Algorithms Theory

04 - Treaps

Data structure
Tree Heap
Dictionary problem
Randomization

Dr. Alexander Souza

The dictionary problem

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

Goal: Maintain 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.

element x = key $\in U$, satellite data

Example: Employee Database

Personal # — key

Salary

Private address

Room #

Boss

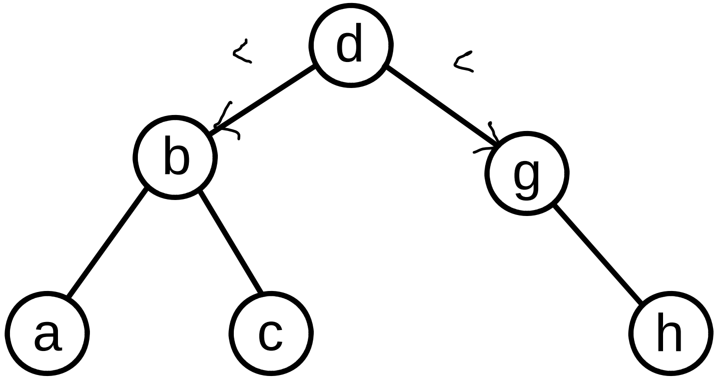
— Satellite data

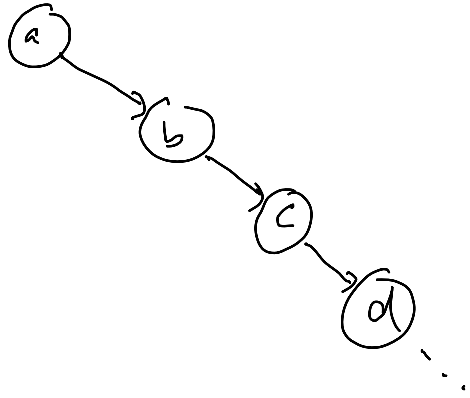
Extended set of operations

- Minimum(S): Return smallest key.
- Maximum(S): Return largest key.
- List(S): Output elements of S in increasing order of 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 $x < x_2$

Known solutions

- **Binary search trees**





Drawback: Sequence of insertions may lead to
a linear list a, b, c, d, e, f   Problem: Search has linear running time

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

Drawback: Complex algorithms or high memory requirements.

AVL: height(LST) − height(RST) ≤ 1 complex
(a,b) − tree: $a \leq$ # children $\leq b$ space overhead } $O(\log n)$

Goal: simple + efficient datastructure

Approach for randomized search trees

Fact: 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

prio(x) $\in$ R

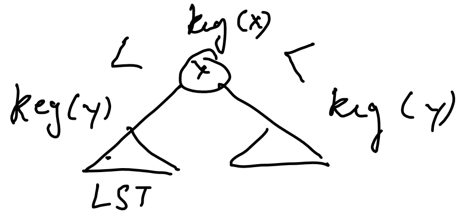
The goal is to establish the following property.

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

Not obvious

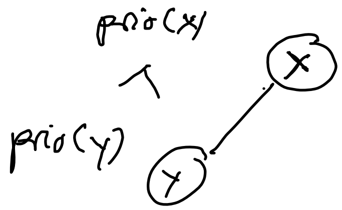
Treaps (Tree + Heap)

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 of x satisfy: key(y) < key(x)
  - elements y in the right subtree of x satisfy : key(y) > key(x)

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

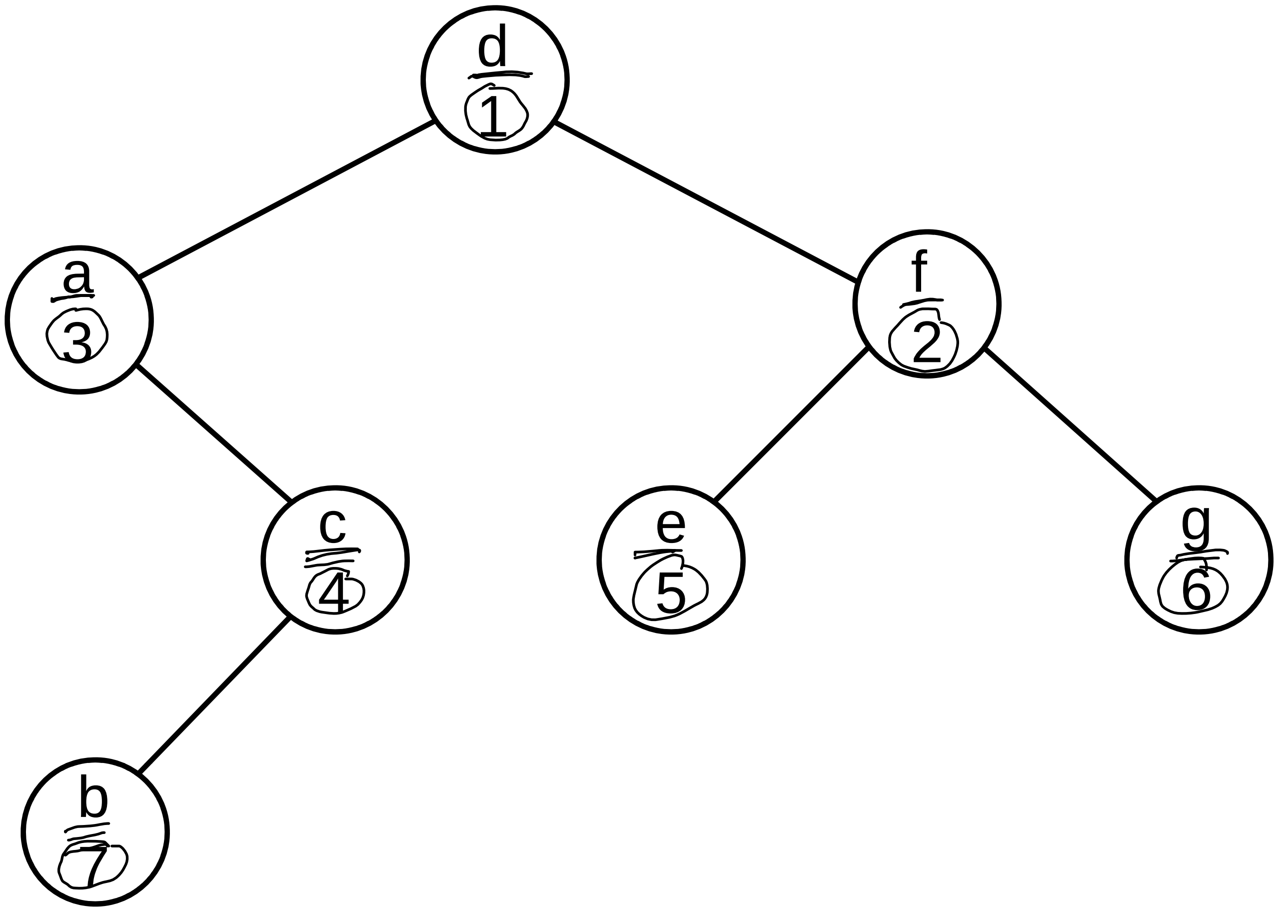


Do Treaps exist?

Example

key	a	b	c	d	e	f	g
priority	3	7	4	1	5	2	6

Search tree property ✓
Heap property ✓

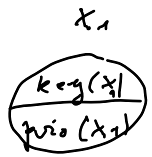


Treap uniqueness

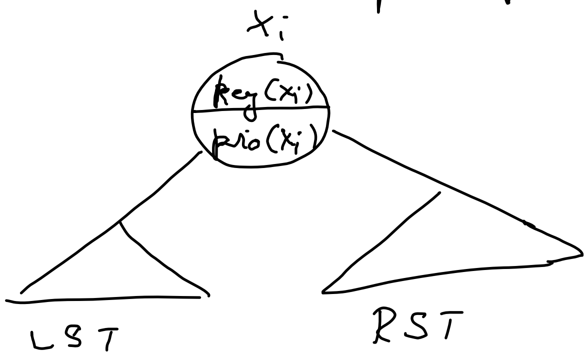
Lemma: For elements $x_1, \ldots, x_n$ with key(x_i) and prio(x_i), there exists a unique treap. It satisfies property (*).

Proof: Induction

n=1: ok



n>1: Element x_i with smallest priority is in the root.

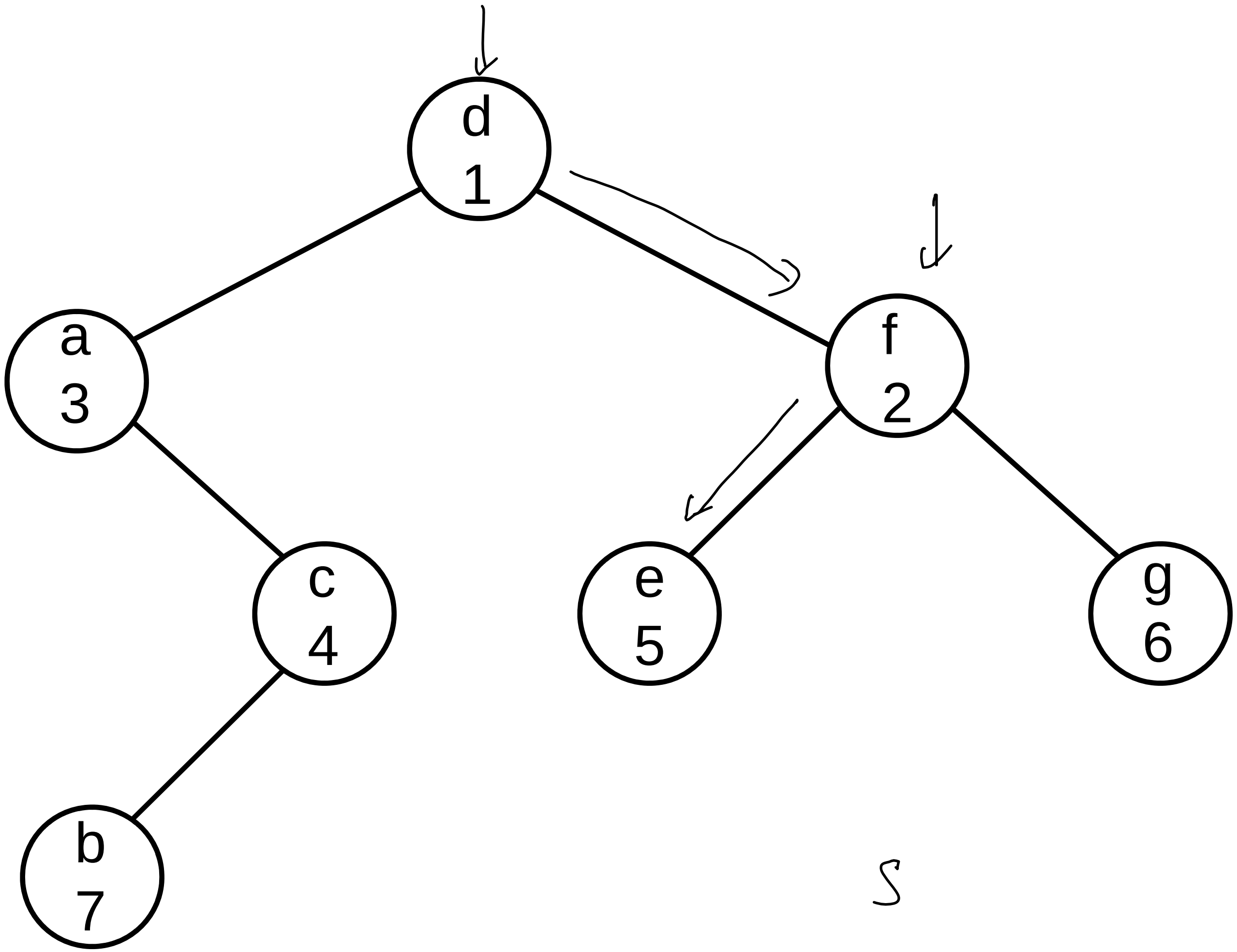


LST: x_j key(x_j) < key(x_i), prio(x_j) > prio(x_i)

RST: x_j key(x_j) > key(x_i), prio(x_j) > prio(x_i)

Search for an element

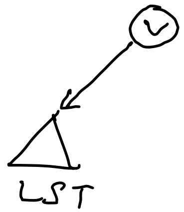
Search (e, S)

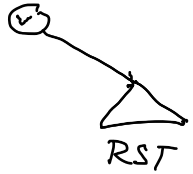


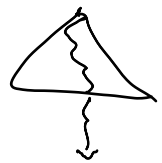
Search for element with key *k*

v current vertex

```
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" (unsuccessful search)
```







Running time: O(# elements on the search path)