CSCI 136: Data Structures and Advanced Programming

Lecture 26

Trees, part 4

Instructor: Dan Barowy

Williams

Topics

More BST methods

Tree balance

Big-O

Implicit BST

Announcements

Spring pre-registration begins Wed, April 27 and runs until Fri, May 6.

The best way to get into the CS course you want is to **pre-register**.

Common "next steps" after CSCI 136:

CSCI 237: Computer Organization
CSCI 256: Algorithms
CSCI 334: Principles of Programming Languages

also, some electives.

Practice Quiz

Binary Search Tree

Let’s add **find** and **contains**.

Should it be a **structure**?

structure5

Interface Structure<E>

All Superinterfaces:
java.lang.Iterable<E>

All Known Subinterfaces:
Graph<V,E>, Linear<E>, List<E>, OrderedStructure<K>, Queue<E>, Set<E>, Stack<E>

Method Summary	
void	`add(E value)` Inserts value in some structure-specific location.
void	`clear()` Removes all elements from the structure.
boolean	`contains(E value)` Determines if the structure contains a value.
java.util.Enumeration	`elements()` Returns an enumeration for traversing the structure.
boolean	`isEmpty()` Determine if there are elements within the structure.
java.util.Iterator<E>	`iterator()` Returns an iterator for traversing the structure.
E	`remove(E value)` Removes value from the structure.
int	`size()` Determine the size of the structure.
java.util.Collection<E>	`values()` Returns a java.util.Collection wrapping this structure.

Binary Search Tree

At home: how is **remove** implemented?

Binary Search Tree

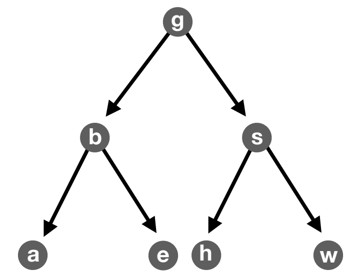
How might an iterator perform a given traversal?

Hint: use a stack!

Hint: the stack maintains all of the elements that still need to be traversed.

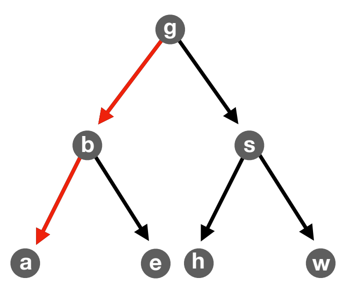
Tree balance

In the **worst case**, how long does it take to find an element in this binary search tree?



Suppose it is the letter a.

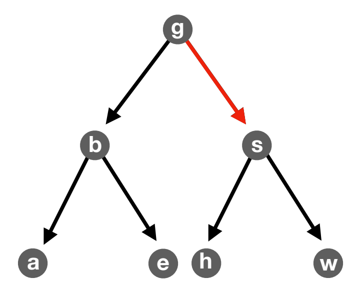
In the **worst case**, how long does it take to find an element in this binary search tree?



Suppose it is the letter a.

Finding a takes **two steps**.

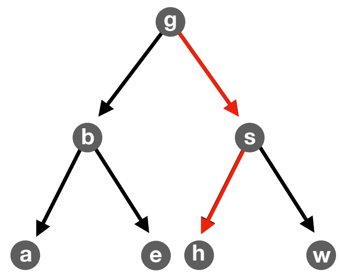
In the **worst case**, how long does it take to find an element in this binary search tree?



Suppose it is the letter s.

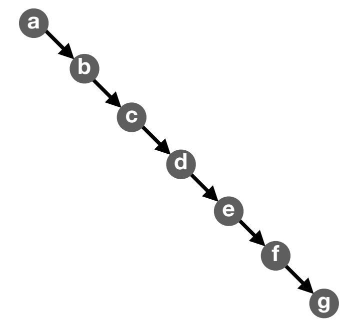
Finding s takes **one step**.

In the **worst case**, how long does it take to find an element in this binary search tree?



In the **worst case**, the time depends on the **length** of the **longest path**.

Suppose a friend gives you the following sequence of values: `[a,b,c,d,e,f,g]`



Ouch!!!

Worst case: O(n)

And asks you to store them in a binary tree to "make accessing them fast."

Is access **guaranteed** to be **fast**?

But what if your tree maintained the following property **on insertion**? (i.e., it is always true)

`isBalanced(t):`

`t` is balanced if and only if
- `t` is empty, or
- **all** of the following
  - `isBalanced(t.left)` is `true` **and**
  - `isBalanced(t.right)` is `true` **and**
  - `| height(t.left) - height(t.right) | ≤ 1`

Keep in mind: we know that the worst case has something to do with **height**.

But what if your tree maintained the following property **on insertion**? (i.e., it is always true)

Clearly a balanced tree.

Yeah, sure, there's no tree. Details, details…

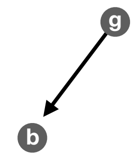
Time to access an element ~ **0 steps**

But what if your tree maintained the following property **on insertion**? (i.e., it is always true)

Balanced? **Yes.**

Max time to access an element ~ **0 steps**

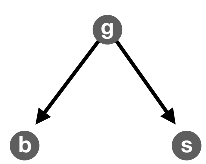
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



Balanced? **Yes.**

Max time to access an element: **1 step**

But what if your tree maintained the following property **on insertion**? (i.e., it is always true)

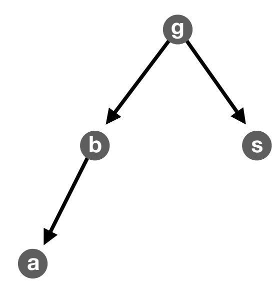


Balanced? **Yes.**

Changes nothing.

Max time to access an element: **1 step**

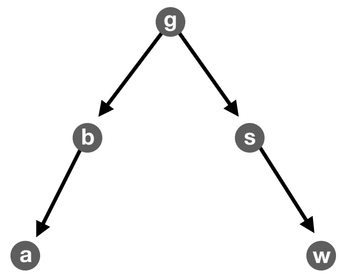
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



Balanced? **Yes.**

Max time to access an element: **2 steps**

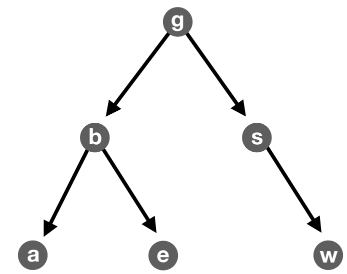
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



Balanced? **Yes.**

Max time to access an element: **2 steps**

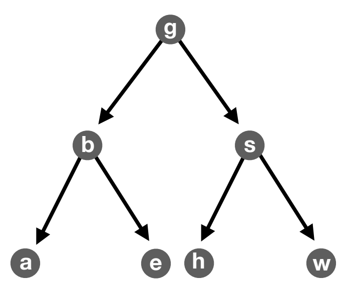
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



Balanced? **Yes.**

Max time to access an element: **2 steps**

But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



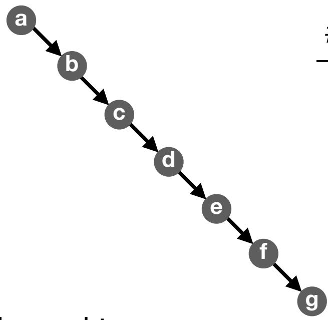
Balanced? **Yes.**

Max time to access an element: **2 steps**

# nodes	max time
1	**0 steps**
2	**1 step**
3	**1 step**
4	**2 steps**
5	**2 steps**
6	**2 steps**
7	**2 steps**
8	**3 steps**
…	**…**

This looks like **time** = $\log_2$(# nodes)

But does this hold up?



# nodes	max time
7	**6 steps**

Clearly **not** a balanced tree.

Logarithmic worst-case access time has something to do with the **compactness** of a tree; **height matters**.

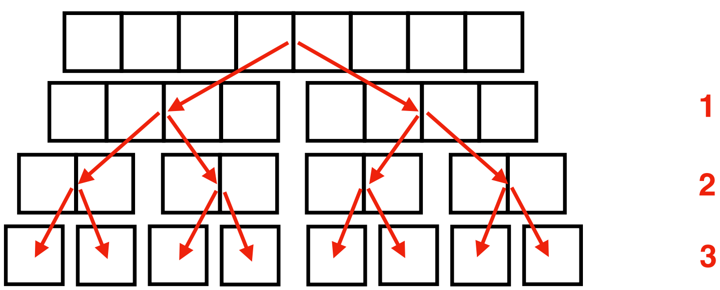
BST Big-O

Worst case time is **$O(\log_2(n))$** for a **balanced binary tree**.

Why?

What is min. binary tree height needed to store **n** nodes?

Cute theorem: **height $\geq \lfloor \log_2(n) \rfloor$**



Intuition: $\log_2(n)$ is the number of times you can **divide n nodes in halves.**

Implicit Data Structures

Recall: binary search tree

A **binary search tree** is a binary tree that maintains the **binary search property** as elements are added or removed. In other words, the **key** in each node:

- must be ≥ any **key** stored in the left subtree, and
- must be ≤ any **key** stored in the right subtree.

As with other ordered structures, order is maintained **on insertion.**

BST is an ADT

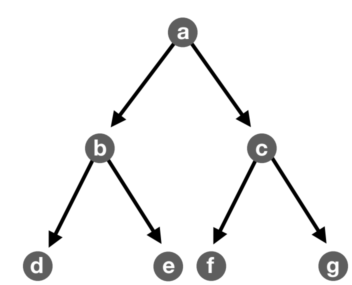
Do we actually need a **tree** to store a **tree**?

No. We can use an **implicit data structure** instead.

Implicit data structure

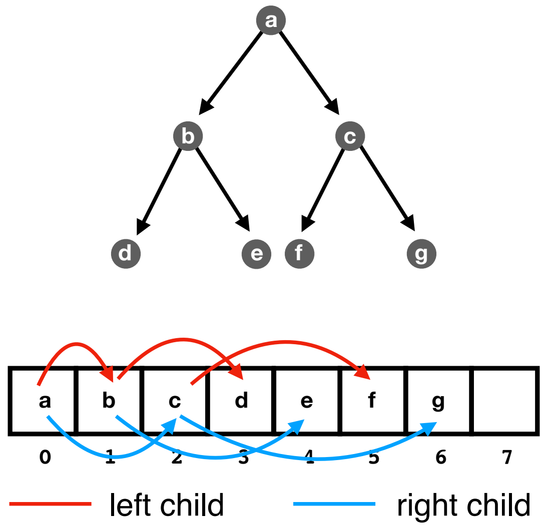
A **implicit data structure** or **space-efficient data structure** is a data structure that stores only **necessary** information. Instead of explicitly representing relationships between elements of the structure using references, an implicit structure **uses the relative positions of elements**.

Implicit binary tree

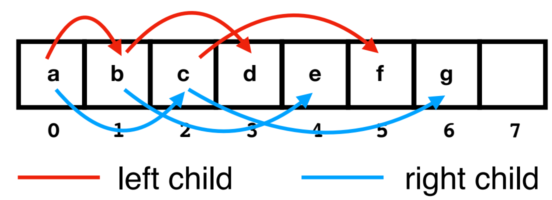


a	b	c	d	e	f	g	
0	1	2	3	4	5	6	7

Implicit binary tree



Implicit relationship



$$\texttt{leftChild(i)} = 2 \times i + 1$$

$$\texttt{rightChild(i)} = 2 \times i + 2$$

$$\texttt{parent(i)} = \lfloor (i - 1) / 2) \rfloor$$

Implicit Binary Search Tree

I will post an implementation on the course website.

Recap & Next Class

Today:

Tree balance

BST asymptotics

Implicit BST

Next class:

Maps