3137 Data Structures and Algorithms in C++

Lecture 4

July 17 2006

Shlomo Hershkop

1

Announcements

- please make sure to keep up with the course, it is sometimes fast paced…
- for extra office hours, please drop us an email etc
- make sure to review class notes/slides
- July 26th take home midterm….

2

Announcements

- review c++
  - not enough emails back on time
  - memory issues
    - leaks
    - misuse
    - casting
    - new/delete
    - overflow

3

Outline

- Trees and more trees
- balancing
- advanced trees
- reading: Chapter 4-4.5, chapter 6-6.5

4

Tree Traversal

- should be comfortable with the idea of how to run through a tree
- Example: should be able to code from scratch the pre-order traversal

5

binary tree height

- Anyone recall the tree height algorithm ??
- code ??

6

```
int height (TreeNode *node){

int lefth, righth;

if (node == NULL){
   return -1; }

lefth = height(node->left);
righth = height(node->right);

return lefth > righth ? lefth+1 : righth + 1 ;
}
```

7

- one of the nice things about binary trees are some interesting mathematical properties
- lets discuss some of them

8

Prove

- Can you prove that at level i the binary tree has 2^i slots ??

9

- Show:
  - at 0 it has 2^0 slots
- Assume
  - ??
- Prove
  - At level N each of the 2^N slots will have at most 2 children
  - so next level: $2 * 2^N = 2^{N+1}$
  - which means level N+1 has 2^{N+1}

10

Prove

- if we have N internal nodes, how many external nodes ??

11

- If N internal nodes, we have N+1 external nodes

- with 1 node = 0 internal , 0+1 external
- Assume: ???
- Proof :
  - for the tree to move from N internal to N+1 internal, an external must add children.
  - subtract one external (its now internal)
  - can have up to +2 children

12

Definition

- Complete Binary Tree:
  - this is a binary tree which is filled in left to right on each level
  - some examples

13

Advantages

- one big advantage to complete binary trees is in representing them
- you can now use an array to represent the tree
- Given a node A[i]
  - where would the left and right child be ??

14

- A[i]
  - left is A[i*2]
  - right A[i*2+1]
  - parent A[i/2]
  - root A[1]
    - can save A[0] for other stuff
  - any ideas on how to check if A[i] is a leaf ??

15

Height

- For a binary tree with N internal nodes, its height is between log n and n-1
  - worst case
    - linked list
  - best
    - any ideas ??

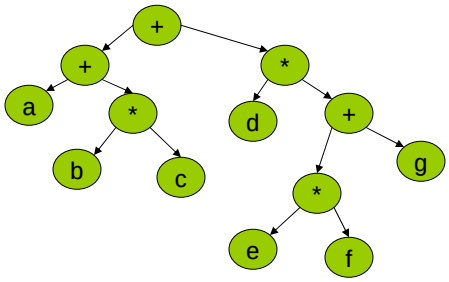
16

Expression Tree

- Another flavor of binary trees are expression trees
  - internal nodes: arithmetic operations
  - leaves: numbers/variables

17

Example



•Can you turn this into postfix ??

•What about infix ??

18

question

- can you write a pseudo code algorithm for converting postfix to expression trees ??

19

```
while ( /*some input*/) {
//read in symbol
if(isSymbol(s)) {
   //make tree t from s
   //push to a stack
}
else {
//pop 2 trees t1, t2 from the stack
//combine to form tree t3
//push to stack
}


}
```

20

run time on trees

- basic operations on trees:
- Insert:
- Find:
- what are the worst case, and best case ??

21

- idea is to have the tree as balanced as possible as a BST to give us best case scenario
- need to keep it BST property while balanced

22

- with BST

- insert
  - 5, 14, 3, 12
- now:
  - 14, 12, 5, 3

23

Overhead

- if you want to keep the BST tree balanced you might end up with a situation where inserts could end up being
  n + log n = O(n)

- need a better way

24

Adelson-Velskii & Landis

- AVL Tree
  - data structure which keeps the trees (almost) balanced at all times, using AVL property
- AVL Property
  1. H(right) == H(left)
  2. H(right) differs from H(left) by 1

H(empty tree) = -1

H(node) = longest path

25

run time

- because the tree is always balanced:
  - inserts = log n
  - find = log n

26

AVL Rotation

- rotation is the operation of keeping an AVL tree balanced
- Remember that because its always an AVL tree, the only way to unbalance it is by insert/delete a node
- if α is the node that needs rebalancing, it means α subtree was edited in some way
- divides itself into 4 cases

27

1. Just inserted into left subtree of left child
2. Just inserted into right subtree of left child
3. Just inserted into left subtree of right child
4. Just inserted into right subtree of right child

- single rotations 1,4
- double rotations 2,3
- lets show this graphically:

28

- insert 5,2,8
- 1,4,7,3

- now insert 6
  - which is unbalanced ?
  - which rule is used ?

- how do you code this ??


```
/**
    * Internal method to insert into a subtree.
    * x is the item to insert.
    * t is the node that roots the subtree.
    * Set the new root of the subtree.
    */
   void insert( const Comparable & x, AvlNode * & t )
   {
       if( t == NULL )
           t = new AvlNode( x, NULL, NULL );
       else if( x < t->element )
       {
           insert( x, t->left );
           if( height( t->left ) - height( t->right ) == 2 )
               if( x < t->left->element )
                   rotateWithLeftChild( t );
               else
                   doubleWithLeftChild( t );
       }
       else if( t->element < x )
       {
           insert( x, t->right );
           if( height( t->right ) - height( t->left ) == 2 )
               if( t->right->element < x )
                   rotateWithRightChild( t );
               else
                   doubleWithRightChild( t );
       }
       else
           ;  // Duplicate; do nothing
       t->height = max( height( t->left ), height( t->right ) ) + 1;
   }
```

```
/**
     * Rotate binary tree node with left child.
     * For AVL trees, this is a single rotation for case 1.
     * Update heights, then set new root.
     */
    void rotateWithLeftChild( AvlNode * & k2 )
    {
        AvlNode *k1 = k2->left;
        k2->left = k1->right;
        k1->right = k2;
        k2->height = max( height( k2->left ), height( k2->right )
   ) + 1;
        k1->height = max( height( k1->left ), k2->height ) + 1;
        k2 = k1;
    }
```

31

```
/**
* Rotate binary tree node with right child.
* For AVL trees, this is a single
* rotation for case 4.
* Update heights, then set new root.
*/
void rotateWithRightChild( AvlNode * & k1 )
{
AvlNode *k2 = k1->right;
k1->right = k2->left;
k2->left = k1;
k1->height = max( height( k1->left ), height( k1-
  >right ) ) + 1;
k2->height = max( height( k2->right ), k1->height )
  + 1;
        k1 = k2;

}
```

32

```
/**
* Double rotate binary tree node: first left child.
* with its right child; then node k3 with new left child.
* For AVL trees, this is a double rotation for case 2.
* Update heights, then set new root.
*/
    void doubleWithLeftChild( AvlNode * & k3 )
    {
        rotateWithRightChild( k3->left );
        rotateWithLeftChild( k3 );
    }
```

33

```
/**
    * Double rotate binary tree node: first right child.
    * with its left child; then node k1 with new right child.
    * For AVL trees, this is a double rotation for case 3.
    * Update heights, then set new root.
    */
   void doubleWithRightChild( AvlNode * & k1 )
   {
      rotateWithLeftChild( k1->right );
      rotateWithRightChild( k1 );
   }
```

34

question

- so using an AVL tree what do we get if we print inorder traversal ??

35

- so using the AVL tree to sort, what is the cost of sorting n items ??
- which tree operations are involved ??

36

Limitations

- one limitation is that the tree might be spread across memory
- as you need to travel down the tree, you take a performance hit at every level down
- one solution: store more information on the path

37

B-trees

- 2-3 B-trees are tree DS found in databases and file systems
- automatically balanced
- keep all data at leaf level
- non leaf keys guide search by storing 1 or 2 keys (2,3) children
- root is either leaf or between 2,3 children

38

Example

- simple case: room in leaf
  - insert: 3,1,5
- Harder: split when full
  - inserting: 8, 7, 6
- Even Harder: move up the tree

39

Data Structure

- we spoke about queues, but many times would like to add a priority to the Queue DS
- Priority Queue
  - Insert
  - FindMin
- Example: phone call service center

40

Implementations

- Any ideas ?

41

Implementations

- sorted linked list
  - insert – O(???)
  - findmin – O(???)
- unsorted linked list
  - insert
  - findmin

42

Question

- What about using a BST ??
  - insert
  - findmin

43

Heap

- A heap is an implementation of a priority queue
  - property: it is a binary tree that is completely filled except on the bottom level
  - Any advantages ??

44

Next time

- reading:
  - start chapter 5
- Feedback :
  - please submit some feedback before leaving
- Any questions ??

45