Programming Languages and Techniques (CIS120)

Lecture 7
Jan 25, 2013

Binary Search Trees
Announcements

• Homework 2 is due *Tuesday*, Jan 29\textsuperscript{th}, at 11:59:59pm

• Did the CIS 120 web page get hacked this week?
**Representing trees**

```haskell
type tree =  
| Empty  
| Node of tree * int * tree
```

- Node (Node (Empty, 0, Empty), 1, Node (Empty, 3, Empty))
- Node (Empty, 0, Empty)
- Empty
Demo: bst.ml
Trees as Containers

- Like lists, trees aggregate ordered data
- Like lists, we can determine whether the data structure contains a particular element
- CHALLENGE: can we use the tree structure to make this process faster?
Searching for Data in a Tree

- Recall the contains function:

```scheme
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
```

- It searches through the tree, looking for n
  - In this case, the search is a *pre-order* traversal of the tree
  - Other traversal strategies would work equally well
- In the worst case, it might search through the entire tree
- Can we do better?
Search during (contains t 8)
Binary Search Trees (BST)

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

• A BST is a binary tree with additional *invariants*:

  • **Empty is a BST**
  • **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 $> x$
An Example Binary Search Tree

Note that the BST invariants hold for this tree.
Search in a BST: \((\text{lookup } t \ 8)\)
Searching a BST

• The BST invariants guide the search.
• Note that lookup may fail (i.e. return an incorrect answer) if the input is not a BST.

(* Assumes that t is a BST *)
let rec lookup (t:tree) (n:int) : bool =
  begin match t with
  | Empty -> false
  | Node(lt,x,rt) ->
    if x = n then true
    else if n < x then (lookup lt n)
    else (lookup rt n)
  end
How to we construct a BST?

• Option 1:
  – Write a function to check whether an arbitrary tree satisfies the BST invariant.
  – Call the check whenever we need to know about a given tree.

• Option 2:
  – Create functions that *preserve* the BST invariant
  – Starting from some trivial BST (e.g. *Empty*), we can apply such functions to get other BSTs
  – Examples: *insert* and *delete*
Checking the BST Invariants

(* Check whether all nodes of t are < n *)
let rec tree_less (t:tree) (n:int) : bool =
begin match t with
  | Empty -> true
  | Node(lt,x,rt) ->
    x < n && (tree_less lt n) && (tree_less rt n)
end

(* Determines whether t is a BST *)
let rec is_bst (t:tree) : bool =
begin match t with
  | Empty -> true
  | Node(lt,x,rt) ->
    is_bst lt && is_bst rt &&
    (tree_less lt x) && (tree_gtr rt x)
end

*Definition of tree_gtr omitted (it’s similar to tree_less)
Inserting a new node: (insert t 4)
Inserting a new node: (insert t 4)
Inserting Into a BST

(*) Inserts 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.
• Assuming that t is a BST, the result is also a BST. Why?