CMSC 341
Lecture 9

Announcements

Proj 1 results

Next week

Exam 1 Mon 16

- review questions will appear on website

Path Lengths

There is a relationship between the IPL and EPL of full BTs.

If n_i is the number of internal nodes in a FBT, then

$EPL(n_i) = IPL(n_i) + 2n_i$

Example:

n_i =

$EPL(n_i)$ =

$IPL(n_i)$ =

$2\ n_i$ =

Proof of Path Lengths

Prove: $EPL(n_i) = IPL(n_i) + 2\ n_i$ by induction

Base: $n_i = 0$ (single node, the root)

$EPL(n_i) = 0$

$IPL(n_i) = 0$; $2\ n_i = 0$ $0 = 0 + 0$

Assume for N: True for all FBT with $n_i < N$

Prove for N+1: Let n_{iL}, n_{iR} be # of int. nodes in L, R subtrees.

$n_{iL} + n_{iR} = N - 1 \Longrightarrow n_{iL} < N;\ n_{iR} < N$

$EPL(n_{iL}) = IPL(n_{iL}) + 2\ n_{iL}$

$EPL\ (n_{iR}) = IPL(n_{iR}) + 2\ n_{iR}$

$EPL(n_i) = EPL(n_{iL}) + EPL(n_{iR}) + n_{iL} + 1 + n_{iR} + 1$

$IPL(n_i) = IPL(n_{iL}) + IPL(n_{iR}) + n_{iL} + n_{iR}$

$EPL(n_i) = IPL(n_i) + 2\ n_i$

Constructing Trees

Is it possible to reconstruct a BT from just one of its pre-order, inorder, or post-order sequences?

Constructing Trees (cont)

Given two sequences (say pre-order and inorder) is the tree unique?

Tree Implementations

What should methods of a tree class be?

Tree class

```
template <class Object>
class Tree {
   public:
   Tree(const Object &notFnd);
   Tree (const Tree &rhs);
   ~Tree();

   const Object &find(const Object &x) const;
   bool isEmpty() const;
   void printTree() const;
   void makeEmpty();
   void insert (const Object &x);
   void remove (const Object &x);
   const Tree &operator=(const Tree &rhs);
```

Tree class (cont)

```
private:
   TreeNode<Object> *root;
   const Object ITEM_NOT_FOUND;
   const Object &elementAt(TreeNode<Object> *t) const;
   void insert (const Object &x, TreeNode<Object> * &t) const;
   void remove (const Object &x, TreeNode<Object> * &t) const;
   TreeNode<Object> *find(const Object &x,
                          TreeNode<Object> *t) const;
   void makeEmpty(TreeNode<Object> *&t) const;
   void printTree(TreeNode<Object *t) const;
   TreeNode<Object> *clone(TreeNode<Object> *t)const;
   };
```

Tree Implementations

- Fixed Binary
  - element
  - left pointer
  - right pointer
- Fixed K-ary
  - element
  - array of K child pointers
- Linked Sibling/Child
  - element
  - firstChild pointer
  - nextSibling pointer

TreeNode : Static Binary

```
template <class Object>
class BinaryNode {
   Object element;
   BinaryNode *left;
   BinaryNode *right;

   BinaryNode(const Object &theElement, BinaryNode *lt,
   BinaryNode *rt) : element (theElement), left(lt),
   right(rt) {}

   friend class Tree<Object>;
   };
```

TreeNode : Static K-ary

```
template <class Object>
class KaryNode {
   Object element;
   KaryNode *children[MAX_CHILDREN];

   KaryNode(const Object &theElement;

   friend class Tree<Object>;
   };
```

TreeNode : Sibling/Child

```
template <class Object>
class KTreeNode {
   Object element;
   KTreeNode *nextSibling;
   KTreeNode *firstChild;

   KTreeNode(const Object &theElement, KTreeNode *ns,
   KTreeNode *fc) : element (theElement),
   nextSibling(ns), firstChild(fc) {}

   friend class Tree<Object>;
   };
```