CSCI 136
Data Structures &
Advanced Programming

Lecture 23
Spring 2018
Profs Bill & Jon
Last Time

- Binary Tree Traversals
- Binary Tree Iterators
- Array representation of trees
  - Node i’s children: 2i+1, 2i+2
  - Node i’s parent: (i-1)/2
  - Good for full or complete trees
  - Wasted space if tree is sparse or unbalanced
Today

- Breadth-First and Depth-First Search
- Application: Huffman Encoding
- Priority Queues
- Heaps
Recall from last class:

- **In-order**: “left, node, right”
- **Pre-order**: “node, left, right”
- **Post-order**: “left, right, node”
- **Level-order**: visit all nodes at depth $i$ before depth $i+1$
Traversals & Searching

- We can use traversals for searching unordered trees.
- How might we search a tree for a value?
  - **Breadth-First**: Explore nodes near the root before nodes far away (level order traversal)
    - Find the nearest gas station
  - **Depth-First**: Explore nodes deep in the tree first (post-order traversal)
    - Solution to a maze
      - Go as far as you can until you hit a dead end, then choose a different branch (Maze video)
## Next up: Huffman Codes

- Computers encode a text as a sequence of bits

### ASCII TABLE

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>67</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BEL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>[LINE FEED]</td>
<td>42</td>
<td>2A</td>
<td>+</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>*</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>]</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td>l</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>\</td>
<td>125</td>
<td>7D</td>
<td>{</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
</tbody>
</table>
Huffman Codes

• In ASCII: 1 character = 8 bits (1 byte)
  • Allows for $2^8 = 256$ different characters
• ‘A’ = 01000001, ‘B’ = 01000010
• Space to store “AN_ANTARCTIC_PENGUIN”
  • 20 characters -> 20*8 bits = 160 bits

• Is there a better way?
  • Only 11 symbols are used (ANTRCIPEGU_)
  • “ASCII-lite” only needs 4 bits per symbol (since $2^4 > 11$)
    • 20*4 = 80 bits instead of 160!

• Can we still do better??
Huffman Codes

• A Huffman code is an optimal prefix code for lossless compression
  • **Compression:** data is converted to a format that takes up less space than the original
  • **Lossless:** all of the information in the original data is preserved in the compressed version
  • **Prefix code:** a variable-length encoding where no codeword is a prefix of another codeword

• Our goal is to take a string and represent it using the smallest number of bits we can, without losing any information about the original string.
Huffman Codes

• Example
  • AN_ANTARCTIC_PENGUIN
  • Compute letter frequencies

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
<th>I</th>
<th>N</th>
<th>P</th>
<th>R</th>
<th>T</th>
<th>U</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

• Key Idea: Use fewer bits for most common letters

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>E</th>
<th>G</th>
<th>I</th>
<th>N</th>
<th>P</th>
<th>R</th>
<th>T</th>
<th>U</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>110</td>
<td>111</td>
<td>1011</td>
<td>1000</td>
<td>000</td>
<td>001</td>
<td>1001</td>
<td>1010</td>
<td>0101</td>
<td>0100</td>
<td>011</td>
</tr>
</tbody>
</table>

• Uses 67 bits to encode entire string
The Encoding Tree

Left = 0; Right = 1
Huffman Encoding Algorithm

Input: symbols of alphabet with frequencies

- Huffman encode algorithm is as follows:
  - Create a single-node tree for each symbol: key is frequency; weight is letter
  - while there is more than one tree:
    - Find two trees $T_1$ and $T_2$ with lowest weights
    - Merge them into new tree $T$ with:
      \[ T.\text{weight} = T_1.\text{weight} + T_2.\text{weight} \]
  - **Theorem**: The tree computed by Huffman is an optimal encoding for given frequencies
Demo

- To run the Huffman code demo found on course webpage:
  
  `java -jar huffman.jar`
The Encoding Tree (With Weights)

*Each node’s value is the sum of the frequencies of all its children

Left = 0; Right = 1
Implementing the Algorithm

- Keep a Vector of Binary Trees
- Sort them by decreasing frequency
  - Removing two smallest frequency trees is fast
- Insert merged tree into correct sorted location in Vector
- Running Time:
  - $O(n \log n)$ for initial sorting
  - $O(n^2)$ for while loop
- Can we do better...?
What Huffman Encoder Needs

- A structure $S$ to hold items with *priorities*
- $S$ should support operations
  - `add(E item);` // add an item
  - `E removeMin();` // remove min priority item
- $S$ should be designed to make these two operations fast
- If, say, they both ran in $O(\log n)$ time, the Huffman while loop would take $O(n \log n)$ time instead of $O(n^2)$!
Priority Queues

- Name is misleading: They are not FIFO
- Always dequeue object with highest priority (smallest rank) regardless of when it was enqueued
- Data can be received/inserted in any order, but it is always returned/removed according to priority
- Like ordered structures (i.e., OrderedVectors and OrderedLists), PQs require comparisons of values
Priority Queues

Priority queues are also used for:

- Scheduling processes in an operating system
  - Priority is function of time lost + process priority
- Order services on server
  - Backup is low priority, so don’t do when high priority tasks need to happen
- Scheduling future events in a simulation
- Medical waiting room
- Huffman codes - order by tree size/weight
- A variety of graph/network algorithms
- To roughly rank choices that are generated out of order
An Apology

• On behalf of computer scientists everywhere, we’d like to apologize for the confusion that inevitably results from the fact that:

  Higher Priority == Lower Rank

• The PQ removes the lowest ranked value in an ordering: that is, the highest priority value!

We’re sorry!
PQ Interface

```java
public interface PriorityQueue<E extends Comparable<E>> {
    public E getFirst(); // peeks at minimum element
    public E remove();   // removes + returns min element
    public void add(E value); // adds an element
    public boolean isEmpty();
    public int size();
    public void clear();
}
```
Notes on PQ Interface

• Unlike previous structures, we do not extend any other interfaces
  • Many reasons: For example, it’s not clear that there’s an obvious iteration order
• PriorityQueue stores Comparables: methods consume Comparable parameters and return Comparable values
  • Could be made to use Comparators instead…
Implementing PQs

• Queue?
  • Wouldn’t work so well because we can’t insert and remove in the “right” way (i.e., keeping things ordered)

• OrderedVector?
  • Like a normal Vector, but no add(int i)
    • Instead, add(Object o) places o at proper location according to the ordering of all objects in the Vector
  • O(n) to add/remove from vector
  • Details in book…
  • Can we do better than O(n)?

• Heap!
  • Partially ordered binary tree