Programming Languages and Techniques (CIS120e)

Lecture 6
Sep 15, 2010

Binary Search Trees

Recap: Binary Trees

A binary tree is either empty, or a node with at most two children, both of which are also binary trees.

A leaf is a node whose children are both empty.

Integer Binary Trees in OCaml

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

```
let t1 = 
  Node(Node(Empty,1,Empty),  
       3,  
       Node(Empty,2,  
             Node(Empty,4,Empty)))
```

Announcements

- Homework 2 is on the web pages.
  - On-time due date: Wednesday 22 Sept. at 11:59:59pm
  - Get started early, and seek assistance if you get stuck!
Search during (contains t 8)

Binary Search Trees (BST)

- Key insight:
  - We can use an ordering on the data to cut down the search space
  - This is why telephone books are arranged alphabetically

- 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

Searching for Data in a Tree

- Recall the contains function:

```plaintext
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
  - Also, the tree might contain duplicate entries, which means more work

- Can we do better?

An Example Binary Search Tree

Note that the BST invariants hold for this tree.
**Search in a BST: (lookup t 8)**

![BST Diagram]

```
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
```

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

**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 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?
Deletion – No Children: (delete t 3)

Deletion – One Child: (delete t 7)

Deletion – One Child: (delete t 7)

Deletion – Two Children: (delete t 5)

If the node to be deleted has no children, simply replace it by the empty tree.

If the node to be deleted has one child, replace the deleted node by its child.
Deletion – Two Children: (delete t 5)

If the node to be delete has two children, promote the maximum child of the left tree.

Subtleties of the Two-Child Case

- Suppose Node(lt,x,rt) is to be deleted and lt and rt are both themselves nonempty trees.
- Then:
  - There exists a maximum element, m, of lt (why?)
  - m is smaller than every element of rt (why?)

- To promote m we replace the deleted node by:
  Node(delete lt m, m, rt)
  - i.e. we recursively delete m from lt
  - Note the resulting tree satisfies the BST invariants

- Question: will this always work?