Programming Languages and Techniques (CIS120)

Lecture 7
January 29th, 2016

Binary Search Trees
(Lecture notes Chapter 7)
What is the height of this tree?

( press # corresponding to answer)

Answer: 4
What is the result of this function on this tree?

1. []
2. [1;2;3;4;5;6;7]
3. 1
4. [4;2;1;3;5;6;7]
5. [4]
6. [1;1;1;1;1;1;1]
7. none of the above

Answer: 2

Let rec inorder (t:tree) : int list =
begin match t with
| Empty -> []
| Node (left, x, right) ->
  inorder left @
  (x :: inorder right)
end
• Homework 2 is online
  – due Tuesday

• Exam 1
  – Main exam, Tuesday evening Feb 16th, 6-8 PM
  – Make-up exam, Wednesday morning, Feb 17th, 9-11 AM
  – You must take the main exam if you can; I need to know ahead of time if you need to take the make-up exam
Trees as containers

Big idea: find things faster by searching less
Trees as Containers

- Like lists, binary trees aggregate data
- As we did for lists, we can write a function to determine whether the data structure *contains* a particular element

```
type tree =
   | Empty
   | Node of tree * int * tree
```
let rec contains (t:tree) (n:int) : bool = 
begin match t with
| Empty -> false
| Node(lt,x,rt) -> x = n ||
    (contains lt n) || (contains rt n)
end

• This function searches through the tree, looking for n
• In the worst case, it might have to traverse the *entire* tree
let rec contains (t:tree) (n:int) : bool = 
begin match t with 
| Empty -> false 
| Node(lt,x,rt) -> x = n || 
(contains lt n) || (contains rt n) 
end 

contains (Node(Node(Node (Empty, 0, Empty), 1, Node(Empty, 3, Empty)), 
5, Node (Empty, 7, Empty))) 7 

5 == 7 
  || contains (Node(Node (Empty, 0, Empty), 1, Node(Empty, 3, Empty))) 7 
  || contains (Node (Empty, 7, Empty)) 7 

(1 == 7 || contains (Node (Empty, 0, Empty)) 7 
  || contains (Node(Empty, 3, Empty))) 7 
  || contains (Node (Empty, 7, Empty)) 7 

contains (Node(Empty, 3, Empty)) 7 
  || contains (Node (Empty, 7, Empty)) 7 

contains (Node (Empty, 7, Empty)) 7
Challenge: Faster Search?
Binary Search Trees

• Key insight: *Ordered data can be searched more quickly*
  – This is why telephone books are arranged alphabetically
  – But requires the ability to focus on *half* of the current data

• A *binary search tree* (BST) is a binary tree with some additional *invariants*:

  • **Node(lt, x, rt)** is a BST if
    - lt and rt are both BSTs
    - all nodes of lt are < x
    - all nodes of rt are > x

• **Empty** is a BST

*An data structure *invariant* is a set of constraints about the way that the data is organized. “types” (e.g. list or tree) are one kind of invariant, but we often impose additional constraints.*
An Example Binary Search Tree

Note that the BST invariants hold for this tree.

• Node(1t, x, 1t) is a BST if
  - 1t and 1t are both BSTs
  - all nodes of 1t are < x
  - all nodes of 1t are > x
• Empty is a BST
Search in a BST: (lookup 8)

8 > 5
5

8 > 7
7

8 < 9
9

✓
Searching a BST

(*) Assumes that t is a BST *)

\[
\begin{align*}
\text{let } & \text{rec lookup } (t:\text{tree}) (n:\text{int}) : \text{bool} = \\
& \quad \text{begin match } t \text{ with} \\
& \quad \quad | \text{Empty} \rightarrow \text{false} \\
& \quad \quad | \text{Node}(lt,x,rt) \rightarrow \\
& \quad \quad \quad \quad \quad \text{if } x = n \text{ then true} \\
& \quad \quad \quad \quad \quad \text{else if } n < x \text{ then (lookup } lt \ n) \\
& \quad \quad \quad \quad \quad \text{else (lookup } rt \ n) \\
& \quad \text{end}
\end{align*}
\]

• The BST invariants guide the search.

• Note that lookup may return an incorrect answer if the input is not a BST!
  – This function assumes that the BST invariants hold of t.
BST Performance

- **lookup** takes time proportional to the *height* of the tree.
  - not the *size* of the tree (as it does with *contains*)
- In a *balanced tree*, the lengths of the paths from the root to each leaf are (almost) *the same*.
  - no leaf is too far from the root
  - the height of the BST is minimized
  - the height of a balanced binary tree is roughly $\log_2(N)$ where $N$ is the number of nodes in the tree
• Node(lt, x, rt) is a BST if
  - lt and rt are both BSTs
  - all nodes of lt are < x
  - all nodes of rt are > x
• Empty is a BST

Answer: no, 7 to the left of 6
Is this a BST??
1. yes
2. no

Answer: Yes
Node\((lt, x, rt)\) is a BST if:
- \(lt\) and \(rt\) are both BSTs
- all nodes of \(lt\) are \(< x\)
- all nodes of \(rt\) are \(> x\)

Empty is a BST

Is this a BST??
1. yes
2. no

Answer: no, 5 to the left of 4
• Node(\text{lt}, x, \text{rt}) is a BST if
  - \text{lt} and \text{rt} are both BSTs
  - all nodes of \text{lt} are < x
  - all nodes of \text{rt} are > x
• Empty is a BST

Is this a BST??
1. yes
2. no

Answer: no, 4 to the right of 4
• Node(lt, x, rt) is a BST if
  - lt and rt are both BSTs
  - all nodes of lt are < x
  - all nodes of rt are > x

• Empty is a BST

Is this a BST??
1. yes
2. no

Answer: yes
• Node(lt, x, rt) is a BST if
  - lt and rt are both BSTs
  - all nodes of lt are < x
  - all nodes of rt are > x
• Empty is a BST

Answer: yes
Constructing BSTs

Inserting an element
How do we construct a BST?

• Option 1:
  – Build a tree
  – Check that the BST invariants hold (unlikely!)
  – Impractically inefficient

• Option 2:
  – Write functions for building BSTs from other BSTs
    • e.g. “insert an element”, “delete an element”, ...
  – Starting from some trivial BST (e.g. \texttt{Empty}), apply these functions to get the BST we want
  – If each of these functions \textit{preserves} the BST invariants, then any tree we get from them will be a BST \textit{by construction}
    • No need to check!
  – Ideally: “rebalance” the tree to make lookup efficient
    (NOT in CIS 120, see CIS 121)
Inserting a new node: (insert t 4)
Inserting a new node: (insert t 4)
Inserting Into a BST

(* Insert n into the BST t *)
let rec insert (t:tree) (n:int) : tree =
  begin match t with
  | Empty -> Node(Empty,n,Empty)
  | Node(lt,x,rt) ->
      if x = n then t
      else if n < x then Node(insert lt n, x, rt)
      else Node(lt, x, insert rt n)
  end

• Note the similarity to searching the tree.
• Note that the result is a new tree with one more Node; the original tree is unchanged
• Assuming that t is a BST, the result is also a BST. (Why?)