CSCI 136: Data Structures and Advanced Programming

Lecture 26

Maps

Instructor: Kelly Shaw

Williams

Topics

Tree Big-O

Map interface

Tree backed map

Your to-dos

1. Read **before Wed**: Bailey, Ch 15.4.
2. Lab 8 **(solo lab)**, **due Tuesday 11/15 by 10pm**.
3. Quiz, due Saturday evening.

Recall: binary search tree

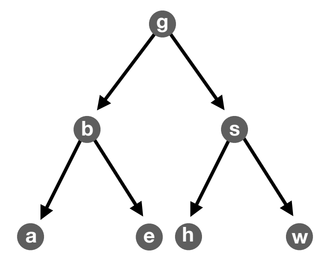
A **binary search tree** is a binary tree that maintains the **binary search property** as elements are added or removed. In other words, the **key** in each node:

- 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.**

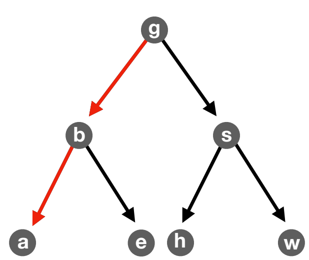
Tree balance

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



Suppose it is the letter a.

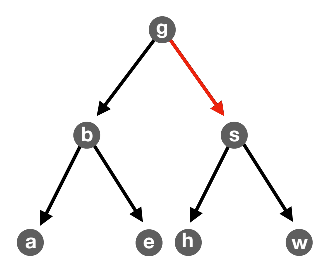
In the **worst case**, how long does it take to find an element in this binary search tree?



Suppose it is the letter a.

Finding a takes **two steps**.

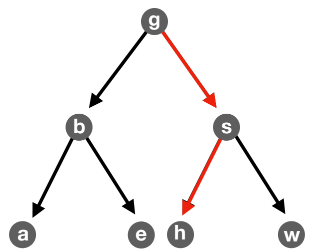
In the **worst case**, how long does it take to find an element in this binary search tree?



Suppose it is the letter s.

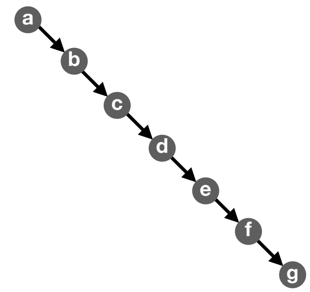
Finding s takes **one step**.

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…

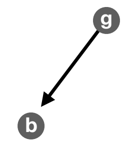
Time to access an element ~ **0 steps**

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**

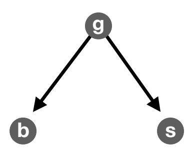
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



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)

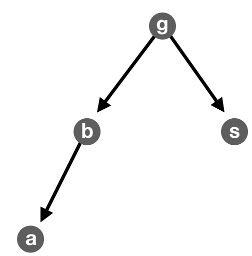


Balanced? **Yes.**

Changes nothing.

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

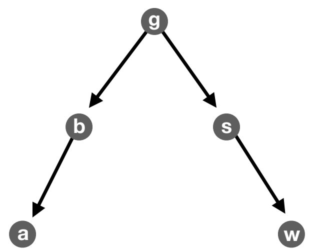
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



Balanced? **Yes.**

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

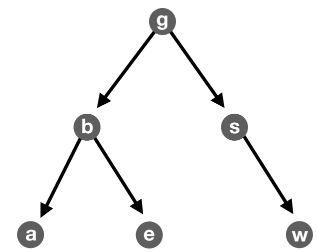
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



Balanced? **Yes.**

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

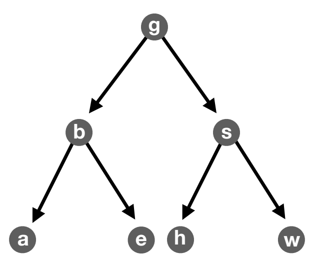
But what if your tree maintained the following property **on insertion**? (i.e., it is always true)



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)



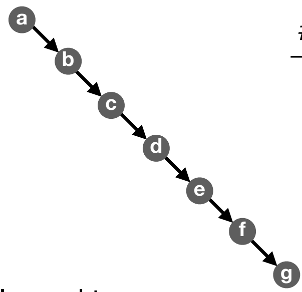
Balanced? **Yes.**

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**.

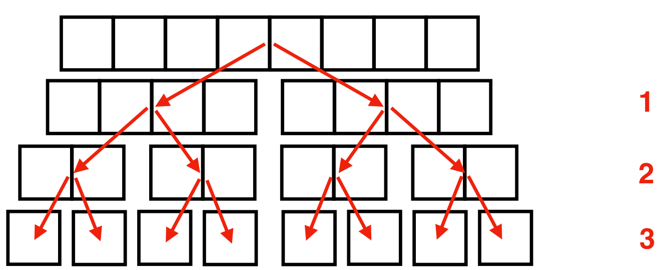
BST Big-O

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 ≥ $\lfloor \log_2(n) \rfloor$**



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

Maps

Map ADT

A **map** (also known as a **dictionary**, **associative array**, or **key-value store**) is an abstract data type that stores a collection of **(key, value)** pairs. Each key appears at most once in a collection. Maps support `lookup`, `insert`, and `remove` operations.

More formally, a map is a function with a finite domain.

Map ADT (intuition)

key	value
Dan	C
Jeannie	A
Bill J	B
Iris	A
Sam	A+

You've seen something like this before (hint: `SymbolTable`)

`structure5` `Map` interface

structure5

Interface Map<K,V>

All Known Subinterfaces:
OrderedMap<K,V>

All Known Implementing Classes:
AbstractMap, ChainedHashtable, Hashtable, MapList, Table

```
public interface Map<K,V>
```

Method Summary	
boolean	**containsKey**(K k)
boolean	**containsValue**(V v)
V	**get**(K k)
V	**put**(K k, V v)
int	**size**()
Structure<V>	**values**()

(I omitted some methods— see `structure5` docs)

structure5's only `Map` implementation

structure5

Class MapList<K,V>

```
java.lang.Object
  └structure5.MapList<K,V>
```

All Implemented Interfaces:
Map<K,V>

```
public class MapList<K,V>
extends java.lang.Object
implements Map<K,V>
```

Associations establish a link between a key and a value. An associative array or map is a structure that allows a disjoint set of keys to become associated with an arbitrary set of values. The convenience of an associative array is that the values used to index the elements need not be comparable and their range need not be known ahead of time. Furthermore, there is no upper bound on the size of the structure. It is able to maintain an arbitrary number of different pieces of information simultaneously. Maps are sometimes called dictionaries because of the uniqueness of the association of words and definitions in a household dictionary.

This implementation is based on a list, so performance for most operations is linear.

What's the problem with this implementation?

Let’s create a tree-backed Map

But first: how will it perform?

Recap & Next Class

Today:

- Map interface
- Tree backed map

Next class:

- Hash tables
- Collisions