CompSci 201, L20: Binary Heaps
Logistics, Coming up

• APT9 (more tree problems) due this Wed., 11/9

• P5 Huffman due next Monday 11/14

• Wrapping up binary heaps and balanced binary search trees this week, then on to graphs!
People in CS: Clarence “Skip” Ellis

- Born 1943 in Chicago. PhD in CS from U. Illinois UC in 1969
  - First African American anywhere in US to complete a PhD in CS
- Founding member of the CS department at U. Colorado, also worked in industry.
  - Developing original graphical user interfaces, object-oriented programming, collaboration tools.

“People put together an image of what I was supposed to be,” he recalled. “So I always tell my students to push.”

Read more here
Huffman Compression

Representing data with bits: Preferably fewer bits

- Zip
- Unicode
- JPEG
- MP3

Huffman compression used in all of these and more!
Prefix property encoding as a tree

Convention: 0 for left and 1 for right

Values you want to encode are leaves: Ensures prefix property.

Values deeper in tree encoded with more bits than those earlier in the tree.

Encoding is the sequence of 0’s and 1’s on root to leaf path

<table>
<thead>
<tr>
<th>char</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>'g'</td>
<td>10</td>
</tr>
<tr>
<td>'o'</td>
<td>11</td>
</tr>
<tr>
<td>'p'</td>
<td>0100</td>
</tr>
<tr>
<td>'h'</td>
<td>0101</td>
</tr>
<tr>
<td>'e'</td>
<td>0110</td>
</tr>
<tr>
<td>'r'</td>
<td>0111</td>
</tr>
<tr>
<td>'s'</td>
<td>000</td>
</tr>
<tr>
<td>' '</td>
<td>001</td>
</tr>
</tbody>
</table>
Huffman Coding

• Greedy algorithm for building an optimal variable length encoding tree.

• High level idea:
  • Start with the leaves/values you want to encode with weights = frequency.
  • Iteratively choose the lowest weight nodes to “connect up” to a new node with weight = sum of children.
Encoding the text “go go gophers”

<table>
<thead>
<tr>
<th>char</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>'g'</td>
<td>00</td>
</tr>
<tr>
<td>'o'</td>
<td>01</td>
</tr>
<tr>
<td>'p'</td>
<td>1110</td>
</tr>
<tr>
<td>'h'</td>
<td>1101</td>
</tr>
<tr>
<td>'e'</td>
<td>101</td>
</tr>
<tr>
<td>'r'</td>
<td>1111</td>
</tr>
<tr>
<td>'s'</td>
<td>1100</td>
</tr>
<tr>
<td>'r'</td>
<td>100</td>
</tr>
</tbody>
</table>
WOTO

Go to duke.is/pge7f

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
P5 Outline

1. Write Decompress first
   • Takes a compressed file (we give you some)
   • Reads Huffman tree from bits
   • Uses tree to decode bits to text

2. Write Compress second
   • Count frequencies of values/characters
   • Greedy algorithm to build Huffman tree
   • Save tree and file encoded as bits
Heaps Revisited
java.util.PriorityQueue Class

- Kept in sorted order, smallest out first
- Objects must be Comparable OR provide Comparator to priority queue

```java
PriorityQueue<String> pq = new PriorityQueue<>();
pq.add("is");
pq.add("CompSci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
CompSci 201
is
wonderful
```

```java
PriorityQueue<String> pq = new PriorityQueue<>((
    Comparator.comparing(String::length));
pq.add("is");
pq.add("CompSci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
is
wonderful
CompSci 201
```
Tradeoffs, Heaps, and Trees

• Fast add and remove?
• Binary Heap: Implements a priority queue with:
  • Peek: $O(1)$
  • Remove: $O(\log(N))$
  • Add: $O(\log(N))$
• `java.util.PriorityQueue` is implemented as a binary heap

<table>
<thead>
<tr>
<th>N</th>
<th>$\log_2(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>6.6</td>
</tr>
<tr>
<td>200</td>
<td>7.6</td>
</tr>
<tr>
<td>400</td>
<td>8.6</td>
</tr>
<tr>
<td>800</td>
<td>9.6</td>
</tr>
<tr>
<td>1600</td>
<td>10.6</td>
</tr>
<tr>
<td>3200</td>
<td>11.6</td>
</tr>
<tr>
<td>6400</td>
<td>12.6</td>
</tr>
<tr>
<td>12800</td>
<td>13.6</td>
</tr>
<tr>
<td>25600</td>
<td>14.6</td>
</tr>
<tr>
<td>51200</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Binary Heap at a high level

• Maintain the **heap property** *that every node is less than or equal to its successors*, and

• The **shape property** *that the tree is complete* (full except perhaps last level, fill from left to right)

**Operations:**

• Peek: Return value of root node

• Remove: Remove root node and fix tree to reestablish properties.

• Add: Insert at first open position, fix to reestablish properties.
How are PriorityQueues implemented?

• Normally think about binary tree used for implementing priority queues, supports:
  • insert/add
  • findMin/peek
  • deleteMin/remove

• Actually implement with an array
  • minimizes storage (no explicit points/nodes)
  • simpler to code, no explicit tree traversal
  • faster too (constant factor, not asymptotically)---children are located by index/position in array
Aside: How much less memory?

• Storing an int takes 4 bytes = 32 bits on most machines.

• Storing one *reference to an object* (a memory location) takes 8 bytes = 64 bits on most machines.

• For a heap storing N integers...
  • Array of N integers takes ~ 4N bytes.
  • Binary tree where each node has an int, left, and right reference takes ~20N bytes.
  • So maybe a 5x savings in memory (just an estimate). Not an asymptotic improvement.
Using an array for a Heap

- Hard to keep track of the last node in the tree.
- Index positions in the tree level by level, left to right:
  - Depth 0
  - Depth 1
  - Depth 2
  - Depth 3

- Last node in the heap is always just the largest index
- Can use indices to represent as an array!

<table>
<thead>
<tr>
<th>6</th>
<th>10</th>
<th>7</th>
<th>17</th>
<th>13</th>
<th>9</th>
<th>21</th>
<th>19</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

(ArrayList if you want it to be growable)
Properties of the Heap Array

- Store “node values” in array beginning at index 1
  - Could 0-index, Zybook does this
- Last node is always at the max index
- Minimum node is always at index 1
- peek is easy, return first value.
  - How about add?
  - Remove?
Relating Nodes in Heap Array

- When 1-indexing: For node with index \(k\)
  - left child: index \(2k\)
  - right child: index \(2k+1\)
  - parent: index \(k/2\)

- Why? Follows from:
  - Heap is complete, and
  - Complete binary tree has \(2^d\) nodes at depth \(d\) (except last)
Adding values to heap in pictures

- Add to first open position in last level of the tree
  - (really, add to end of array)
- Swap with parent if heap property violated
  - stop when parent is smaller
  - Or you reach the root

Heap property re-established
Heap add implementation

```java
public void add(Integer value) {
    heap.add(value); // add to last position
    size++;

    int index = size; // note we are 1-indexing
    int parent = index / 2;

    while (parent >= 1 && heap.get(parent) > heap.get(index)) {
        swap(index, parent);
        index = parent;
        parent /= 2;
    }
}
```

Index 0: 6
Index 1: 10
Index 2: 7
Index 3: 17
Index 4: 13
Index 5: 9
Index 6: 21
Index 7: 19
Index 8: 25
Index 9: 8

parent=5
index=10
Heap add implementation

```java
public void add(Integer value) {
    heap.add(value); // add to last position
    size++;

    int index = size; // note we are 1-indexing
    int parent = index / 2;

    while (parent >= 1 && heap.get(parent) > heap.get(index)) {
        swap(index, parent);
        index = parent;
        parent /= 2;
    }
}
```

**ArrayList<Integer> heap**

**parent=2**

**index=5**
Heap add implementation

```java
public void add(Integer value) {
    heap.add(value); // add to last position
    size++;
    int index = size; // note we are 1-indexing
    int parent = index / 2;
    while(parent >= 1 && heap.get(parent) > heap.get(index)) {
        swap(index, parent);
        index = parent;
        parent /= 2;
    }
}
```

11/7/22
Compsci 201, Fall 2022, L20: Binary Heaps
Heap remove in pictures

• Always return root value

• Replace root with last node in the heap

• While heap property violated, swap with smaller child.
Heap Complexity

• Claimed that:
  • Peek: O(1)
  • Add: O(log(N))
  • Remove: O(log(N))

• On a heap with N values. Why?
  • Peek: Easy, return first value in an Array
  • Complete binary tree always has height O(log(N)).
  • add and remove “traverse” one root-leaf path, at most O(log(N)).
WOTO

Go to duke.is/guzv2

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
Choose your own adventure

1. Look at Generic DIYBinaryHeap: coursework.cs.duke.edu/cs-201-fall-22/diybinaryheap, or

2. Solve Greedy Coding Problem with PriorityQueue: leetcode.com/problems/non-overlapping-intervals/

Live coding