Data Structures and Algorithms

Session 15. March 23, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

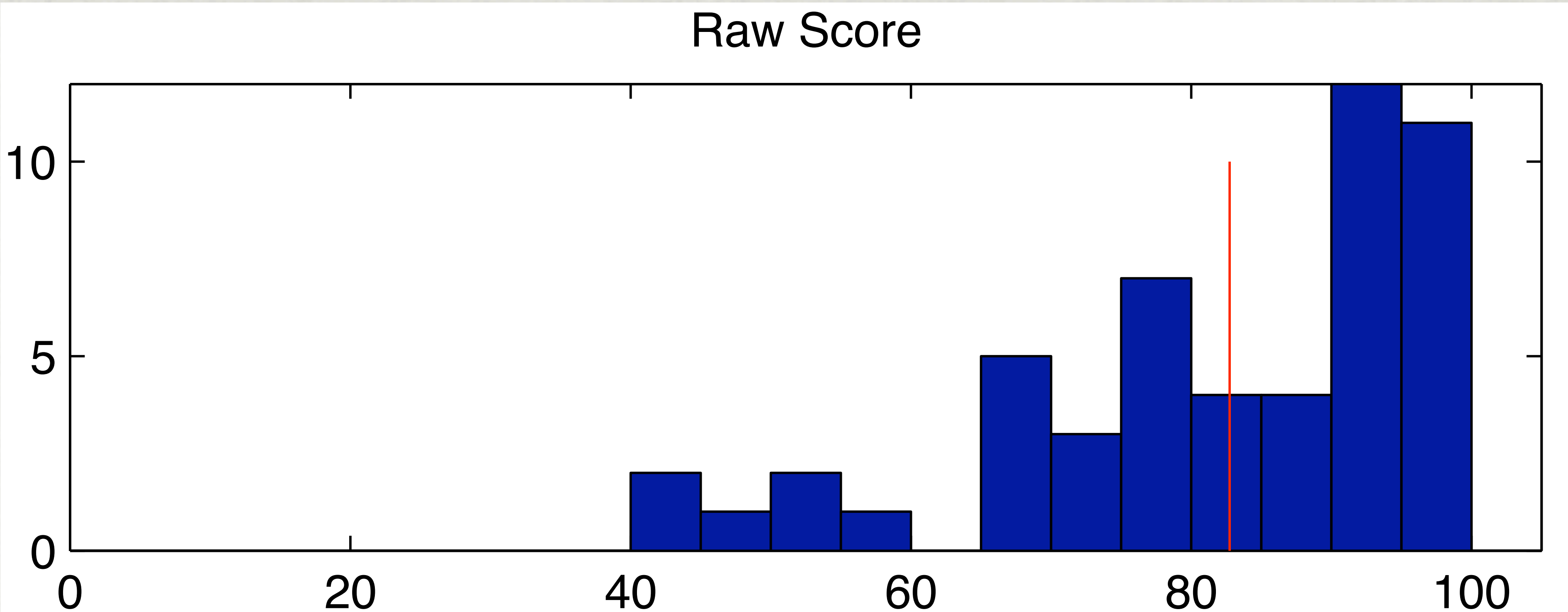
Announcements

- Homework 4 up on website
  - New GraphDraw.java, should fix Concurrency Exceptions
- Homework 3 solutions up

Today's Plan

* Midterm Solutions
* Huffman Coding Trees
    * Data compression method

Midterm



- Average was 82 out of 100
- Scaling formula: 100*(x+30)/130

Huffman Codes

- Basic lossless data compression
- General purpose codes are fixed length:
  - e.g., ASCII character code is 7 bits
    ‘a’ is 7 bits, ‘!’ is 7 bits, ‘~’ is 7 bits
- Strategy: encode more common characters with shorter codes

Example

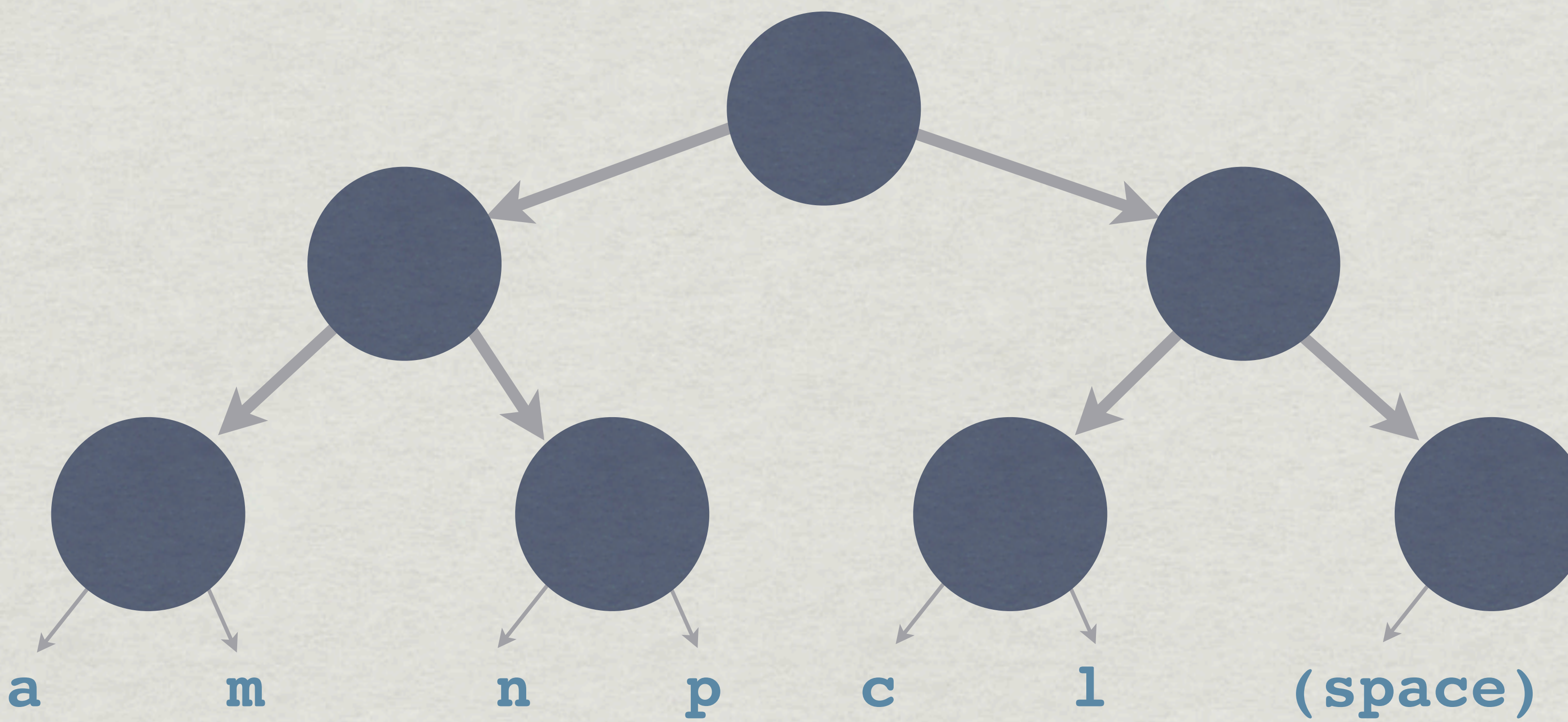
- “a man a plan a canal panama”
- 7 characters: a m n p l c (space)
- We can use 3 bits to create a unique code for each

a	m	n	p	l	c	space
000	001	010	011	100	101	110

- Resulting encoding is 27*3 = 81 bits:
  000 110 001 000 010 110 000 110 011 100 000 010 110 000 110 101 000 010 000 100 110 011 000 010 000 001 000

Tree Representation

- We can think of binary codes as binary **tries**
- Each node can have a 0 (left) or a 1 child (right)

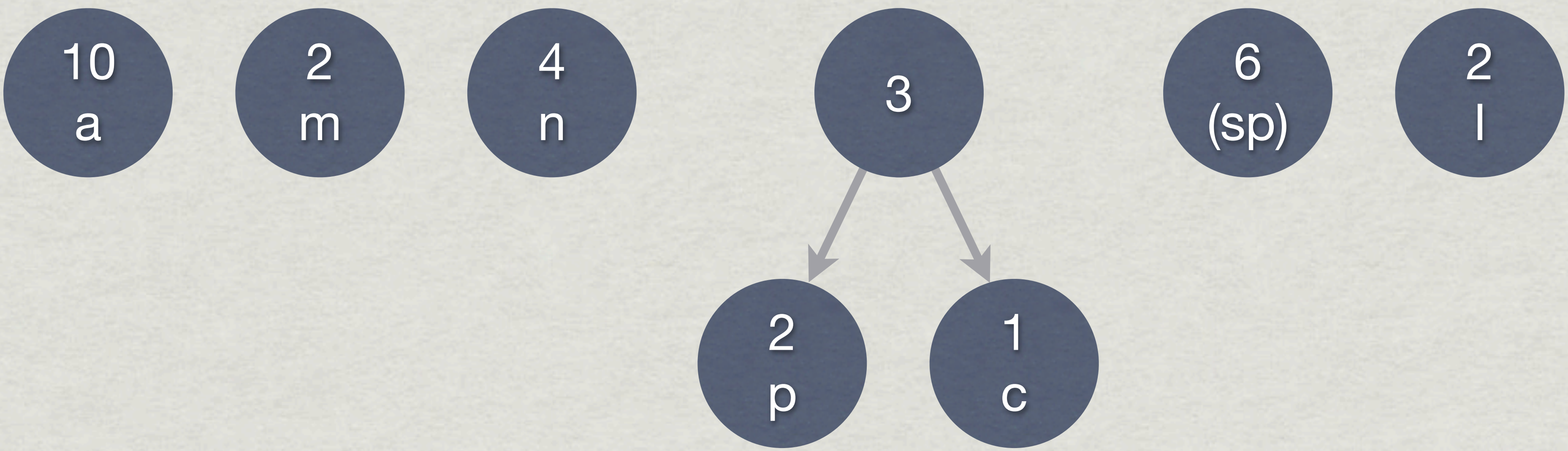


Huffman's Algorithm

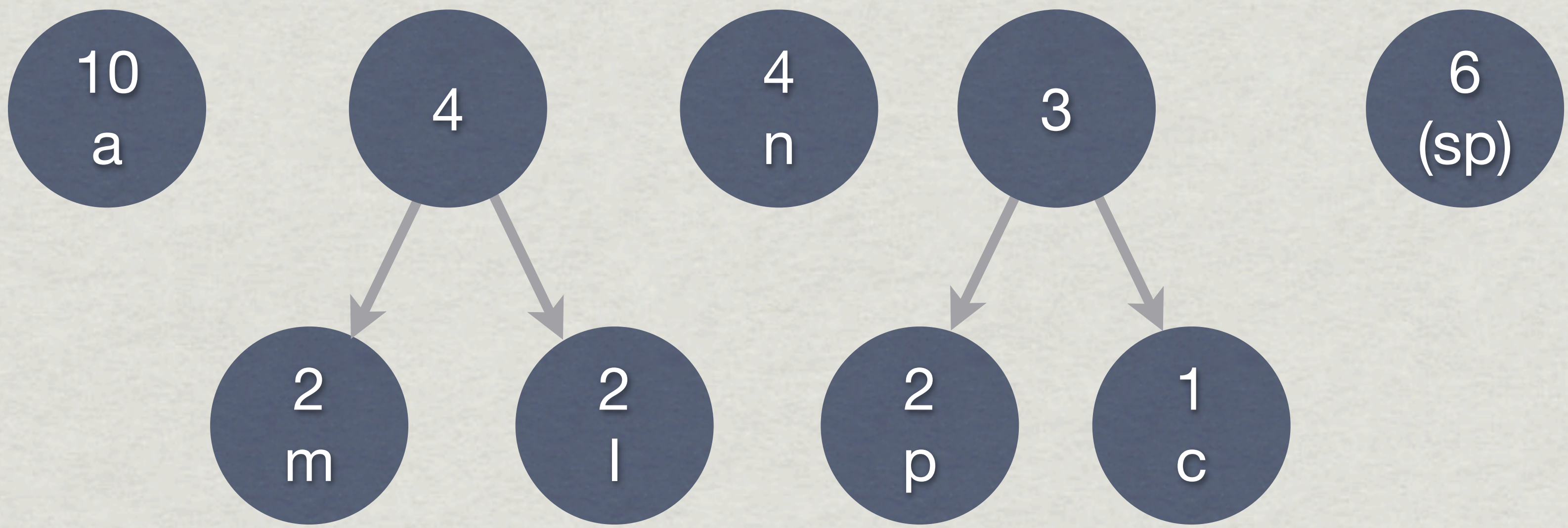
- Compute character frequencies:
  a 10, m 2, n 4, p 2, c 1, l 2, (space) 6
- Create forest of 1-node trees for all the characters.
- Let the **weight** of the trees be the sum of the frequencies of its leaves
- Repeat until forest is a single tree:
  Merge the two trees with minimum weight.
  Merging sums the weights.

Example

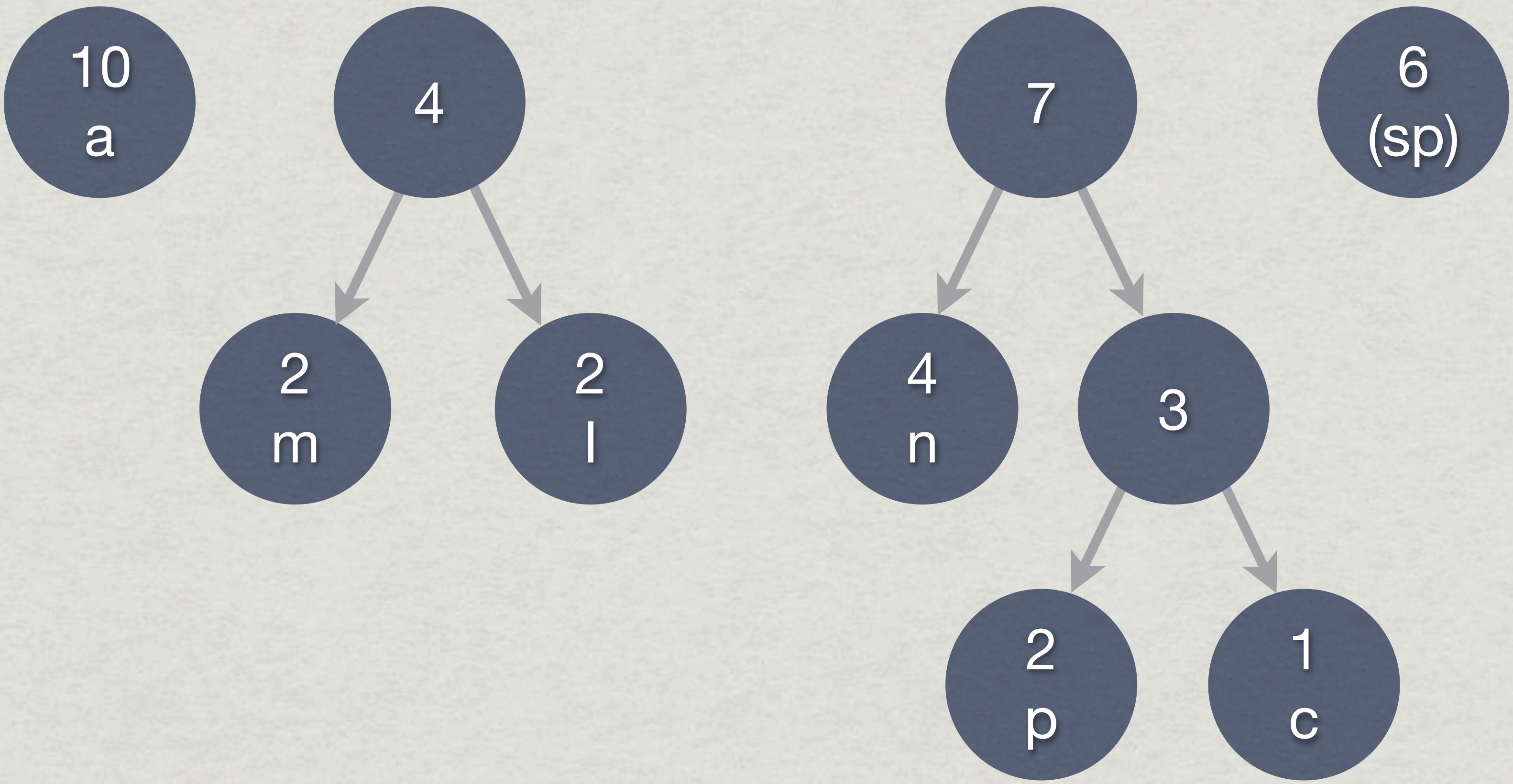
Example



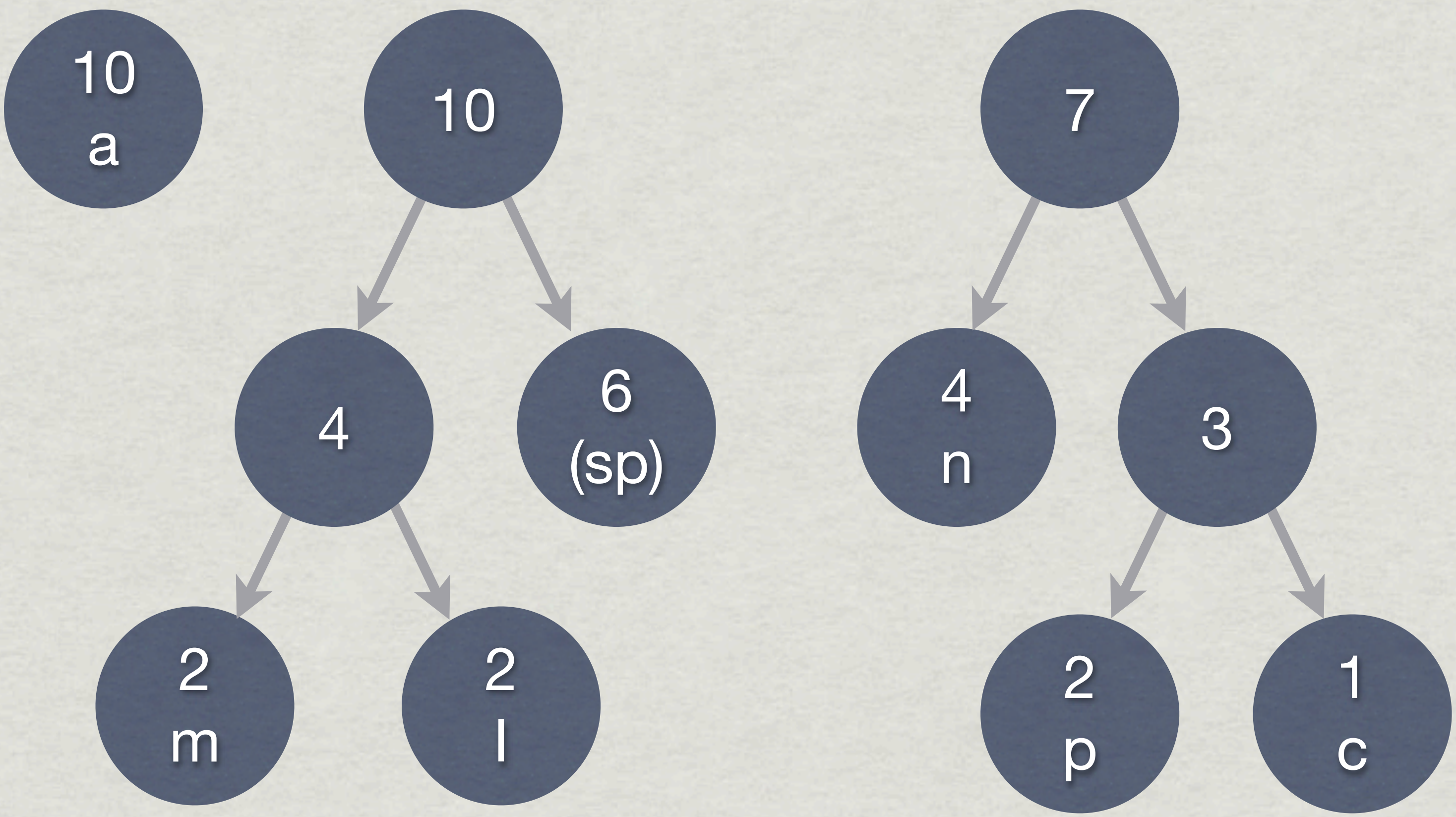
Example



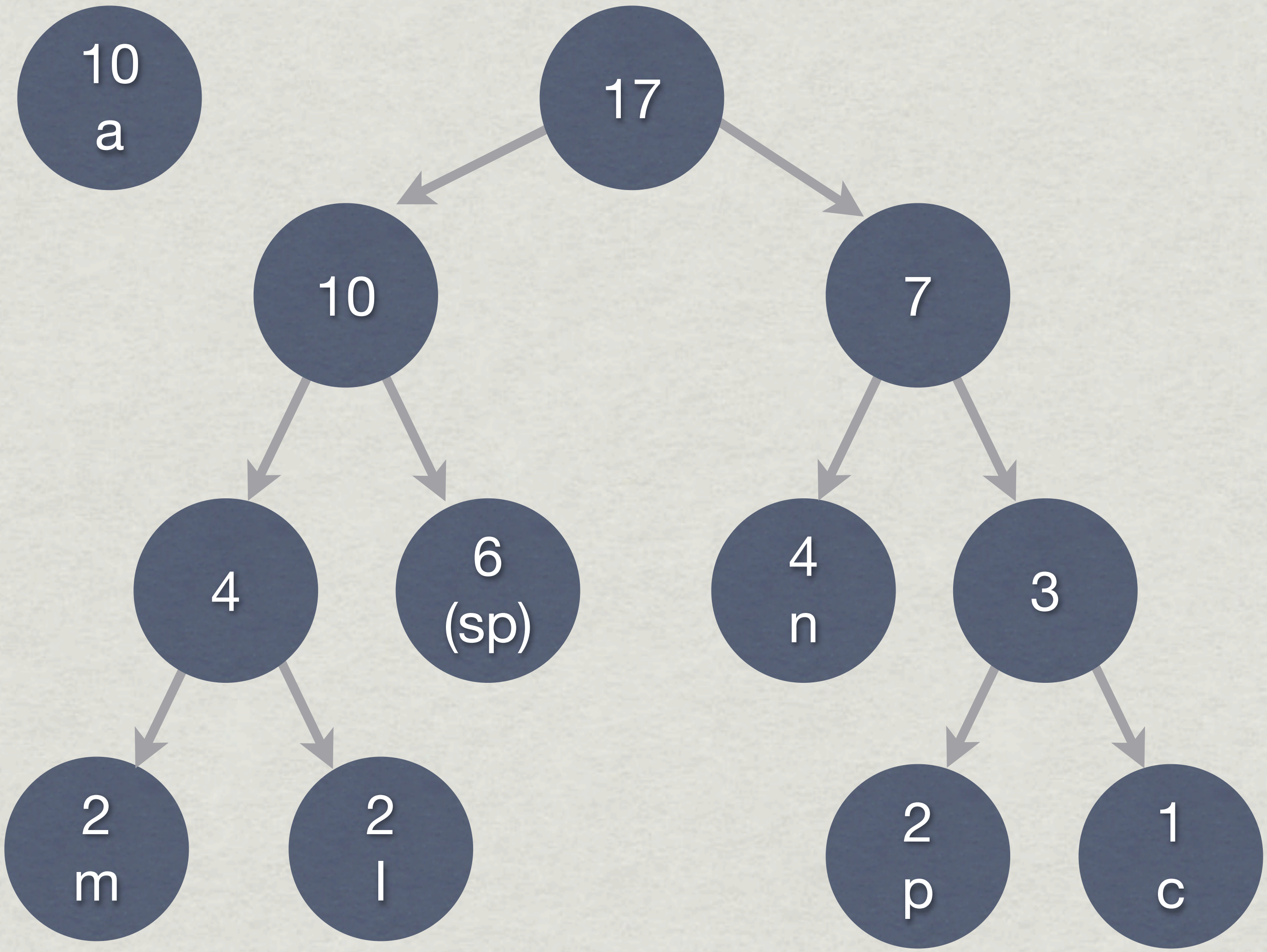
Example



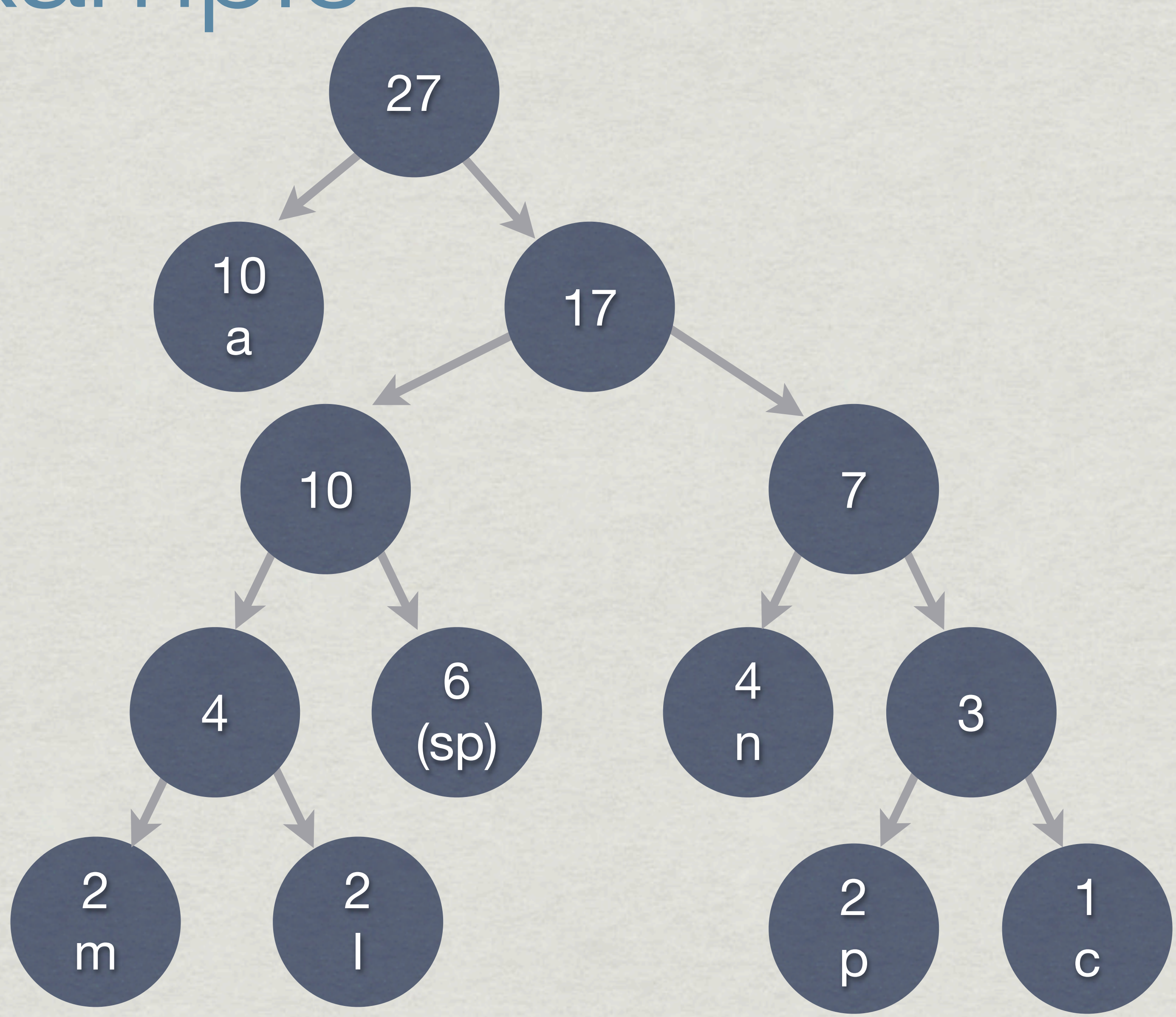
Example



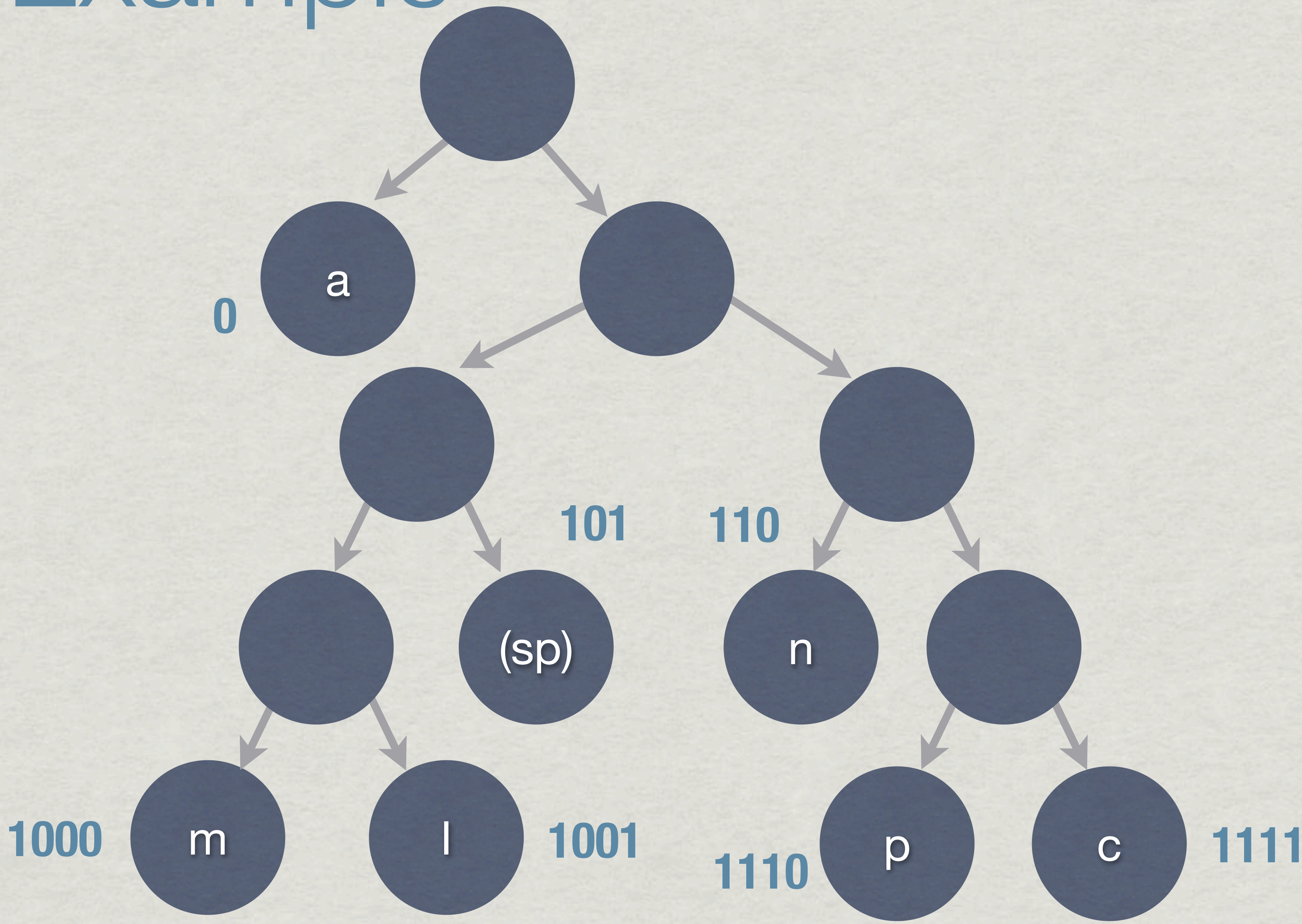
Example



Example



Example



Resulting Code

a	m	n	p	l	c	space
0	1000	110	1110	1001	1111	101

- “a man a plan a canal panama”
  0 101 1000 0 110 101 0 101 1110 1001 0 110 101
  0 101 1111 0 110 0 1001 101 1110 0 110 0 1000 0
- 68 bits

Huffman Details

- We can manage the forest with a priority queue:
- **buildHeap** first,
  - find the least weight trees with 2 **deleteMin**s,
  - after merging, **insert** back to heap.
- In practice, also have to store coding tree, but the payoff comes when we compress larger strings

Optimality of Huffman

- Induction: Suppose Huffman tree is optimal for **N** characters. What about **N+1** characters?
  - Lemma 1: Optimal tree is full
  - Lemma 2: the 2 least frequent characters are at the deepest level in optimal tree
  - Lemma 3: Swapping characters at same depth doesn’t affect optimality

Optimality of Huffman

- Induction: Suppose Huffman tree is optimal for **N** characters. What about **N+1** characters?
  - Lemma 1: Optimal tree is full
  - Lemma 2: the 2 least frequent characters are at the deepest level in optimal tree
  - Lemma 3: Swapping characters at same depth doesn’t affect optimality
- Lemma 4: An optimal tree exists where the least frequent characters are siblings at deepest level.

Optimality of Huffman

- number of bits of an encoding is $B(T) = \sum_{i=1}^{N+1} F_i D_i$
- F is the frequency of the character, D is the depth in the tree (the number of bits)
- Create new tree T* by removing least frequent chars and replacing with a meta-character whose frequency is the frequency of both chars,
  - meta-character is one level less deep

Optimality of Huffman

- $B(T) = B(T*) + F_1 + F_2$
- Proof by contradiction: Assume there is a different tree T' that is better than T

$$
\begin{aligned}
B(T') &< B(T) \\
B(T'*) + F_1 + F_2 &< B(T*) + F_1 + F_2 \\
B(T'*) &< B(T*)
\end{aligned}
$$

- That is a contradiction because T* has N characters, which means Huffman is optimal via our inductive hypothesis

Optimality of Huffman

- Assuming falseness of inductive **step** produced contradiction to inductive **hypothesis**
- Therefore, if Huffman codes are optimal for **N** characters, they are also for **N+1** characters
- Huffman is obviously optimal for 2 characters
- Huffman codes are optimal □

Reading

- Homework 4
- Weiss 10.1.2