DS-210: PROGRAMMING FOR DATA SCIENCE

LECTURE 34

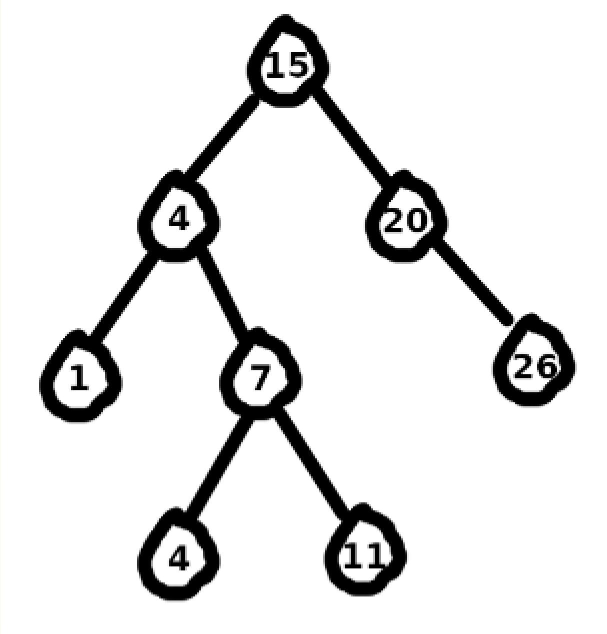
1. BINARY SEARCH TREES

2. APPLICATIONS (RANGE SEARCHING)

3. RUST: `BTreeMap` AND `BTreeSet`

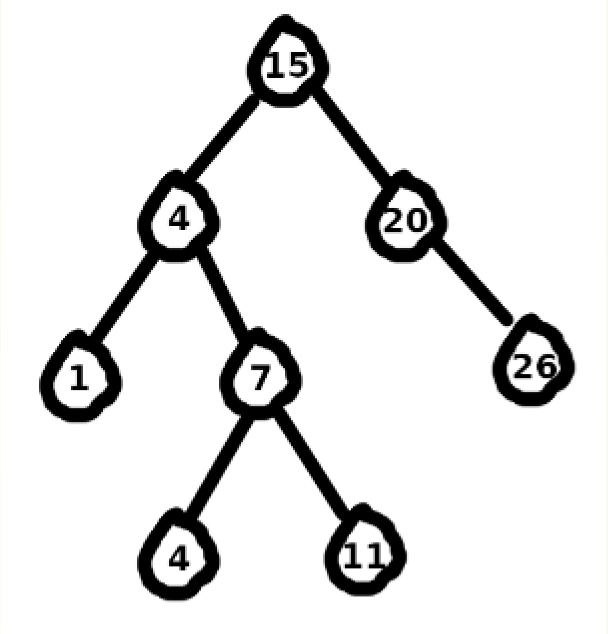
BINARY SEARCH TREES

- Organize data into a binary tree
  - Similar to binary heaps

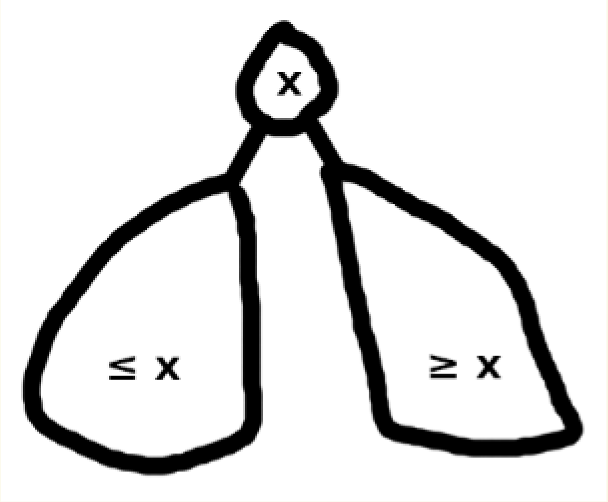


BINARY SEARCH TREES

- Organize data into a binary tree
  - Similar to binary heaps

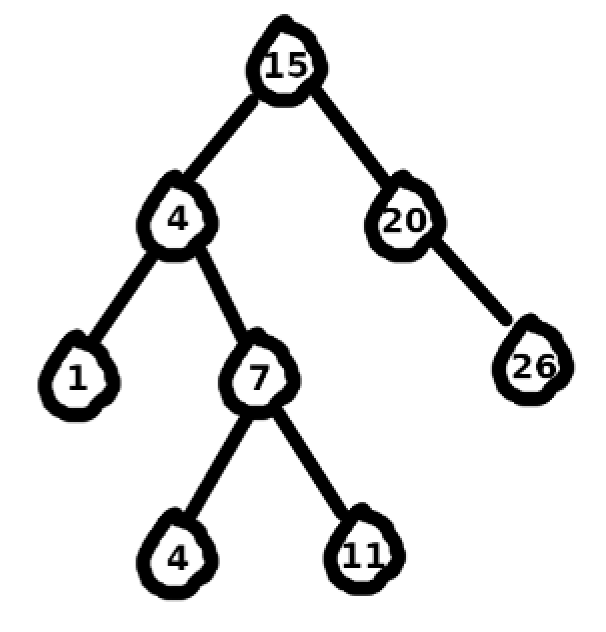


- Invariant at each node:
  - all left descendants $\leq$ parent
  - parent $\leq$ all right descendants

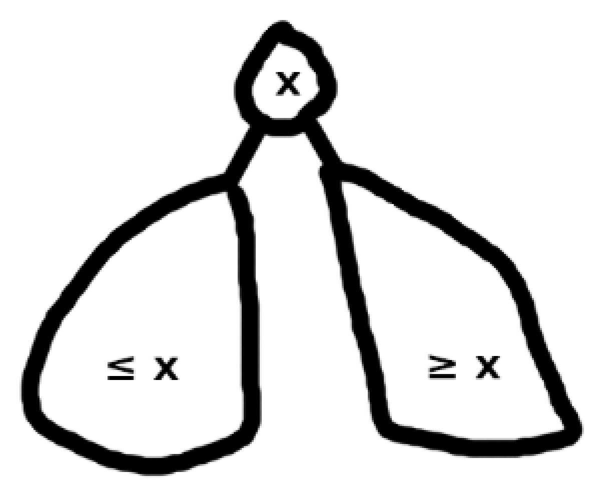


BINARY SEARCH TREES

- Organize data into a binary tree
  - Similar to binary heaps



- Invariant at each node:
  - all left descendants $\leq$ parent
  - parent $\leq$ all right descendants



- Compared to binary heaps:
  - different ordering of elements

BASIC OPERATIONS: FIND A KEY k

How can we do this?

BASIC OPERATIONS: FIND A KEY k

How can we do this?

- Descend recursively from the root until k found or stuck:
  - If $k <$ value at the current node, go left
  - If $k >$ value at the current node, go right

[see examples on the board]

BASIC OPERATIONS: INSERT A KEY k

How can we do this?

BASIC OPERATIONS: INSERT A KEY k

How can we do this?

- Keep descending from the root until you leave the tree
  - If $k \leq$ value at the current node, go left
  - If $k >$ value at the current node, go right
- Create a new node containing k there

[see examples on the board]

BASIC OPERATIONS: DELETE A NODE

How can we do this?

BASIC OPERATIONS: DELETE A NODE

How can we do this?

- More complicated: need to find a replacement
- If the node is a leaf: nothing to do
- If only one child: move the child up
- Otherwise:
    - find the **rightmost** descendant in the **left** subtree
    - it will have at most one child

[see examples on the board]

COST OF THESE OPERATIONS?

COST OF THESE OPERATIONS?

$$O(\textbf{depth of the tree})$$

COST OF THESE OPERATIONS?

$$O(\textbf{depth of the tree})$$

Bad news: the depth can be made proportional to n, the number of nodes

COST OF THESE OPERATIONS?

$$O(\textbf{depth of the tree})$$

Bad news: the depth can be made proportional to n, the number of nodes

Good news: smart ways to make the depth $O(\log n)$

BALANCED BINARY SEARCH TREES

There are smart ways to rebalance the tree!

- Depth: $O(\log n)$
- Usually additional information has to be kept at each node
- Popular examples:
  - Red–black trees
  - AVL trees
  - ...

WHY USE BINARY SEARCH TREES?

- Hash maps and hash sets give us $O(1)$ time operations?

WHY USE BINARY SEARCH TREES?

- Hash maps and hash sets give us $O(1)$ time operations?

REASON 1:

- Good worst case behavior: no need for a good hash function

WHY USE BINARY SEARCH TREES?

- Hash maps and hash sets give us $O(1)$ time operations?

REASON 1:

- Good worst case behavior: no need for a good hash function

REASON 2:

- Can answer efficiently questions such as:
    - What is the smallest/greatest element?
    - What is the smallest element greater than x?
    - List all elements between x and y

EXAMPLE: FIND THE SMALLEST ELEMENT GREATER THAN x

EXAMPLE: FIND THE SMALLEST ELEMENT GREATER THAN x

Question: How can you list all elements in order in $O(n)$ time?

EXAMPLE: FIND THE SMALLEST ELEMENT GREATER THAN x

Question: How can you list all elements in order in $O(n)$ time?

Answer: recursively starting from the root

- visit left subtree
- output current node
- visit right subtree

EXAMPLE: FIND THE SMALLEST ELEMENT GREATER THAN x

Question: How can you list all elements in order in $O(n)$ time?

Answer: recursively starting from the root

- visit left subtree
- output current node
- visit right subtree

Outputting smallest element greater than x:

- Like above, ignoring whole subtrees smaller than x
- Will get the first element greater than x in $O(\log n)$ time

EXAMPLE: FIND THE SMALLEST ELEMENT GREATER THAN x

Question: How can you list all elements in order in $O(n)$ time?

Answer: recursively starting from the root

- visit left subtree
- output current node
- visit right subtree

Outputting smallest element greater than x:

- Like above, ignoring whole subtrees smaller than x
- Will get the first element greater than x in $O(\log n)$ time

For balanced trees: listing t first greater elements takes $O(t + \log n)$ time

BINARY SEARCH TREES IN RUST'S STANDARD LIBRARY?

- Not exactly
- For efficiency reasons, B-trees:
  - generalization of binary trees
  - between B and $2B$ keys in a node
  - corresponding number of subtrees

BINARY SEARCH TREES IN RUST'S STANDARD LIBRARY?

- Not exactly
- For efficiency reasons, B-trees:
  - generalization of binary trees
  - between B and $2B$ keys in a node
  - corresponding number of subtrees

Where can you meet B-trees

- Traditionally, very popular in databases
- Interesting that now considered more efficient for in memory operations

`std::collections::BTreeSet` AND `...::BTreeMap`

Sets and maps, respectively

std::collections::BTreeSet AND ...::BTreeMap

Sets and maps, respectively

In [2]:
```
// let's create a set
use std::collections::BTreeSet;
let mut set: BTreeSet<i32> = BTreeSet::new();
set.insert(5);
set.insert(7);
set.insert(11);
set.insert(23);
set.insert(25);
```

`std::collections::BTreeSet` AND `...::BTreeMap`

Sets and maps, respectively

In [2]:
```
// let's create a set
use std::collections::BTreeSet;
let mut set: BTreeSet<i32> = BTreeSet::new();
set.insert(5);
set.insert(7);
set.insert(11);
set.insert(23);
set.insert(25);
```

In [3]:
```
// listing a range
set.range(7..24).for_each(|x| println!("{}", x));
```

```
7
11
23
```

std::collections::BTreeSet AND ...::BTreeMap

Sets and maps, respectively

In [2]:
```
// let's create a set
use std::collections::BTreeSet;
let mut set: BTreeSet<i32> = BTreeSet::new();
set.insert(5);
set.insert(7);
set.insert(11);
set.insert(23);
set.insert(25);
```

In [3]:
```
// listing a range
set.range(7..24).for_each(|x| println!("{}", x));
```

```
7
11
23
```

In [4]:
```
// listing a range: another way of specifying it
use std::ops::Bound::{Included,Excluded};
set.range((Excluded(5),Included(11))).for_each(|x| println!("{}", x));
```

```
7
11
```

?