CS 240 Project 4
Message Compression and Encryption

Youhan Fang

yfang@cs.purdue.edu

11/6/2009

Motivation

- **Efficiency** and **Security** issues in data communications
- Efficiency: use fewer bits
- Security: encrypt the message

Efficiency: Huffman Coding

- **Entropy**: shortest average code length

  H = - p(c1)log2(p(c1)) - p(c2)log2(p(c2)) - ...
  - p(cn)log2(p(cn))

  – you can not achieve it in general
- But we can use some method to get close to it.
- In this project, we use **Huffman coding.**
- Once you get the Huffman code, you may compare the average code length with the entropy.

Security: RC4

- **RC4**: Encrypt the message by using some "key"
- Encrypt and decrypt in the same way

Huffman coding

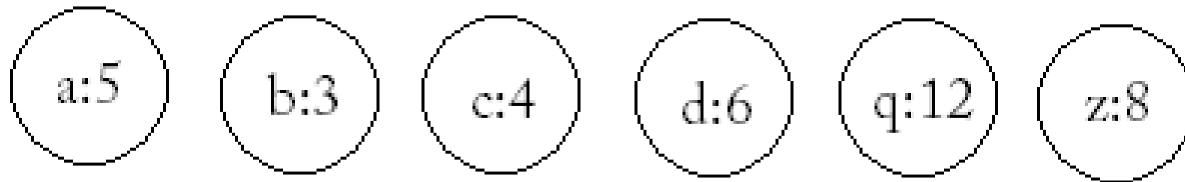
- Basic idea: more frequently used characters have shorter lengths
  - so we may have shorter average code length
- Key steps:
  - 1. find the frequency of each character
  - 2. recursively extract two nodes with the lowest frequency and combine them in one node
  - 3. assign the bit patterns according to the built tree

Huffman Coding

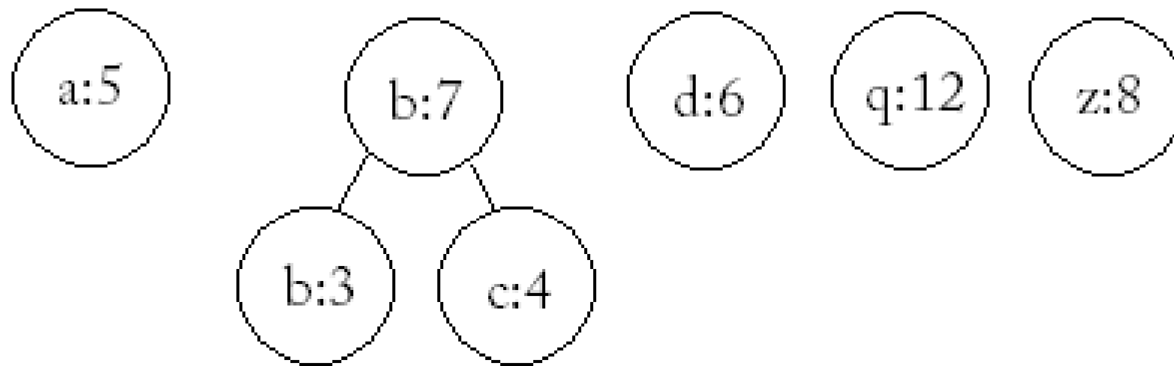
- Example:

  qzzzzqqdaaaqqbcdzdqqdabcazqqczcbddzqqq

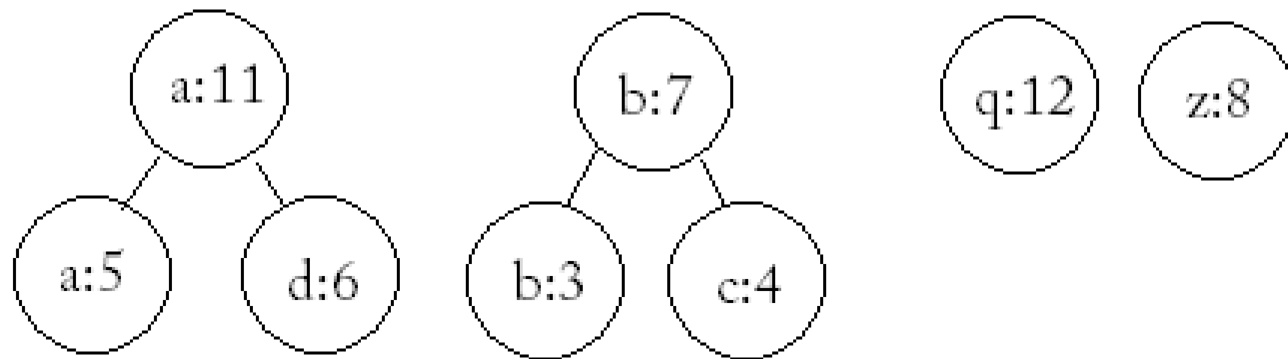
  a:5,b:3,c:4,d:6,q:12,z:8



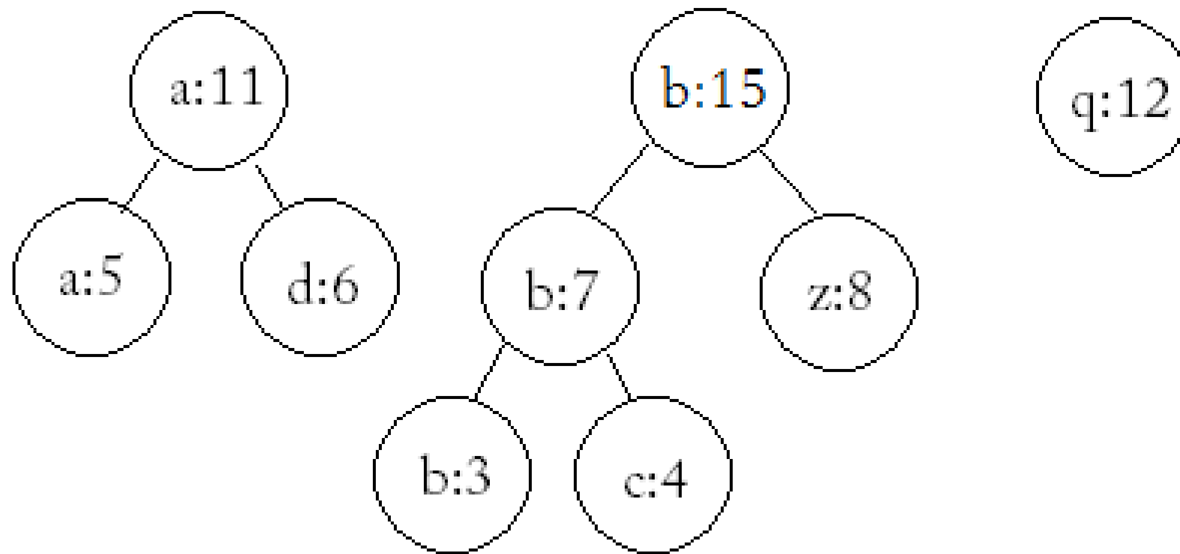
Huffman coding



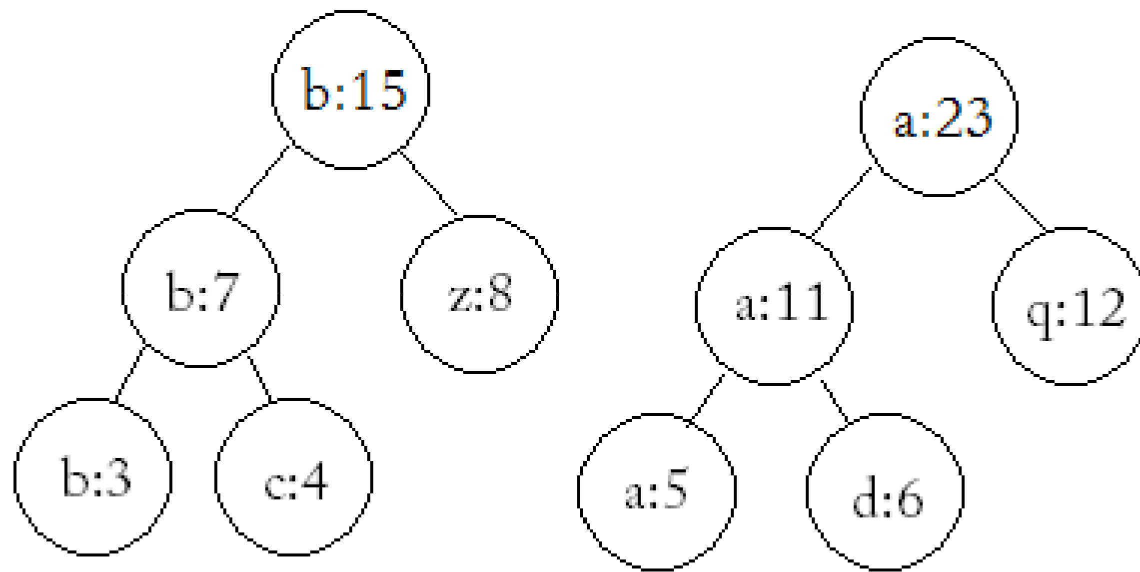
Huffman coding



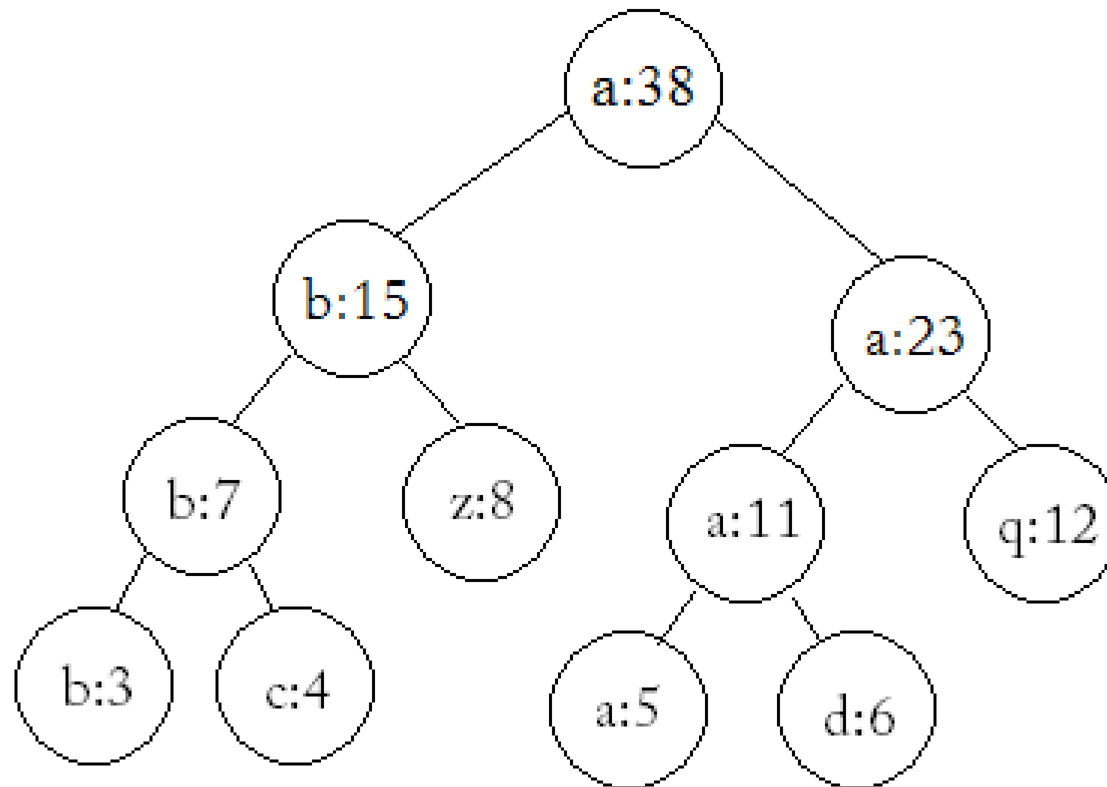
Huffman coding



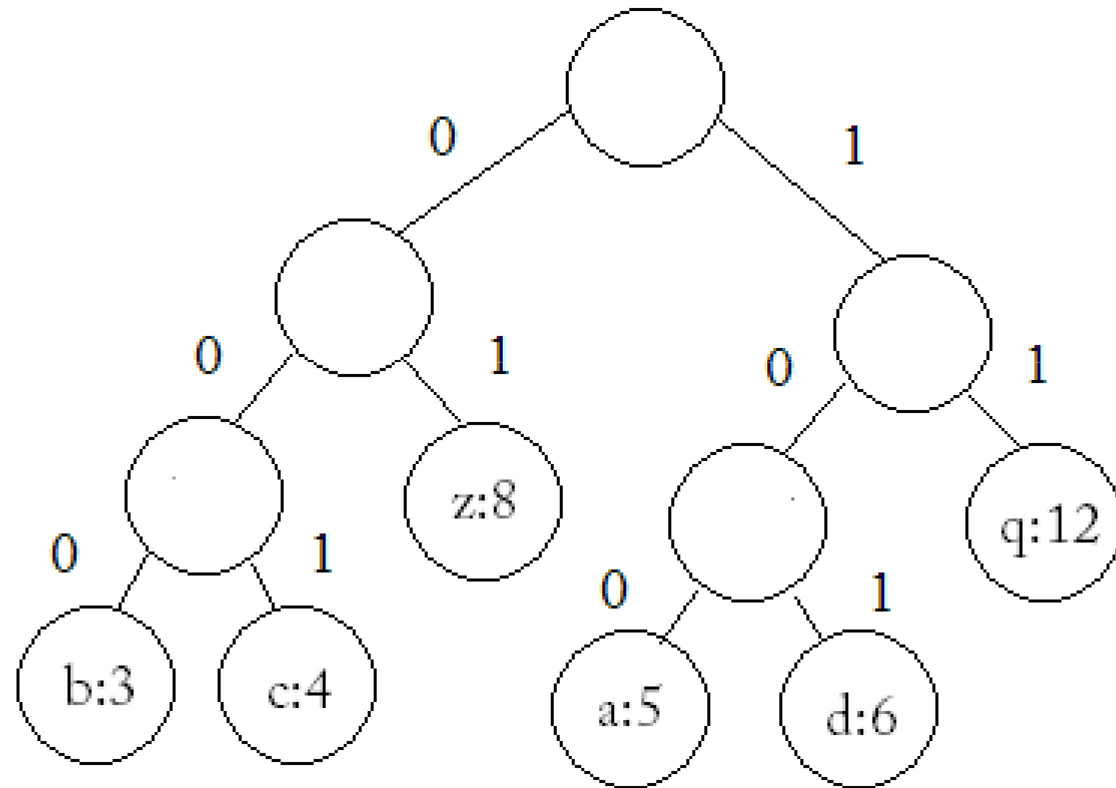
Huffman coding



Huffman coding



Huffman coding



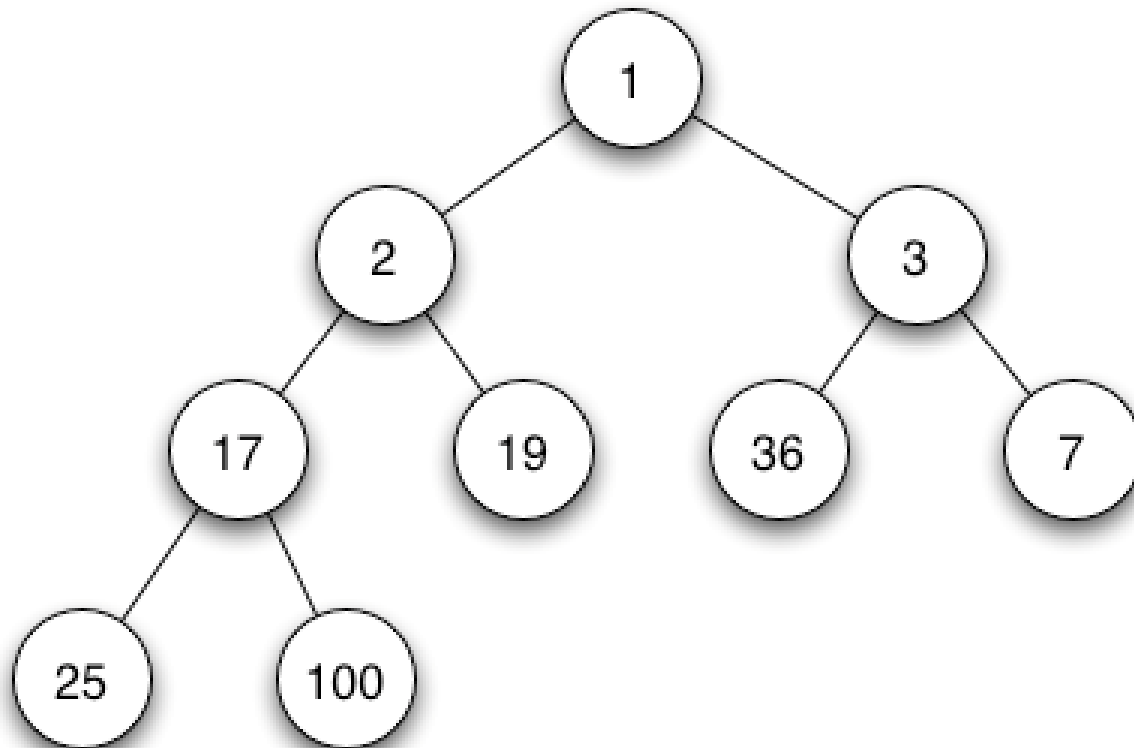
Huffman coding

- So, to build a Huffman tree, you need to do the following things recursively:
  - extract two nodes with the lowest frequency from the pool
  - combine them in one node
  - put it back into the pool
- How to do it efficiently?
  - use min heap
  - fast extract the min value and insert new values
- Note: using min heap is only a SUGGESTION, not a REQUIREMENT.
  - you can just use an array and sort it.

Min Heap

- A heap can be seen as a binary tree with two additional constraints:
  - 1. The *shape property*: the tree is an *almost complete binary tree*; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
  - 2. The *heap property*: each node is greater than (note: min heap applies “less than”) or equal to each of its children according to some comparison predicate which is fixed for the entire data structure

Min Heap



Implementation of Huffman coding

- If you feel min heap is “over-skilled” for this task, we introduce you a simpler method: using a sorted array.
- We take the same example

a:5,b:3,c:4,d:6,q:12,z:8

Implementation of Huffman coding

- 1. Sort this array

b:3, c:4, a:5,d:6, z:8, q:12

- 2. Extract the two nodes with the lowest frequency

b:3, c:4

Implementation of Huffman coding

- 3. combine them into one node

b:3, c:4 => b:7

Newnode->leftchild = b:3
Newnode->rightchild = c:4
Newnode->character=b
Newnode->frequency=7
…

- 4. put it back to the array in the proper position

a:5,d:6, **b:7**, z:8, q:12

Implementation of Huffman coding

- 5. Do step 2, 3 and 4 recursively
- 6. Finally you will have only one node

a:38

- 6. Because in each iteration you saved all the useful information in each node (e.g. leftchild, rightchild, frequency, …), so you have successfully built the tree!

Padding

- We can't guarantee that the bit length of the encoded message is divisible by 8. If it is not, we have to use padding (i.e., add 0's) to the end of the encoded message.
- Example:

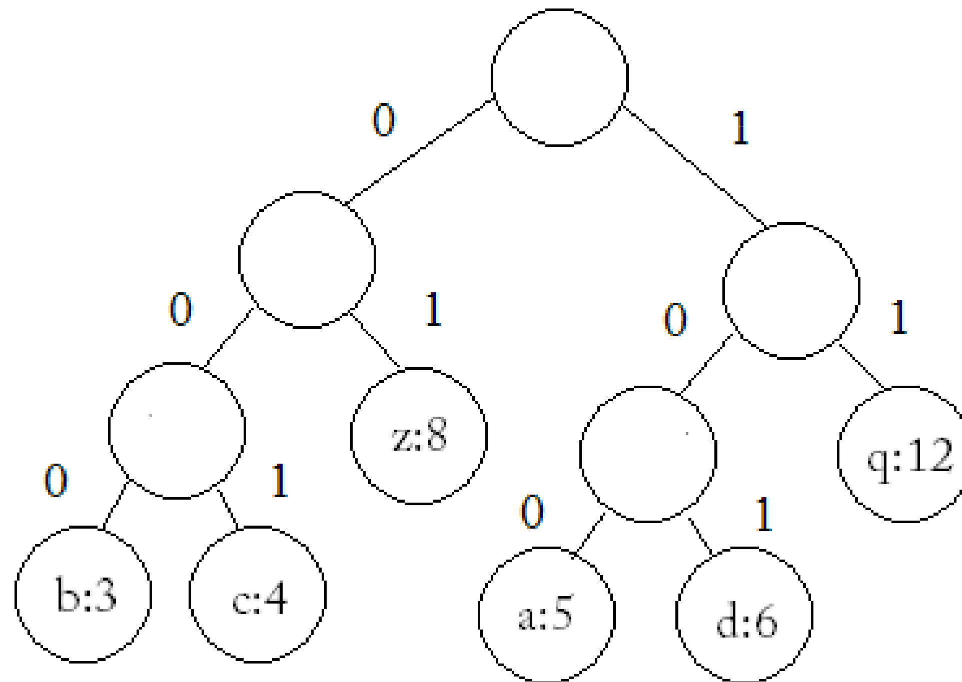
Encoded message length: 52

Padding number: 4

Total length: 56

Huffman Decoding

- Simply go from the root to one leaf
- Example: 100101001 => adc



RC4

- Initialize: generate a string according to the ‘key’.
  - note: the string has the same length with the input
- Do XOR: do XOR on the input and the generated string **bit by bit**.
- There is a simple implementation by using C in the link we gave to you!

RC4

- Initialization

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap(S[i],S[j])
endfor
```

RC4

- XOR:

```
i := 0
j := 0
k := 0
while k<input_length:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    output S[(S[i] + S[j]) mod 256] ^ input[k++]
endwhile
```

Hints

- The character of the new node should be the character with the smaller ASCII value, no matter what the frequencies of the two children are
  - some of you have already made the mistake
- You don’t need to worry about those new terms (entropy, heap, RC4, etc.). Just focus on the most important and difficult part: Huffman coding
  - once you finish Huffman coding, other parts become trivial
- We don’t require the efficiency of your code, so you may choose the method for which you feel comfortable
  - e.g. you can ignore min heap and just use array

Hints

- We added two additional test cases for you to test your program. They are more advanced.
- Note: you still need to come up with additional test cases of your own

- Questions?