EE 368

Weeks 5 (Notes)

Chapter 5: Trees

Skip pages 273 - 281, Section 5.6

- If A is the root of a tree and B is the root of a subtree of that tree, then A is B's **parent** (or father or mother) and B is A's **child** (or son or daughter).
- A node with no children is a **leaf/terminal node**.
- The node A is the **ancestor** of node B if A is B's parent or is the parent of some ancestor of B. Then ancestors of a node are all the nodes along the path from the root to that node.
- **Siblings** (brothers, sisters) Two nodes are siblings if they are children of the same parent.

Some Definitions

Degree of a node: The numbers of children of this node.

Degree of a tree: Maximum degree of nodes in that tree

Level: Node at Level i has its children at level i + 1. Root's level is 0.

Height (depth): Maximum level of leaves in a tree

Binary Trees:

Types:

Strictly Binary Tree

Complete Binary Tree

Almost Complete Binary Tree

Advanced Binary Trees: Balanced and AVL

Others

Applications:

Expression Trees

Heaps

Search Trees

Counting Trees

Huffman Coding

Representing General Trees

Binary Trees:(cont.)

Representation:

Arrays

Linked Lists

Threaded Binary Trees

Binary Tree: More Definitions

- **Left child**.
- **Right child**.
- A node with no children is a **leaf node**.
- **left and right descendants**

  - B is a **left descendant** of A if B is a left child of A or is a descendant of a left child of A.

  - similarly for **right descendants**

Some Binary Trees:

Strictly Binary Tree: Every non-leaf node has exactly two children

Complete Binary Tree of height d is a strictly binary tree whose all leaves are at level d. Its size is $2^{d+1} - 1$

Almost Complete Binary Tree of depth d:

1. Each leaf in the tree is either at level d or at level d - 1. Any node at level less than d - 1 has two sons.
2. For any node (x) in the tree if there is a right descendant at level d, it must have a left son and every left descendant of (x) is either a leaf at level d or has two sons.

More Definitions

- **A complete binary tree of depth d** is a strictly binary tree whose leaves are all at level d.

Example: {a level 0 complete binary tree}

Example: {complete binary tree of depth 1}

Example: {complete binary tree of depth 2}

Size of a Complete Binary Tree

If # of nodes at level k = m

How many nodes at level k + 1?

Finding the maximum # of nodes in a binary tree of height d (to be done in class):

Labeling (Numbering) of Binary Trees

The nodes of a complete and an almost complete binary tree can be labeled intelligently. Let the label for the root be 1.

Then the left son' label = (2 * parent's label number)

Right son's label= (2 * parent's label number) + 1

This rule is applied to every node.

Summary of Binary Trees:

Strictly Binary Tree:

Every non-leaf node has two children.

If **n** leaf nodes, it has **2n - 1** nodes.

Complete Binary Tree:

All levels are full. Therefore, leaf nodes only at the last level. Its size is

$2^{d+1} - 1$

Almost Complete Binary Tree:

1. Each leaf in the tree is either at level d or at level d - 1. Any node at level less than d - 1 has two sons.
2. For any node (x) in the tree if there is a right descendant at level d, it must have a left son and every left descendant of (x) is either a leaf at level d or has two sons.

   $2^d - 1 <$ number of nodes $< 2^{d+1} - 1$

Applications of Binary Tree

1. To encode information where 2-way decisions are made at each point (Huffman Coding) Binary
2. Expression trees
3. Sorting (Heaps) (to be discussed later on)
4. Search (to be discussed later on)

Applications of Binary Tree

Two phases for the application on binary trees:

1. Building
2. Traversal

Huffman Coding

Suppose we want to code a message with letters A . . . D using bits. A sequence of 2 bits is appropriate.

Symbol	Code	(Code 1)
A	00	
B	01	
C	10	
D	11	

To code a message with n symbols

(out of k distinct symbols) will always require ?? bits.

Huffman Coding

Suppose we want to minimize the amount of storage for a string of symbols.

With a variable length of code depending on how frequent the symbol is in the message, we can achieve a savings!

Reason: We will be using a variable length code, we must not allow one code for a certain letter be the prefix of another (Ambiguity!)

Example

Consider Code 2:

Symbol	Code
A	0
B	110
C	10
D	111

Variable Length Code

Example

Compare Code 1 with Code 2 for the message: ABACCDA

	A	B	A	C	C	D	A
Code 1	00	01	00	10	10	11	00

14 bits

Code 2	0	110	0	10	10	111	0

13 bits

With Code 2, since no string is a prefix of another, we can recover the original symbols by scanning the bit stream from left to right!

Huffman Coding Procedure

Generation of Huffman code for alphabets A, B, C, D to encode the message

A B A C C D A

How do we choose code word for the alphabet? Depends on the message!

Two Steps:

- Build a binary tree (Huffman Tree).
- Generate code for each alphabet from the tree.

Huffman Tree Generation

Bottom up Approach:

- From the given message, find frequency of each symbol.
- Select two symbols with the smallest frequency, join them to generate a single “symbol” having a frequency as the sum the two corresponding frequencies.
- Repeat Step 2, until all the symbols in the message are merged to form a single symbol.

This process allows us to build a binary tree (Huffman Tree).

Each leaf represents a symbol of original alphabet.

Each non-leaf node is a “symbol”.

Huffman Coding Procedure

Once the tree is generated, traverse and encode it as follows:

- To root, do not assign any bit.
- Start from root and move downward and assign
  - 0 to each L child
  - 1 to each R child
- Traverse path from root to each leaf node to identify the Huffman Code for each leaf node.

Example

For the message, ABACCDA, we find:

symbol	A	B	C	D
frequency	3	1	2	1

- Select two symbols with the smallest frequency, B and D, and make a single “BD” node with frequency 2.

symbol	A	BD	C
frequency	3	2	2

Example

- Again select two symbols with the smallest frequency, joining them, BD and C --> CBD with the frequency 4.

symbol	A	BDC
frequency	3	4

- Again, we pick two symbols to finish up.

  “ACDB” --> frequency = 7

Use this bottom up approach to build the Huffman coding tree.

Huffman Coding Procedure

(to be discussed in class)

Assign 0 and 1 to nodes (root is assigned nothing “e”) and traverse paths from root.

A - 0

C - 10

B - 110

D - 111

Binary Tree Representation

- Arrays. Tree labels for the complete and almost complete binary tree can be mapped into index values of a 1-dimensional array. (see homework problem)

- linked list (ECE264 material, see notes on the Web)