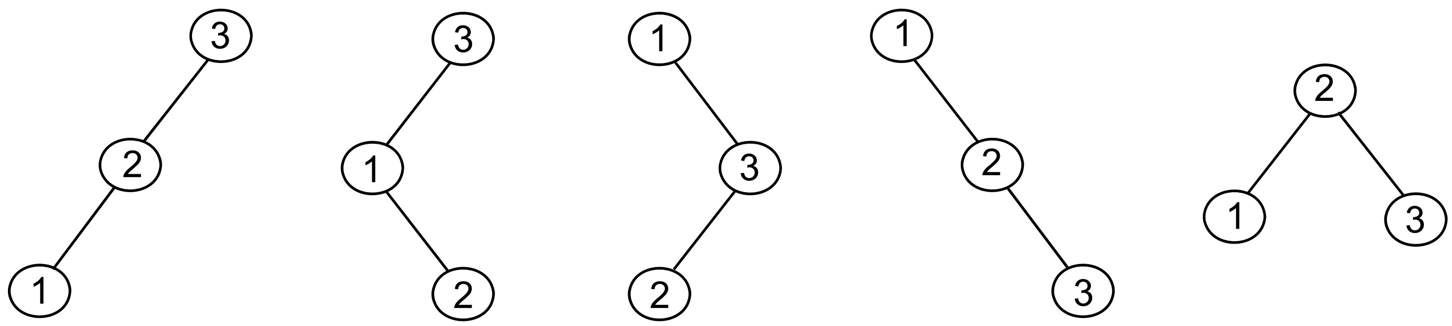
2. Shape analysis, i.e. average over all structurally different trees with n keys .

Difference of the expected values for the internal path:

1. $\approx 1.386\, n \log_2 n - 1.846 \cdot n + O(\log n)$

2. $\approx n \cdot \sqrt{\pi n} + O(n)$

Reason for the difference



3,2,1 3,1,2 1,3,2 1,2,3 2,1,3 and 2,3,1

$\longrightarrow$ Random tree analysis counts more balanced trees more often.

Internal path length

Internal path length I: measure for judging the quality of a search tree t.

Recursive definition:

1. If t is empty, then $I(t) = 0.$

2. For a tree t with left subtree t_l and right subtree t_r :

   $I(t) = I(t_l) + I(t_r)$ + # inner nodes of t

Apparently:

$$I(t) = \sum_p (depth(p) + 1)$$

$$p \text{ internal node in } t$$

Average search path length

For a tree *t* the average search path length is defined by:

$$D(t) \;:= I(t)/n, n \;=\; \#\ \text{nodes in}\ t.$$

Question: What is the size of *D*(*t*) in the

- best
- worst
- average

case for a tree *t* with *n* internal nodes?

Internal path: best case

We obtain a complete binary tree

Internal path: worst case

Random trees

- Without loss of generality, let $\{1,\ldots,n\}$ be the keys to be inserted.
- Let $s_1,\ldots, s_n$ be a random permutation of these keys.
- Hence, the probability that s_1 has the value k, $P(s_1=k) = 1/n$.
- If k is the first key, k will be stored in the root.
- Then the left subtree contains k-1 elements (the keys 1, …, k-1) and the right subtree contains n-k elements (the keys k+1, …,n).

Expected internal path length

- *EI*(*n*) : Expectation for the internal path length of a randomly generated binary search tree with *n* nodes
- Apparently we have:

$$\begin{aligned}
EI(0) &= 0 \\
EI(1) &= 1 \\
EI(n) &= \frac{1}{n}\sum_{k=1}^{n} (EI(j-1) \;+\; EI(n-k) \;+\; n) \\
&= n + \frac{1}{n}\left(\sum_{k=1}^{n} EI(k-1) \;+\; \sum_{k=1}^{n} EI(n-k)\right)
\end{aligned}$$

- Assume: $EI(n) = 1.386n \log_2 n - 1.846n + O(\log n)$.

Proof (1)

$$EI(n+1) = (n+1) + \frac{2}{n+1} * \sum_{k=0}^{n} (EI(k)$$

and hence

$$(n+1) * EI(n+1) = (n+1)^2 + 2 * \sum_{k=0}^{n} (EI(k)$$

$$n * EI = n^2 + 2 * \sum_{k=0}^{n-1} (EI(k)$$

From the last two equations it follows that

$$(n+1)\ EI(n+1) - n * EI(n) = 2n + 1 + 2 * RI(n)$$

$$(n+1)\ EI(n+1) = (n+2)\ EI(n) + 2n + 1$$

$$EI(n+1) = \frac{2n+1}{n+1} + \frac{n+2}{n+1} EI(n).$$

Proof (2)

By induction over *n* it is possible to show that for all n ≥ 1:

$$EI(n) = 2(n+1)H_n - 3n$$

$H_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$ is the *n*-th harmonic number,

which can be estimated as follows:

$$H_n = ln\ n + \gamma + \frac{1}{2n} + O\left(\frac{1}{n^2}\right)$$

where $\gamma = 0.5772\ldots$ the so-called Euler constant.

Proof (3)

Thus,

$$EI(n) = 2n\,ln\,n - (3 - 2\gamma) * n + ln\,n + 1 + 2\gamma + O\left(\frac{1}{n}\right)$$

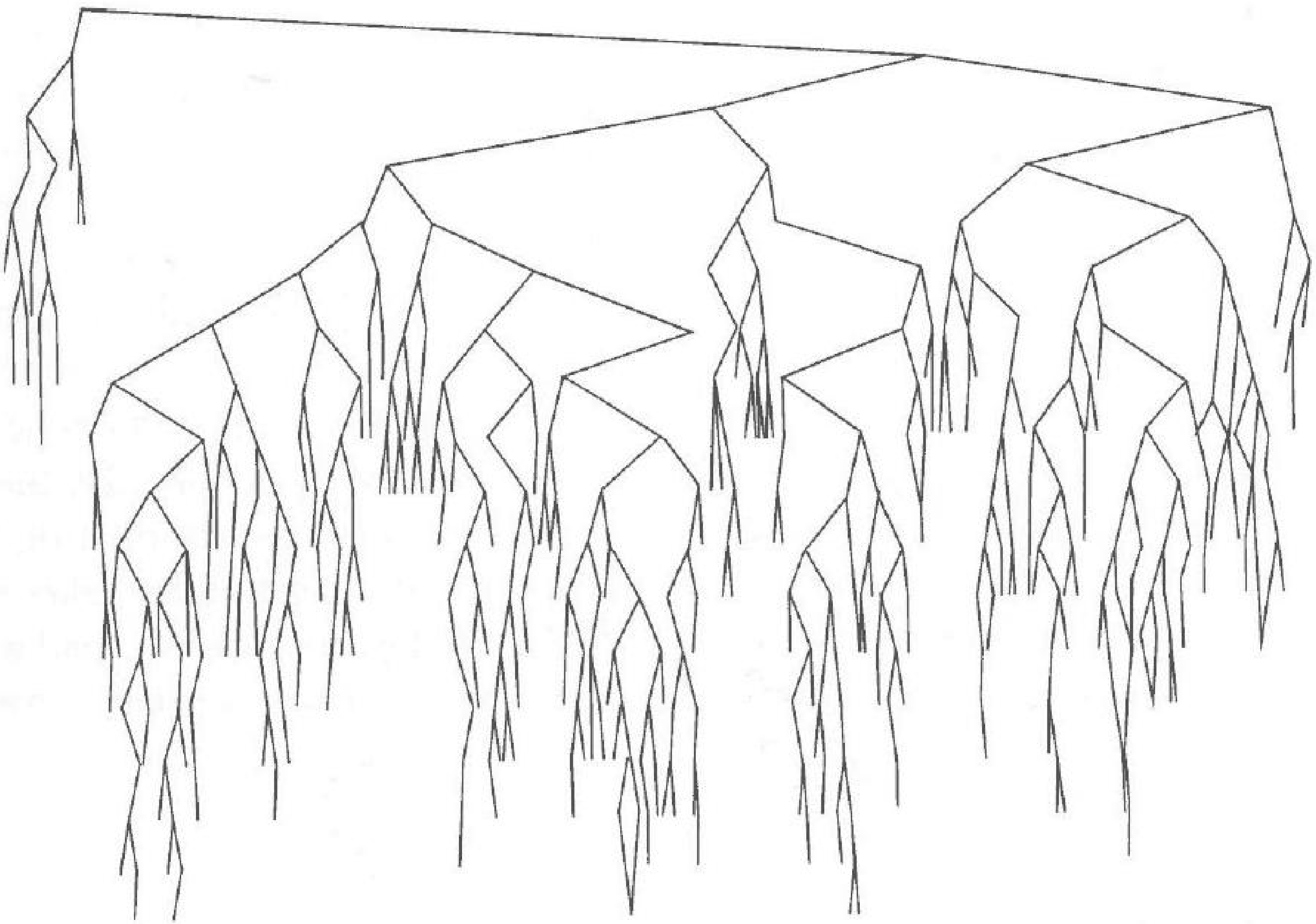
and hence,

$$\begin{aligned}
\frac{EI(n)}{n} &= 2\ ln\ n - (3 - 2\gamma) + \frac{2\ ln\ n}{n} + \ldots \\
&= \frac{2}{log_2\ e} * log_2\ n - (3 - 2\gamma) + \frac{2\ ln\ n}{n} + \ldots \\
&= \frac{2\ log_{10}\ 2}{log_1 0\ e} * log_2\ n - (3 - 2\gamma) + \frac{2\ ln\ n}{n} + \ldots \\
&= \frac{2}{log_2\ e} * log_2\ n - (3 - 2\gamma) + \frac{2\ ln\ n}{n} + \ldots \\
&\approx 1.386\ log_2\ n - (3 - 2\gamma) + \frac{2\ ln\ n}{n} + \ldots
\end{aligned}$$

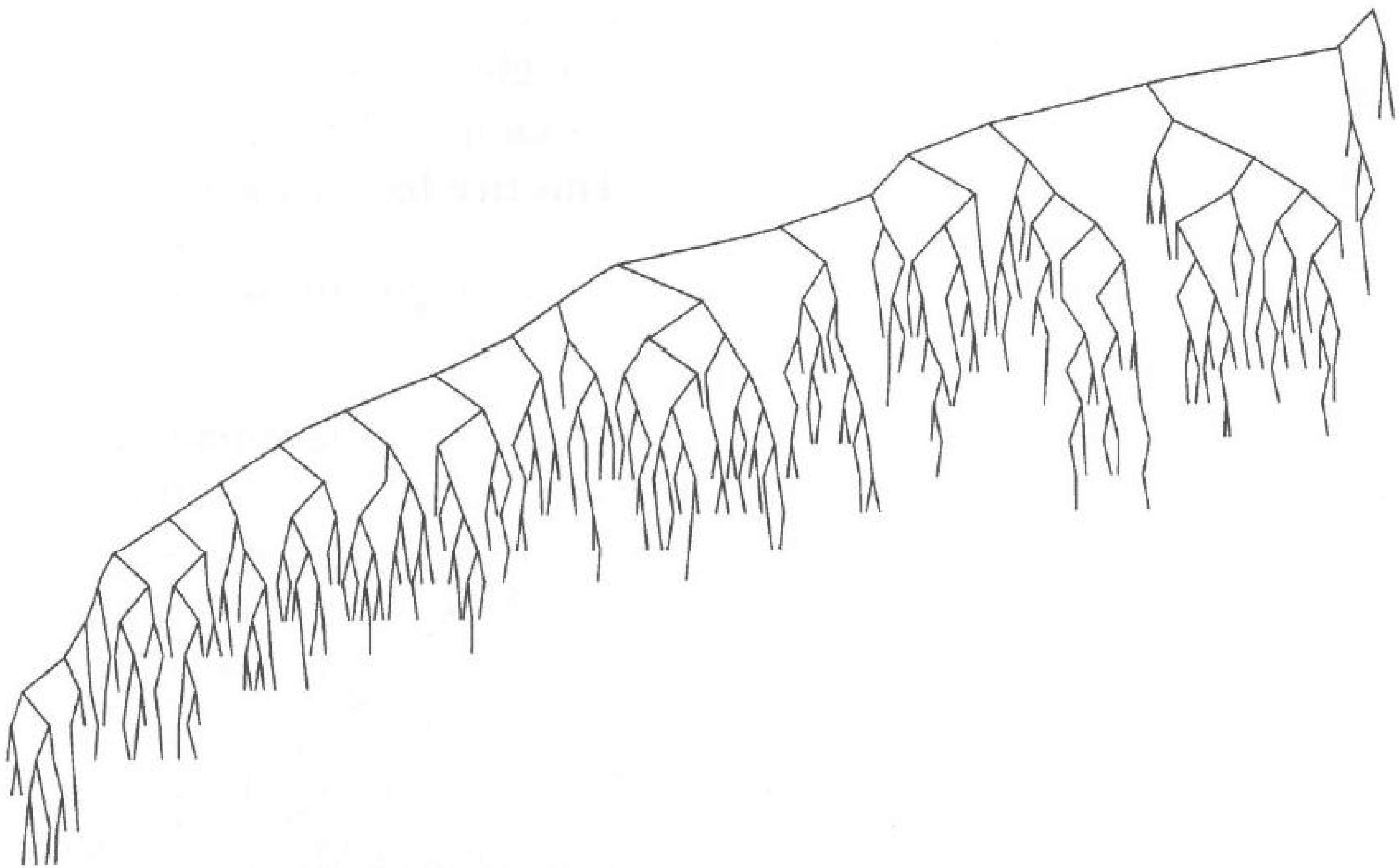
Observation

- Search, insertion and deletion of a key in a randomly generated binary search tree with n keys can be done, on average, in $O(\log_2 n)$ steps.
- In the worst case, the complexity can be $\Omega(n)$.
- One can show that the average distance of a node from the root in a randomly generated tree is only about 40% above the optimal value.
- However, by the restriction to the symmetrical successor, the behaviour becomes worse.
- If n^2 update operations are carried out in a randomly generated search tree with n keys, the expected average search path is only $\Theta(\sqrt{n})$.

Typical binary tree for a random sequence of keys



Resulting binary tree after n^2 updates



Structural analysis of binary trees

Question: What is the average search path length of a binary tree with *N* internal nodes if the average is made over all structurally different binary trees with *N* internal nodes?

Answer: Let

I_N = total internal path length of all structurally different binary trees with *N* internal nodes

B_N = number of all structurally different trees with *N* internal nodes

Then I_N/B_N =

Number of structurally different binary trees

Total internal path length of all trees with N nodes

- For each tree t with left subtree t_l and right subtree t_r :

Summary

The average search path length in a tree with *N* internal nodes (averaged over all structurally different trees with *N* internal nodes) is:

$1/N \cdot I_N/B_N$