ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

Algorithm Theory

8 Treaps

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

The Dictionary Problem

Given: Universe $(U,<)$ of keys with a total order

Goal: Maintain a set $S \subseteq U$ under the following operations

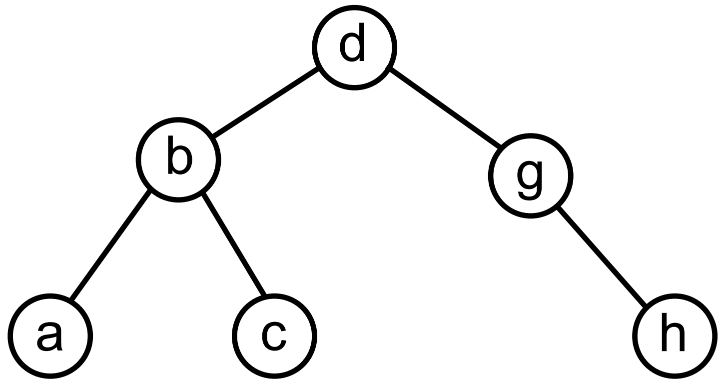
- **Search(x,S):** Is $x \in S$?
- **Insert(x,S):** Insert x into S if not already in S.
- **Delete(x,S):** Delete x from S

Extended Set of Operations

- Minimum(S): Return smallest key
- Maximum(S): Return largest key
- List(S): Output elements of S in increasing order by key
- Union(S_1,S_2): Merge S_1 and S_2.
  Condition: $\forall\, x_1 \in S_1 \,,\, x_2 \in S_2$: $x_1 < x_2$
- Split(S,x,S_1,S_2): Split S into S_1 and S_2
  $\forall\, x_1 \in S_1 \,,\, x_2 \in S_2$: $x_1 \leq x$ and $x_2 > x$

Known Solutions

- **Binary search trees**



  Disadvantage: Sequence of insertions may lead to a linear list a, b, c, d, e, f

- **Height balanced trees:** AVL-trees, (a,b)-trees

  Disadvantage: complex algorithms or high memory

Approach for Randomized Search Trees

If n elements are inserted in random order into a binary search tree, the expected depth is 1.39 log n.

Idea: Each element x is assigned a priority chosen uniformly at random

$$\text{prio}(x) \in R$$

The goal is to establish the following property

(*) The search tree has the structure that would result if elements were inserted in the order of their priorities

Treaps (Tree + Heap)

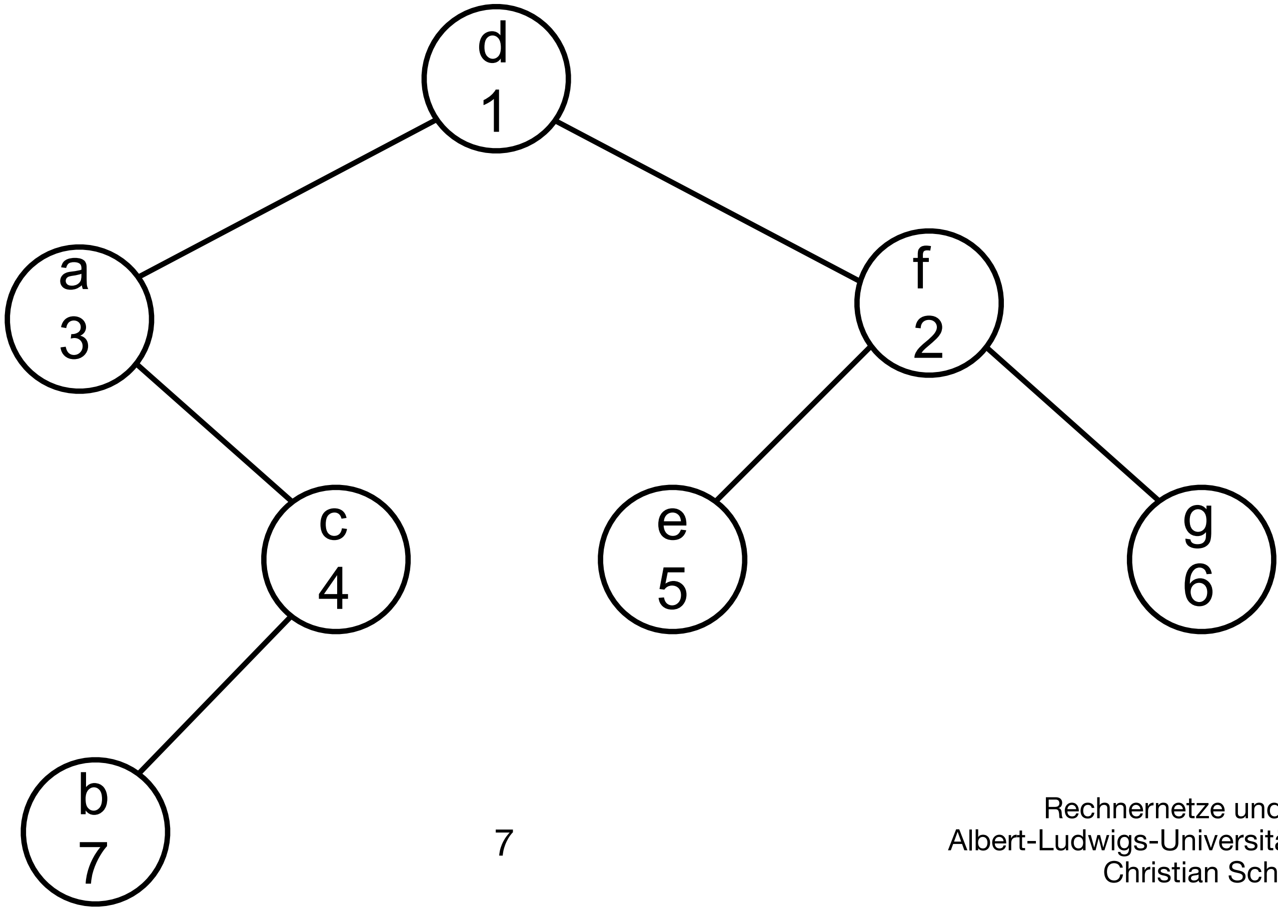
Definition: A Treap is a binary tree.
Each node contains one element x with key(x) $\in U$ and prio(x) $\in R$.
The following properties hold.

- **Search tree property**
  For each element x:
  - elements y in the left subtree x satisfy: key(y) < key(x)
  - elements y in the right subtree x satisfy: key(y) > key(x)

- **Heap property**
  For all elements x,y:
  if y is a child of x then prio(y) > prio(x).
  All priorities are pairwise distinct.

Example

Key	a	b	c	d	e	f	g
Priority	3	7	4	1	5	2	6



Uniqueness of Treaps

Lemma: For elements x_1, ... , x_n with key(x_i) and prio(x_i), there exists a unique treap. It satisfies property (*).

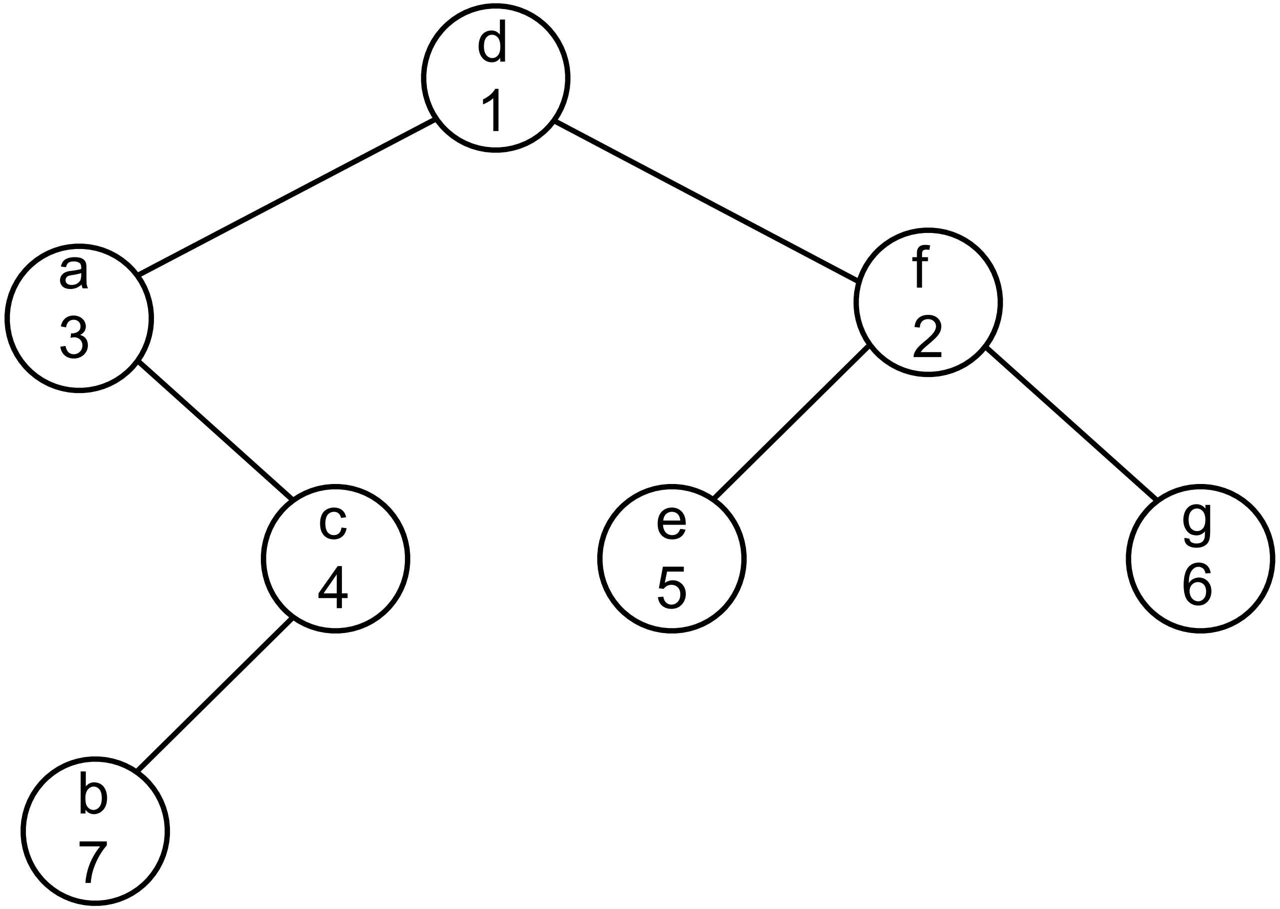
Proof:

n=1: ok

n>1: at the root there is only one choice: key with minimal prio(x_r)

the left tree and the right tree is determined by induction assumption

Search for an Element



Search for Element with Key *k*

```
v := root;
while v ≠ nil do
   case key(v) =  k : stop;  "element found" (successful search)
        key(v) <  k : v:= RightChild(v);
        key(v) >  k : v:= LeftChild(v);
   endcase;
endwhile;
 "element not found" (failed search)
```

Runtime: O(# elements on the search path)

Analysis of the Search Path

Elements $x_1, \ldots , x_n$ $\quad$ x_i has i-th smallest key

Let M be a subset of the elements.

$P_{min}(M)$ = element in M with lowest priority

Lemma:

a) Sei $i<m$. $\quad$ x_i is ancestor of x_m iff $P_{min}(\{x_i,\ldots,x_m\}) = x_i$

b) Sei $m<i$. $\quad$ x_i is ancestor of x_m iff $P_{min}(\{x_m,\ldots,x_i\}) = x_i$

Analysis of the Search Path

Proof: a) Use (*). Elements are inserted in order of increasing priorities.

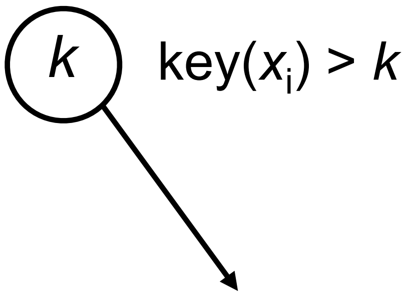
"$\Leftarrow$" $P_{min}(\{x_i,\ldots,x_m\}) = x_i \quad \Rightarrow \quad x_i$ is inserted first among $\{x_i,\ldots,x_m\}$.

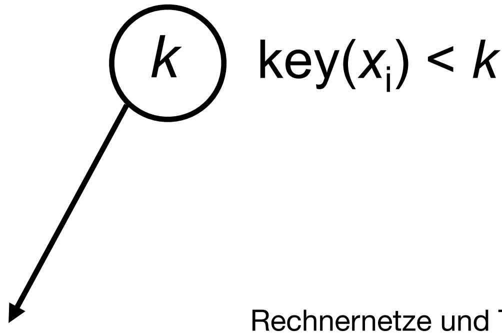
When x_i is inserted, the tree contains only keys k with

$k < \text{key}(x_i)$ or $k > \text{key}(x_m)$

When elements $x_j \in \{x_i,\ldots,x_m\}$ is inserted it traverses the same path as x_i.

$\Rightarrow x_i$ is an ancester of $x_j \Rightarrow x_i$ is ancestor of x_m





Analysis of the Search Path

Beweis: a) (Let $i<m$. x_i is ancestor of x_m iff $P_{min}(\{x_i,\ldots,x_m\}) = x_i$)

"$\Rightarrow$" Let $x_j = P_{min}(\{x_i,\ldots,x_m\})$. To prove $x_i = x_j$

Assume: $x_i \neq x_j$

Use (*) and consider the search path when x_j is inserted.

As before we can show that any $x_k \in \{x_i,\ldots,x_m\}$ traverses the same search path as x_j

$\Rightarrow$ x_j is ancestor of x_k

Case 1: $x_j = x_m$ $\Rightarrow$ x_m is ancestor of x_i Contradiction!

Case 2: $x_j \neq x_m$ key(x_i) < key(x_j) $\Rightarrow$ x_i in left subtree

key(x_i) > key(x_j) $\Rightarrow$ x_k in right subtree

Part b) follows analogously.

Analysis of the Search Operation

Let T be a treap with elements $x_1, \ldots, x_n$ x_i has i-th smallest key

n-th Harmonic number

$$H_n = \sum_{k=1}^{n} 1/k$$

Lemma:

1. **Successful Search:** The expected number of nodes x_m on the path is $H_m + H_{n-m+1} - 1$.

2. Failed Search: Let m be the number of keys being smaller than the search key k. The expected number of nodes on the search path is $H_m + H_{n-m}$.

Analysis of the Search Operation

Beweis: Part 1

$$X_{m,i} = \begin{cases} 1 & x_i \text{ is ancestor of } x_m \\ 0 & \text{else} \end{cases}$$

X_m = # nodes on the path from the root to x_m (incl. x_m)

$$X_m = 1 + \sum_{i<m} X_{m,i} + \sum_{i>m} X_{m,i}$$

$$E[X_m] = 1 + E\left[\sum_{i<m} X_{m,i}\right] + E\left[\sum_{i>m} X_{m,i}\right]$$

Analysis of the Search Operation

i < *m* :

$$E[X_{m,i}] = \text{Prob}[x_i \text{ is ancestor of } x_m] = 1/(m - i + 1)$$

All elements of $\{x_i, \ldots, x_m\}$ have the same probability of being the one with the smallest priority

$\text{Prob}[P_{min}(\{x_i,\ldots,x_m\}) = x_i] = 1/(m-i+1)$

i > *m* :

$$E[X_{m,i}] = 1/(i - m + 1)$$

Analysis of the Search Operation

$$E[X_m] = 1 + \sum_{i<m} \frac{1}{m-i+1} + \sum_{i>m} \frac{1}{i-m+1}$$

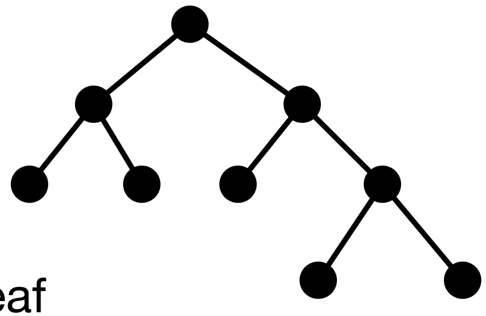
$$= 1 + \frac{1}{m} + \ldots + \frac{1}{2} + \frac{1}{2} + \ldots + \frac{1}{n-m+1}$$

$$= H_m + H_{n-m+1} - 1$$

Teil 2 analog

Inserting a New Element *x*

1. Choose prio(x).
2. Search for the position of *x* in the tree.



3. Insert *x* as a leaf
4. Restore the heap property.

while prio(parent(*x*)) > prio(*x*) **do**

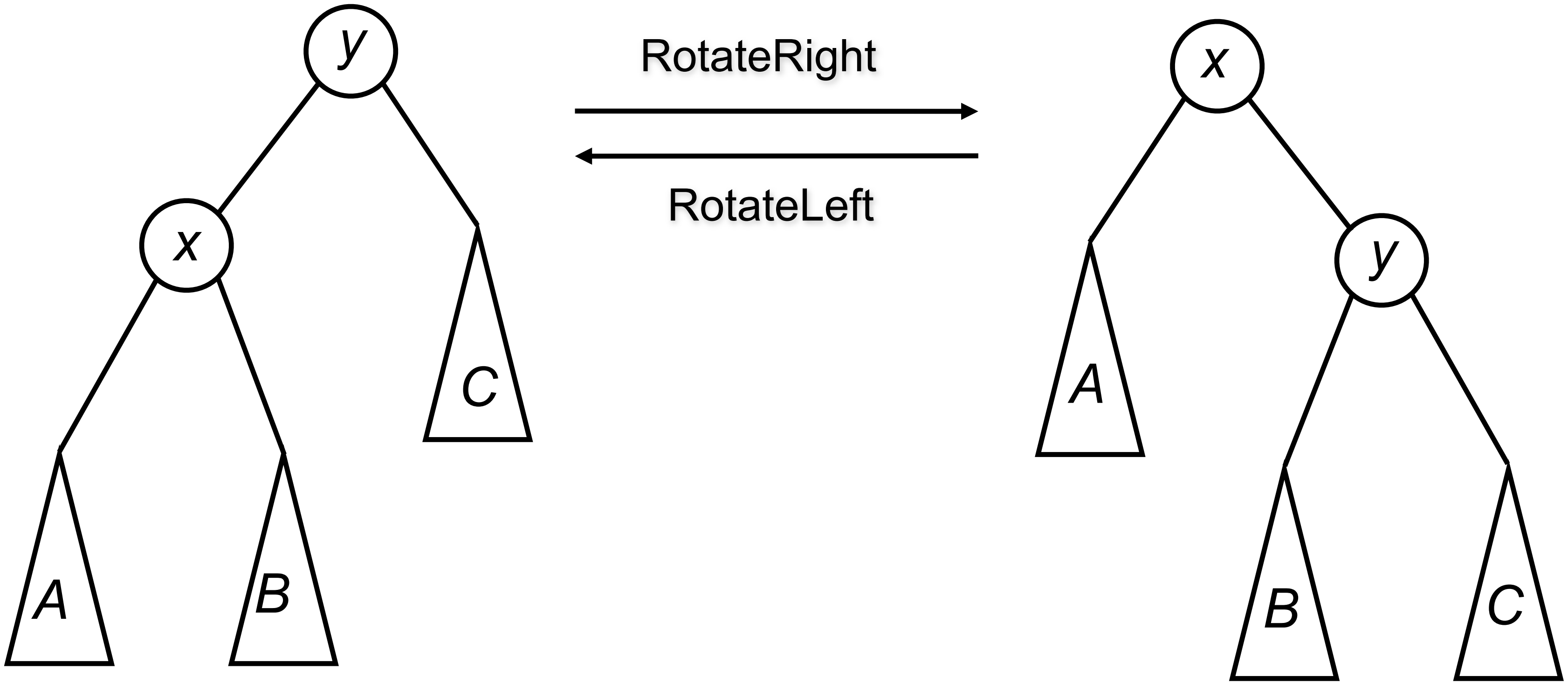
if *x* is left child **then** RotateRight(parent(*x*))

else RotateLeft(parent(*x*));

endif

endwhile;

Rotations



The rotations maintain the search tree property and restore the heap property.

Deleting an Element *x*

1. Find *x* in the tree.
2. **while** *x* is not a leaf **do**

    u := child with smaller priority;

    if *u* is left child **then** RotateRight(*x*))

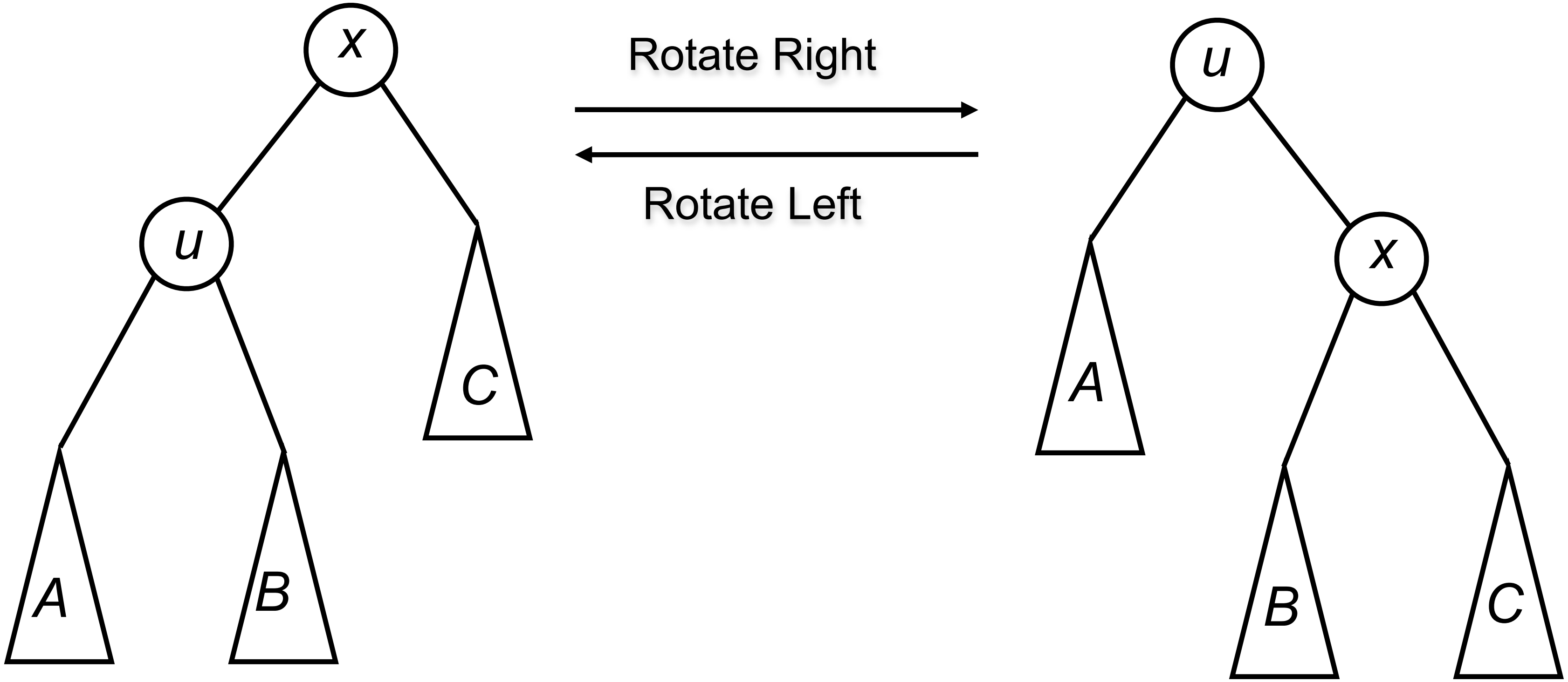
    else RotateLeft(*x*);

    endif;

   endwhile;

3. Delete x;

Rotations



Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg
Christian Schindelhauer

Analysis of Insert and Delete Operations

Lemma: The expected running time of insert and delete operations is O(log n). The expected number of rotations is 2.

Proof: Analysis of insert (delete is the inverse operation)

#rotations = depth of x after being inserted as a leaf (1)

- depth of x after rotations (2)

Let $x = x_m$

(2) Expected depth is $H_m + H_{n-m+1} - 1$

(1) Expected detph is $H_{m-1} + H_{n-m} + 1$

The tree contains $n-1$ elements, $m-1$ of them are smaller.

#rotations = $H_{m-1} + H_{n-m} + 1 - (H_m + H_{n-m+1} - 1) < 2$

Extended Set of Operations

n = number elements in treap T

- Minimum(S): Return smallest key O(log n)
- Maximum(S): Return largest key O(log n)
- List(S): Output elements of S in increasing order by key O(n)
- Union(S_1,S_2): Merge S_1 and S_2.
  Condition: $\forall x_1 \in S_1, x_2 \in S_2$: $x_1 < x_2$
- Split(S,x,S_1,S_2): Split S into S_1 and S_2
  $\forall x_1 \in S_1, x_2 \in S_2$: $x_1 \leq x$ and $x_2 > x$

The Split Operation

Split(T,k,T_1,T_2): Split T into T_1 and T_2.

$\forall x_1 \in T_1, x_2 \in T_2$: key($x_1$) $\leq k$ und key(x_2) $> k$

W.l.o.g key k is not in T.

Otherwise delete the element with key k and re-insert it into T_1 after the split operation.

1. Generate a new element x with key(x)=k and prio(x) = $-\propto$.
2. Insert x into T.
3. Delet the new root. The left subtree is T_1, the right subtree is T_2.

The Union Operation

Union(T_1,T_2): Merge T_1 and T_2.

Condition: $\forall\ x_1 \in T_1\ ,\ x_2 \in T_2$: key($x_1$) < key($x_2$)

1. Determine key k with key(x_1) < k < key(x_2)

   for all $x_1 \in T_1$ and $x_2 \in T_2$.

2. Generate element x with key(x)=k and prio(x) = $-\infty$.
3. Generate treap T with root x, left subtree T_1 and

   right subtree T_2.

4. Delete x from T.

Analysis

Lemma: The expected running time of Union and Split is O(log n).

Implementation

Priorities from [0,1)

Priorities are used only when two elements are compared to find out which of them has the higher priority

In case of equality, extend both priorities by bits chosen uniformly at random until two bits differ.

$p_1 = 0{,}010111001$	$p_1 = 0{,}010111001011$
$p_2 = 0{,}010111001$	$p_2 = 0{,}010111001010$

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

8 Treaps

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

IIF
INSTITUT FÜR INFORMATIK FREIBURG