Lecture 19: Huffman Codes and Priority Queues

8/08/22

berryl5@uw.edu

https://www.youtube.com/watch?v=5Cdv932eoU0
A8: Huffman

Note: This assignment is worth a total of 30 points. It is divided into two parts, each worth approximately half of the points. Please note that this assignment cannot be resubmitted and solutions to this homework will not be accepted after 11:59 pm on Friday, August 19th.

This assignment will assess your mastery of the following objectives:

- Implement a well-designed Java class to meet a given specification.
- Implement, manipulate, and traverse a binary tree.
- Implement the Comparable interface
- Follow prescribed conventions for code quality, documentation, and readability.
Cats and Codes

How many patterns can you make with 3 cats?
Cats and Codes

How many patterns can you make with 3 cats?

$2^3$
Cats and Codes

How many encoding can you make with 3 bits?

$2^3$
Herman Hollerith Story - Census
Hollerith Story - Punched Card

A pantograph used to create punch cards

Hollerith’s electronic tabulator, 1902.

More Info: https://www.census.gov/history/www/innovations/technology/the_hollerith_tabulator.html
American Standard Code for Information Interchange

128 characters
How many bits? 256
Ascii

American Standard Code for Information Interchange

128 characters
How many bits?

7
Unicode (UTF-8)
What can we do to compress some text?

- Different code sizes for different characters
- Differing frequency
  - Code for specific words
  - Encode symbols
Making a Huffman Code - Example

Text: babyyaxcab

Extracted Letters: c x yy aaa bbb
Making a Huffman Code

Extracted Letters:
c x yy aaa bbb

- Freq: 1, Letter: ‘c’
- Freq: 1, Letter: ‘x’
- Freq: 2, Letter: ‘y’
- Freq: 3, Letter: ‘a’
- Freq: 3, Letter: ‘b’
### Making a Huffman Code

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
</tbody>
</table>

**Extracted Letters:**
c x y y a a a b b b

**Freq: 1**
- Letter: ‘c’

**Freq: 2**
- Letter: ‘y’

**Freq: 3**
- Letter: ‘a’
- Letter: ‘b’
Making a Huffman Code

Extracted Letters: c x yy aaa bbb

Freq: 1
Letter: ‘c’

Freq: 1
Letter: ‘x’

Freq: 2
Letter: ‘y’

Freq: 3
Letter: ‘a’

Freq: 3
Letter: ‘b’
Making a Huffman Code

Extracted Letters:
- c (Freq: 1)
- x (Freq: 1)
- y (Freq: 2)
- a (Freq: 3)
- b (Freq: 3)

Diagram:
- Node: Letter 'c' (Freq: 1)
- Node: Letter 'x' (Freq: 1)
- Node: Letter 'y' (Freq: 2)
- Node: Letter 'a' (Freq: 3)
- Node: Letter 'b' (Freq: 3)

Diagram shows the structure of the Huffman code based on the frequencies of the letters.
Making a Huffman Code

Extracted Letters: c x yy aaa bbb
Making a HuffmanCode - Priority Queue

Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’
Freq: 2
Letter: ‘y’
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’

Priority Queue

A collection of ordered elements that provides fast access to the minimum (or maximum) element.

```
public class PriorityQueue<E> implements Queue<E>
{   
PriorityQueue<E>() constructs an empty queue
add(E value) adds value in sorted order to the queue
peek() returns minimum element in queue
remove() removes/returns minimum element in queue
size() returns the number of elements in queue

Queue<String> tas = new PriorityQueue<String>();
tas.add("Watson");
tas.add("Sherlock");
tas.remove(); // "Sherlock"
```
Making a Huffman Code

Extracted Letters: 
c x yy aaa bbb

Freq: 2  
Letter: ‘y’

Freq: 2  
Letter:

Freq: 3  
Letter: ‘a’

Freq: 3  
Letter: ‘b’

Freq: 1  
Letter: ‘c’

Freq: 1  
Letter: ‘x’
Making a Huffman Code

Extracted Letters:
c x yy aaa bbb

Diagram of Huffman Code:
- Letter: 'c' (Freq: 1)
- Letter: 'x' (Freq: 1)
- Letter: 'y' (Freq: 2)
- Letter: 'a' (Freq: 3)
- Letter: 'b' (Freq: 3)
Making a Huffman Code

Extracted Letters: c x yy aaa bbb

Freq: 3
Letter: ‘a’

Freq: 3
Letter: ‘b’

Freq: 2
Letter: ‘y’

Freq: 2
Letter: ‘x’

Freq: 1
Letter: ‘c’

Freq: 1
Letter: ‘x’
Making a Huffman Code

Extracted Letters:
c x yy aaa bbb

Freq: 1
Letter: 'c'

Freq: 1
Letter: 'x'

Freq: 2
Letter: 'y'

Freq: 2
Letter: 'y'

Freq: 3
Letter: 'a'

Freq: 3
Letter: 'b'

Freq: 4
Letter:
Making a Huffman Code

Extracted Letters: c x y y a a a b b b

Freq: 3
Letter: ‘a’

Freq: 3
Letter: ‘b’

Freq: 4
Letter:

Freq: 2
Letter: ‘y’

Freq: 1
Letter: ‘c’

Freq: 1
Letter: ‘x’
Making a Huffman Code

Extracted Letters: c x yy aaa bbb

Freq: 1
Letter: 'c'
Freq: 1
Letter: 'x'
Freq: 2
Letter: 'y'
Freq: 2
Letter: 'b'
Freq: 3
Letter: 'a'
Freq: 3
Letter: 'b'
Freq: 4
Letter:
Making a Huffman Code

Extracted Letters:
c x yy aaa bbb

Freq: 4
Letter:

Freq: 2
Letter: ‘y’

Freq: 1
Letter: ‘c’

Freq: 1
Letter: ‘x’

Freq: 3
Letter: ‘a’

Freq: 3
Letter: ‘b’

Freq: 2
Letter: •
Making a Huffman Code

Extracted Letters:
c x yy aaa bbb

Frequency
Letter

6
Freq: 6
Letter:

3
Freq: 3
Letter: ‘a’

3
Freq: 3
Letter: ‘b’

4
Freq: 4
Letter:

2
Freq: 2
Letter: ‘y’

2
Freq: 2
Letter:

1
Freq: 1
Letter: ‘c’

1
Freq: 1
Letter: ‘x’
Making a Huffman Code

Extracted Letters: c x yy aaa bbb
Making a Huffman Code

Extracted Letters:
c x yy aaa bbb
Making a Huffman Code

Extracted Letters:
- c
- x
- y
- y
- a
- a
- a
- b
- b
Decoding a Message

Binary Bits: 010101111
Decoding a Message

Binary Bits:
010101111

Output:
Decoding a Message

Binary Bits: 010101111
Output: c
Decoding a Message

Binary Bits:
010101111

Output:
ca
Decoding a Message

Binary Bits:
010101111

Output:
cab
Decoding a Message

Binary Bits: 01010111
Output: cabb
Decoding a Message

Binary Bits: 010101111
Output: cabb

BitInputStream: Binary Bits, not Ascii 1’s and 0’s
Decoding a Message

Binary Bits: 01010111 10000000
Output: cabbyyy?

Problem:
Can only write characters, which are 8 bits long
Solution:
Making a Huffman Code - Example with Pseudo EOF

Text: babyyacab

Extracted Letters: c yy aaa bbb
Making a Huffman Code With EOF

Extracted Letters: c yy aaa bbb

- Freq: 1
  Letter: ‘c’

- Freq: 1
  Letter: eof

- Freq: 2
  Letter: ‘y’

- Freq: 3
  Letter: ‘a’

- Freq: 3
  Letter: ‘b’
Making a Huffman Code With EOF

Extracted Letters: c yy aaa bbb
Making a Huffman Code With EOF

![Huffman Code Diagram]

Binary Bits:

01010111 10110000
Making a Huffman Code With EOF

Binary Bits:
01010111 10110000

Output:
C

Diagram:

- Root: Freq: 10, Letter:
  - Left: Freq: 4, Letter: 'y'
    - Left: Freq: 2, Letter: 'c'
    - Right: Freq: 2, Letter: 'a'
  - Right: Freq: 6, Letter:
    - Left: Freq: 3, Letter: 'a'
    - Right: Freq: 3, Letter: 'b'

Output path for 'c': 01010111
Making a Huffman Code With EOF

Binary Bits: 01010111 10110000
Output: ca
Making a Huffman Code With EOF

Binary Bits:
01010111 10110000

Output:
cab
Making a Huffman Code With EOF

Binary Bits: 01010111 10110000

Output: cabb
Making a Huffman Code With EOF

Binary Bits:
01010111 10110000

Output:
cabb
Making a Huffman Code With EOF

Binary Bits:
01010111 10110000

Output:
cabb

Extra 0’s don’t matter because we stop at eof
Saving Huffman Code

```
Freq: 10
Letter:

Freq: 4
Letter:

Freq: 2
Letter: ‘y’

Freq: 2
Letter: ‘a’

Freq: 1
Letter: ‘c’

Freq: 1
Letter: eof
```
Saving Huffman Code, .code file

```
Freq: 10
Letter:

Freq: 4
Letter:

Freq: 2
Letter: 'y'

Freq: 2
Letter: 'a'

Freq: 1
Letter: 'c'

Freq: 1
Letter: eof

Freq: 6
Letter:

Freq: 3
Letter: 'b'

Freq: 3
Letter: 'a'

Freq: 11
Letter: '

Freq: 0
Letter: '

Saving Huffman Code, .code file

```

taxi.code

```
121
00
99
010
255
011
97
10
98
11
```
Saving HuffmanCode, .code file

taxi.code

121 (y)
00
99 (c)
010
255 (eof)
011
97 (a)
10
98 (b)
11