Lecture Notes

1 Trees Overview

A tree is a particular kind of directed graph, specifically, one that:

- Is connected
- Has exactly one node with in-degree 0 (the root)
- Has in-degree of one for every other node in the graph

There is exactly one path from the root to any other node.

Why study trees?

- To model hierarchical data
- As a convenient data structure for for important ADTs, like Sets, Maps, PriorityQueues, etc. Leads to efficient operations.

2 Definitions

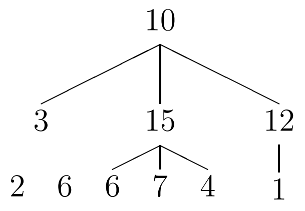
- Tree exercises activities based on tree terminology page

3 Implementation

3.1 Pointer

- Commonly organized as a dynamically allocated linked structure, similar to linked lists
- Tree root specified by a pointer.
  - For fixed out-degree: array of subtrees pointers
  - For arbitrary out-degree: pointer to first child and pointer to next sibling

Example:

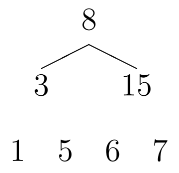


3.2 Arrays

ACBTs can be stored very efficiently with an array. Array representation, starting at element 1, the index of the array is its node in level order:

8	3	15	1	5	6	7

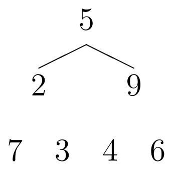
`last` points to the last node in the ACBT.



You can view an array as a ACBT, and vice-versa. Indexing:

- Given a node at index i, what is the index of its parent? floor(i/2)
- Given a node at index i, what is is the index of its left child, its right child? i*2, i*2+1.

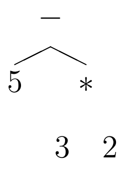
PPQ Draw the following tree as parent/sibling linked, fixed out-degree 2 linked, an ACBT array



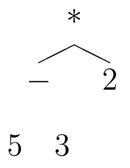
4 Expression Trees

- An expression tree uses a tree to represent an expression
  - E.g. Algebraic, Boolean/Logical
  - We'll focus on binary expression trees for binary operators
- The non-terminals are operators, their operands are their children

What is the value of $5 - 3 * 2$? Why -1 and not 2? Precedence.



and not

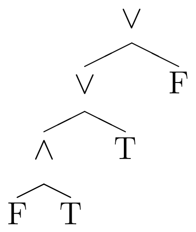


- You evaluate an expression tree as follows
  - If a terminal, evaluates to itself
  - Else, evaluate left and right children and apply operator to the results

What is the precendence in arithmetic: $\div, *, +, -$?

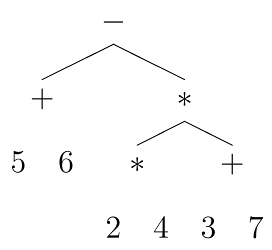
ABCD If multication and division are left associative, what is the expression tree for $5*3*2$? $(5 * 3) * 2$.

$F \wedge T \vee T \vee F$

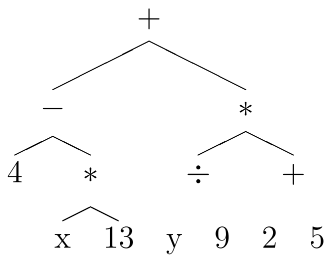


Out of curiousity, what does this evaluate to? T.

PPQ: Draw the expression tree for $5 + 6 - 2 * 4 * (3 + 7)$

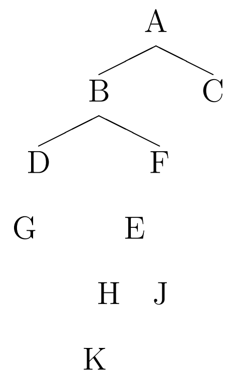


Example: $4 - x * 13 + y \div 9 * (2 + 5)$
$(4 - (x * 13)) + ((y \div 9) * (2 + 5))$

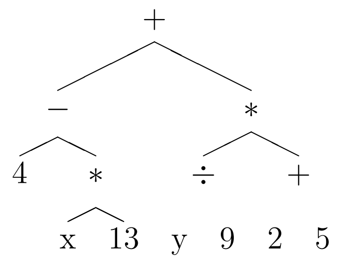


5 Traversals

- Preorder aka NLR
- Inorder aka LNR
- Postorder aka LRN
- Level order



Preorder: `A B D G F E H K J C`
Inorder: `G D B K H E J F A C`
Postorder: `G D K H J E F B C A`
Levelorder: `A B C D F G E H J K`



Prefix (polish) notation: `+ - 4 * x 13 * / y 9 + 2 5`
Like Scheme `(+ (- 4 (* x 13)) (* (/ y 9) (+ 2 5)))`
Infix notation: `4 - x * 13 + y / 9 * 2 + 5`
Notice the corruption: `((4 - (x * 13)) + ((y / 9) * 2)) + 5`
Postfix (reverse polish) notation: `4 x 13 * - y 9 / 2 5 + * +`
Like backwards scheme : `((4 (x 13 *) -) ((y 9 /) (2 5 +) *) +)`

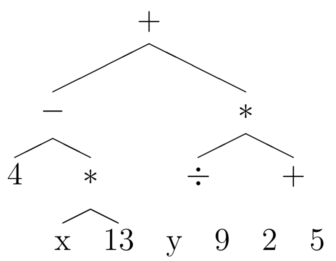
- Polish notation named after polish logician Jan Lukasiewicz, who invented the notation in 1924 (according to Wikipedia).

CS 241 Lecture Notes

Wed Oct 25, 2006

Example: $4 - x * 13 + y \div 9 * (2 + 5)$

$(4 - (x * 13)) + ((y \div 9) * (2 + 5))$



Expression to Tree algorithm in the book example.

6 Binary Search Trees

- A binary tree where ever node is associated with a key, and where the key at any vertex is greater than all of the keys in its left subtree and less than all keys in the right subtree. (We assume unique keys here, although we could permit duplicates). Recursively:
  - A leaf is a BST
  - A vertex is the root of a BST if its key value is greater than that of its left childs and less than or equal to that of its right child, so long as each child is either empty or itself the root of a BST.
- BSTs are a convenient way to organize a collection of data that itself has no hierarchical structure, because they permit efficient operations.
- Can be used as the data structure to implement Set, Dictionary and Priority Queue abstract data types.

ABCD: Draw several small trees, vote yes BST (A) or not BST (B). Why or why not?

PPQ: Draw three different BSTs all containing the keys $1, 2, 3, 4$

6.1 Common Operations

6.1.1 Insert

Add a new node into the tree (or make no change if duplicate).

```
Insert(T,k)
   if T is empty then
      Create new node k here
   elsif T.key > k then
      Insert(T.left,k)
   elsif T.key < k then
      Insert(T.right,k)
   fi
end
```

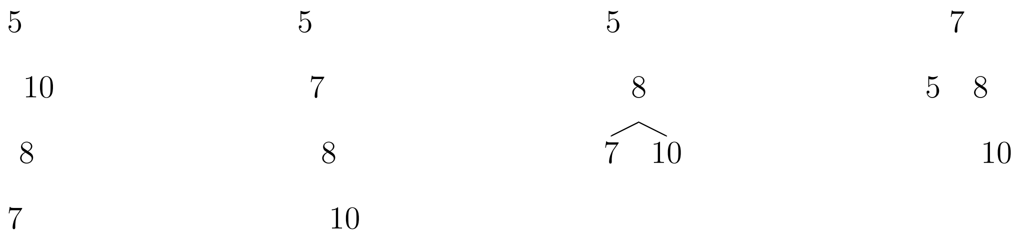
Note: only new leaves are added!

Example: Insert $10, 5, 15, 12, 17, 13, 14$

- Time complexity: $O(h)$ where h is the height of the tree
  - Worst case: $O(N)$ (just a linked list)
  - Average case: $O(\lg N)$

ABCD: Does the order of insertions affect the shape of the tree? A=yes B=no.

ABCD: Inserting $5, 10, 8, 7$ leads to which of the following trees?



PPQ: What order of insertions could have led to the above trees?

6.1.2 Search

Return pointer to TreeNode with key k, or null if no node has key k:

```
Search(T,k)
   if T is null (empty) || T.key == k then
      return T
   elsif T.key > k then
      return Search(T.left,k)
   else
      return Search(T.right,k)
   fi
end
```

- Time complexity: $O(h)$ where h is the height of the tree
  - Worst case: $O(N)$ (just a linked list)
  - Average case: $O(\lg N)$

6.1.3 Maximum/Minimum

Find the maximum (or minimum) value in the tree rooted at T, return pointer to max (or min) node (null if T is empty).

```
TreeMinimum(T)
   if T == null (empty) then
      return null
   fi
   while T.left != null do
      T := T.left
   od
   return T
end
```

Example: Apply it to a few different nodes in a BST.

- Time complexity: $O(h)$ where h is the height of the tree
  - Worst case: $O(N)$ (just a linked list)
  - Average case: $O(\lg N)$

Max is just the symmetric case.

6.1.4 Successor

Returns the pointer to the next largest node after T in the tree, or null if none exists

```
Successor(T)
   if T.right != null then
      return Minimum(T.right)
   fi
   Y := T.parent
   while Y != null && T == Y.right do
      T := Y
      Y := Y.parent
   od
   return Y
end
```

Example Find successor to 10 in the insertion example above.
Example Find successor to 14 in the insertion example above.
Example Find successor to 17 in the insertion example above.

PPQ or ABCD: How many children can the Successor to a node have?

- Time complexity: $O(h)$ where h is the height of the tree
  - Worst case: $O(N)$ (just a linked list)
  - Average case: $O(\lg N)$

Predecessor is just the symmetric case.

6.1.5 PrintInSortedOrder

To print all of the keys in the tree in sorted order, simply do an in-order traversal, printing each node as they are visited.

- Time complexity: $O(N)$ (best and worst)

6.1.6 Delete

Delete a node with key k from tree T, if it exists.

```
Delete(T,k)
   X := Search(T,k)
   if X has no children
      just remove X
   elsif X has one child
      remove X, promoting the child into its place
   else
      Y := Successor(X)
      Copy Y over X
      Remove original Y using one of the two above rules
   fi
end
```

- Time complexity: $O(h)$ where h is the height of the tree
  - Worst case: $O(N)$ (just a linked list)
  - Average case: $O(\lg N)$

PPQ: Delete 17 from insertion sort example, then delete 10.

7 Tree Sort

For a set of n elements, call Insert(T,A[n]), and then do an inorder traversal.

```
TreeSort(A)
   T := empty tree
   for I in 0 .. (N-1) do
      Insert(T,A[I])
   od
   PrintInSortedOrder(T)
end
```

- Time complexity:
  - Average case: $O(N \lg N)$ (for the N insertions)
  - Worst case: $O(N^2)$ (for the N insertions)
  - **Question:** What triggers the worst case behavior?
- Space complexity:
  - $O(N)$ (for all of the TreeNodes)
  - There's also $O(\lg N)$ for the traversal call stack, but that's low order

- Average case: each insertion costs $O(\lg N)$, N insertions $O(N \lg N)$
- Worst case: each insertion costs $O(N)$, N insertions $O(N^2)$
- Both cases: memory usage $O(N) + O(\lg N) = O(N)$
- Traversal: costs $O(N)$ time, $O(\lg N)$ memory

CS 241 Lecture Notes
Fri Oct 27, 2006

8 Encoding

Say we have the message: `DATA STRUCTURES`

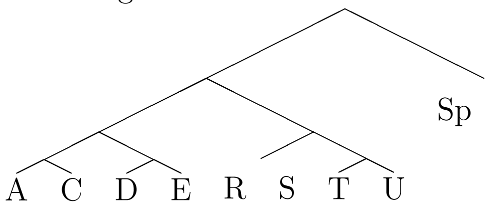
How many bits does it take to encode this message using a fixed-length encoding?

- If we use ASCII, with 256 values, then 8 bits per character times 15 characters = 120 bits
- If we use just the letters present, then we have we need enough bits to represent 9 distinct values, or ceil lg(9) or 4 bits, then we have 15 * 4 = 60

Lets assign a fixed length coding to these characters:

```
A: 0000
C: 0001
D: 0010
E: 0011
R: 0100
S: 0101
T: 0110
U: 0111
Sp : 1000
```

This can be represented by a coding tree:



Each left branch denotes a zero, and each right branch a 1. The characters are in the leaves, and the unique path to the character is its code. Because it is a fixed-length code, all characters/leaves have the same level.

- What is the following message: `0110010001110011`? (True).
- Characters are only located in the terminals/leaves. What if we have allowed characters in non-terminals? We couldn't guarentee an unambiguous parse.
- A code in which all characters are at the terminals is known as a **prefix code** because it satisfies the **prefix property**: no code is a prefix of another code.
- What if we allowed variable length codes whose characters were still at the leaves? For example, make `Sp` have the code `1`.
- Could we still parse unambiguously? Yes.
- How much space would we need in this case to store the the message? $14 * 4 + 1 = 57$.
- We have compressed the code, the compression ratio is $\frac{57}{60} = .95$

9 Huffman Codes

The idea between huffman codes is to exploit knowledge of how frequent characters occur to design a coding tree that assigns shorter codes to more frequent characters, at the expense of potentially longer codes for less frequent characters.

Huffman invented an algorithm to construct optimal prefix codes, giving the name to huffman codes, huffman coding trees, etc.

- Step 1: Count the frequencies and store in a counts table:

  ```
  A: 2
  C: 1
  D: 1
  E: 1
  R: 2
  S: 2
  T: 3
  U: 2
  Sp : 1
  ```

- Step 2:
  Make a tree for each character that includes its label and weight. The weight is set to its count. These will be leaves.
- Step 3:
  For $n-1$ times, do the following:
  Create a new node, set is left child to be the character with the least weight, set its right child to be the node with the 2nd to least weight. Set the new nodes weight to be the some of its children's weights.

Example:

- Join C - D
- Join E - Sp
- Join A - (C D)
- Join (E Sp) - R
- Join S - U
- Join T - (A (C D))
- Join ((E Sp) R) - (S U)
- Join (T (A (C D))) - (((E Sp) R) (S U))

Lengthof encoded message:

```
A:  010 x 2
C:  0110 x 1
D:  0111 x 1
E:  1000 x 1
R:  101 x 2
```

```
S:  110 x 2
T:  00 x 3
U:  111 x 2
Sp: 1001 x 1
```

46 bits

$$\frac{46}{60} = .766$$

When would we be able to acheive the best compression? When would we acheive the worst? What types of files do you think fall into these categories?

Discuss Program 2

CS 241 Lecture Notes
Mon Oct 30, 2006

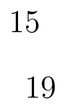
10 BST Operations

- Initialize
- Insert
- Search
- Retrieve
- Delete
- Traverse
- Save/Restore

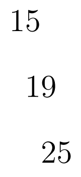
Go over the code in the book, and then the following example:

- T : Tree;
- Initialize(T);
- Insert(T,15);

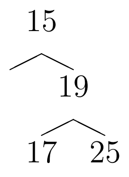
- Insert(T,19);



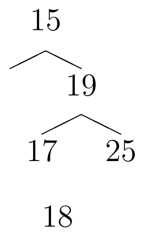
- Insert(T,25);



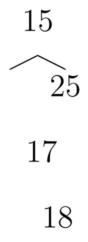
- Insert(T,17);



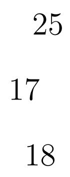
- Insert(T,18);



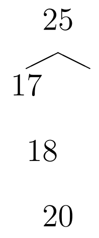
- Delete(T,19);



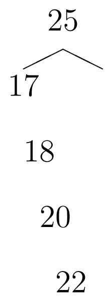
- Delete(T,15);



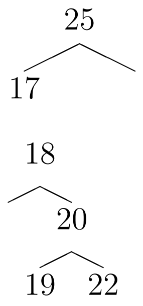
- Insert(T,20);



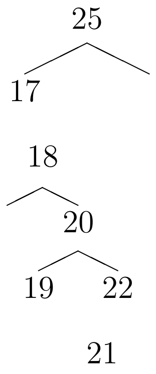
- Insert(T,22);



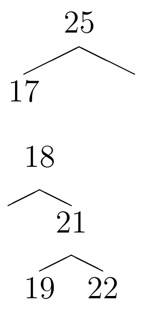
- Insert(T,19);



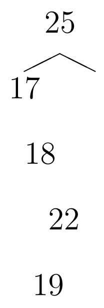
- Insert(T,21);



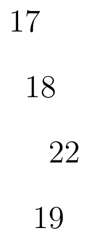
- Delete(T,20);



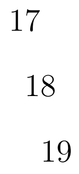
- Delete(T,21);



- Delete(T,25);

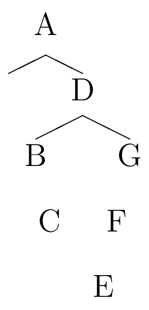


- Delete(T,22);



What happens if you insert 1 through 100 in order? Tree of height 100. A tree taking this shape degenerates to what data structure? List.

11 BST Save/Restore



We need to traverse all items to save them off, but in what order? What happens if we perform an inorder traversal?

```
A B C D E F G
```

Worst case scenario - bad idea.
What happens if we perform a preorder traversal?

```
A D B C G F E
```

Rebuilds the original tree. Works for me.