Trees III

Huffman Trees

Announcement

- Final Exam
  - Wednesday, February 25, 2004
  - 8:00am – 10:00 am
  - 70-3435
- Please report all exam conflicts now!

Project 2 Notes

- Writeup now on the Web
- Due Dates:
  - Minimum submission due Friday, Feb 6th
    - Entry.java & Document.java
    - Partial implementation of Directory.java provided
  - Final submission due Sunday, Feb 15th
    - A little more than a week after the minimum!
    - Complete Directory.java & VFSystem.java
    - Integration tests WILL be performed.

Announcement

- Exam 2
- Wednesday, Feb 4th
- Covering:
  - Java IO
  - Recursion
  - Asymptotic Analysis
  - Searching
  - Sorting

Before we begin

- Questions on trees?

Huffman Trees

- Another “real live” application of binary trees
- Binary (I.e. 0, 1) encoding of characters

Huffman Trees

- Suppose we want to "encode" a text message into a sequence of 1's and 0's:
  - Each character will be given a binary code
  - No code for one character is a prefix of the code for another character
  - More frequently used characters have shorter codes.

Huffman Trees

- Example:
  - Say we wish to encode 5 characters a,b,c,d,e
  - One possible encoding:

Char	Code
a	000
b	001
c	010
d	011
e	100

Huffman Trees

- Using this encoding
- abca = 000 001 010 000

Char	Code
a	000
b	001
c	010
d	011
e	100

Huffman Trees

- Suppose we are given frequency of occurrences for each character

Char	Occurrence	Code
a	12 %	000
b	40 %	11
c	15 %	01
d	8 %	001
e	25 %	10

Huffman Trees

- Chars that occur more frequently have shorter codes.

Char	Occurrence	Code
a	12 %	000
b	40 %	11
c	15 %	01
d	8 %	001
e	25 %	10

Huffman Trees

- No code is a prefix of any other code
- Using this encoding
  - abca = 000 11 01 000
  - 10 bits
- Using last encoding
  - abca = 000 001 010 000
  - 12 bits

Char	Code
a	000
b	11
c	01
d	011
e	10

Huffman Trees

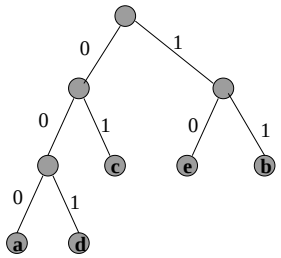
- This sequence of codes can be represented by a binary tree:
  - Leaves represent characters
  - Following left child represents appending a 0 to a code
  - Following right child represents appending a 1 to a code

Huffman Trees

- To obtain a code for a given character
  - Start at the root
  - Find a path to the character's leaf node
  - Append a 0 to a code every time you follow a left child
  - Append a 1 to a code every time you follow a right child.

Huffman Tree

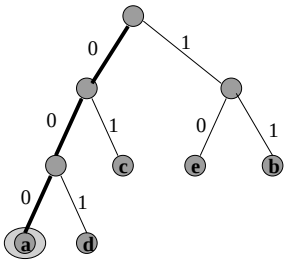
Char	Code
a	000
b	11
c	01
d	011
e	10



Huffman Tree

Find the code for **a** 0 0 0

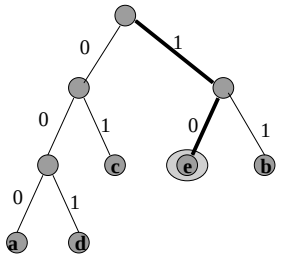
Char	Code
a	000
b	11
c	01
d	011
e	10



Huffman Tree

Find the code for **e** 1 0

Char	Code
a	000
b	11
c	01
d	011
e	10



Decoding using a Huffman Tree

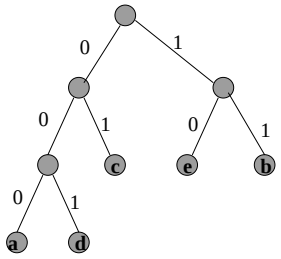
- Start with the root
- For each "bit" follow an edge
- When you get to leaf, write the char associated with the leaf
- Go back to the root.

Decoding using a Huffman Tree

Decode:

00100101100011101

d d c e d b c



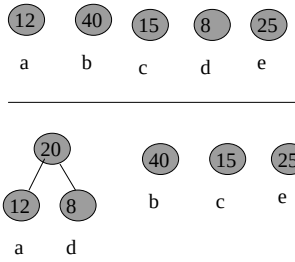
Huffman Coding

- How to build these Huffman trees
  - Given:
    - Set of characters to be encoded
    - A "weight" assigned to each character (indicating its frequency of occurrence).

Huffman Coding

- How to build these Huffman trees
  1. Begin with a forest of trees. All trees are one node with the weight equal to the weight of the character.
  2. Repeat until there is only 1 tree
     1. Choose 2 trees: T_1 and T_2 with the smallest weights and combine creating a new tree with left subtree = T_1 and right subtree = T_2

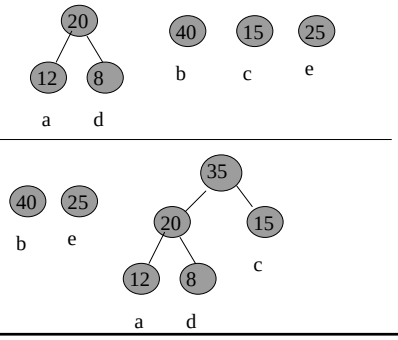
Huffman Coding



Start: Each char is its own tree

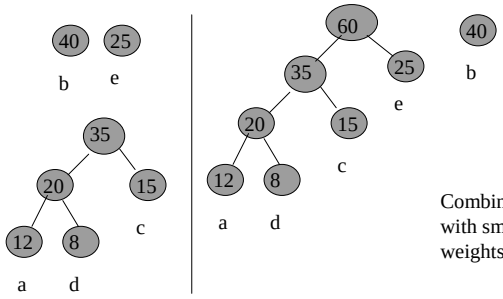
Combine trees with smallest weights: a & d

Huffman Coding



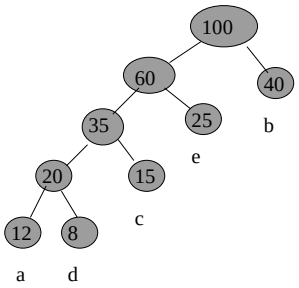
Combine trees with smallest weights:

Huffman Coding



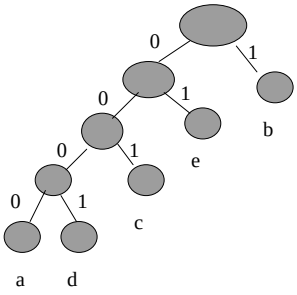
Combine trees with smallest weights:

Huffman Coding



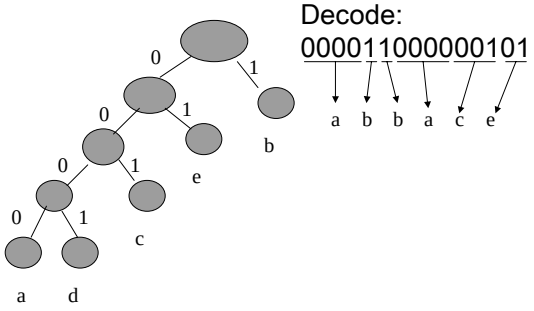
Finally, combine the last 2 trees

Huffman Coding



Char	Code
a	0000
b	1
c	001
d	0001
e	01

Huffman Coding



Huffman Coding

- This is an example of a greedy algorithm.
  - Only considers information available during a given iteration.
  - Local decision → Global solution
- You will be implementing the Huffman coding algorithm in Lab 9.

Summary

- Huffman Coding
  - Used to encode characters into 0's and 1's
  - More frequent characters have smaller codes
  - Result represented by a binary tree
  - Built using a greedy algorithm
  - Questions?

Next time

- Introduction to hashing