Intro to (Binary) Heaps

- So far, the binary trees we've seen are linked structures
  - (Like a linked list, but each node has two references)

- Turns out, binary trees can also be implemented using **arrays**
  - One popular array-based binary tree is called a **heap**
  - Advantage: Heaps are always balanced!
  - Disadvantage:
    - Array-based, so need to re-allocate sometimes
    - Can waste space with unused array slots
Binary Heap Organization

- Binary Heap organization:
  - Root node is at index $i=0$
  - For any node at index $i$
    - Its left child is at index: $i*2+1$
    - Its right child is at index: $i*2+2$
    - Its parent node is at index: $??$

- What is stored:

- What it represents:
Binary Heap organization:

- Root node is at index $i=0$
- For any node at index $i$
  - Its left child is at index: $i*2+1$
  - Its right child is at index: $i*2+2$
  - Its parent node is at index: ???

What is stored:

```
10  25  50  40  56  59  95
0  1  2  3  4  5  6
```

What it represents:

```
        10
       / \   \
      25   50
     /|   /|  \
    40 56 59 95
```

CSCI 261: Computer Science II - 8- Heaps
Binary Heap Organization

- Binary Heap organization:
  - Root node is at index $i=0$
  - For any node at index $i$
    - Its left child is at index: $i*2+1$
    - Its right child is at index: $i*2+2$
    - Its parent node is at index: $(i-1)/2$

- What is stored:
  - 10
  - 25
  - 50
  - 40
  - 56
  - 59
  - 95

- What it represents:
Outline

- (Binary) Heaps
- Min-Heap
  - Properties
  - Add
    - Percolate up
  - Remove
    - Percolate down
- Performance and Applications
- Conclusion
One popular organization is called a *Min-Heap*

Important: Min-Heap Properties (Recursive)

- The value at the root is the smallest in the tree
- Every subtree is a min-heap

Note that Min-Heap's properties are more relaxed than BST's!
One popular organization is called a **Min-Heap**

**Important: Min-Heap Properties (Recursive)**

- The value at the root is the smallest in the tree
- Every subtree is a min-heap

Note that Min-Heap's properties are more relaxed than BST's!
Min-Heap

- One popular organization is called a Min-Heap
- Important: Min-Heap Properties (Recursive)
  - The value at the root is the smallest in the tree
  - Every subtree is a min-heap

- Note that Min-Heap's properties are more relaxed than BST's!
One popular organization is called a **Min-Heap**

**Important: Min-Heap Properties (Recursive)**
- The value at the root is the smallest in the tree
- Every subtree is a min-heap

Note that Min-Heap's properties are more relaxed than BST's!
One popular organization is called a **Min-Heap**

**Important: Min-Heap Properties (Recursive)**
- The value at the root is the smallest in the tree
- Every subtree is a min-heap

```
  10
 /  \
25   50
 /  \
40   77
```

Note that Min-Heap's properties are more relaxed than BST's!
Min-Heaps or Non-Heaps?

```
    5
   / \
  14  23
 /   /\   /\  
20 16 48 62
 / \          / \  
71 53         71 53
```

Min-Heaps or Non-Heaps?

```
5
/  \
14  23
/  \
12  26  34  20
/  \
24  35
```
Min-Heaps or Non-Heaps?

```
5
/  \
14  23
/  \
32 41
   /  \
   50 64 53

```
Outline

- (Binary) Heaps
- Min-Heap
  - Properties
  - Add
    - Percolate up
  - Remove
    - Percolate down
- Performance and Applications
- Conclusion
Adding Items

- Add the following items: 43, 18, 2
Adding Items

- Add the following items: 43, 18, 2

![Binary Heap Diagram]

- The heap structure is visualized with numbers in nodes and arrows indicating the parent-child relationships.
Adding Items

- Add the following items: 43, 18, 2
Adding Items

- Add the following items: 43, 18, 2
Adding Items

- Add the following items: 43, 18, 2
Adding Items

- Add the following items: 43, 18, 2
Adding Items

- Add the following items: 43, 18, 2

Percolate 18 up!

Swap!
Adding Items

» Add the following items: 43, 18, 2

Percolate 18 up!
Add the following items: 43, 18, 2

Percolate 18 up!
Adding Items

- Add the following items: 43, 18, 2

Percolate 18 up!
Adding Items

- Add the following items: 43, 18, 2

Percolate 18 up!
Adding Items

- Add the following items: 43, 18, 2
Adding Items

- Add the following items: 43, 18, 2

Diagram showing a heap structure with nodes containing numbers: 50, 64, 53, 43, 87, 23, 90, 5, 32, 41, 14, 18, and 0.

Table below the diagram:

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>14</th>
<th>18</th>
<th>32</th>
<th>41</th>
<th>23</th>
<th>90</th>
<th>50</th>
<th>64</th>
<th>53</th>
<th>43</th>
<th>87</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>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
Adding Items

- Add the following items: 43, 18, 2
Your turn! Add: 2

- Add the following items: 43, 18, 2
Implementation of Min-Heap

- Explore the Heap code (given)
  - Need to compare heap's stored data for percolate up/down...

```java
/**
   * Heaps: A lot like ArrayList under the hood
   */
public class Heap<E extends Comparable<E>> {
    private final int INITIAL_CAPACITY = 10;
    private E[] the_data;
    private int capacity;
    private int size;

    /**
     * Creates an empty heap of default capacity of 10
     */
    public Heap() {
        capacity = INITIAL_CAPACITY;
        size = 0;
        the_data = (E[]) new Comparable[capacity];
    }
```
Implementation: add(E item)

```java
/**
 * Adds an item to the heap
 * @param item The element to add to the heap
 */
public void add(E item) {
    if (size == capacity) { // out of space; double capacity
        reallocate();
    }
    the_data[size] = item; // add element to the end of the array
    percolateUp(size); // possibly need to percolate item up the heap
    size++;
}

private void percolateUp(int i) {
    int parent = (i-1)/2;
    if (parent >= 0) { // if parent exists...
        // this item is bigger than parent; swap with parent!
        if (the_data[i].compareTo(the_data[parent]) < 0) {
            E tmp = the_data[i];
            the_data[i] = the_data[parent];
            the_data[parent] = tmp; // swap with parent!
            percolateUp(parent);
        }
    }
}
```
Outline

- (Binary) Heaps
- Min-Heap
  - Properties
  - Add
    - Percolate up
  - Remove
    - Percolate down
- Performance and Applications
- Conclusion
Removing Items

- Remove the following items: 53, 14, 5
Removing Items

- Remove the following items: 53, 14, 5
Removing Items

- Remove the following items: 53, 14, 5
Removing Items

- Remove the following items: 53, 14, 5

![Binary Heap Diagram]

Replace

last node
Removing Items

- Remove the following items: 53, 14, 5
Removing Items

- Remove the following items: **53, 14, 5**

*Min-heap property satisfied! (Done)*
Removing Items

- Remove the following items: 53, 14, 5
Removing Items

- Remove the following items: 53, 14, 5

Min-heap property violated!
(Percolate 43 down!)
Removing Items

- Remove the following items: 53, 14, 5

Min-heap property violated!
(Percollate 43 down!)
Removing Items

- Remove the following items: **53, 14, 5**

Min-heap property violated!
(Percolate 43 down!)

Swap with smaller child
Removing Items

- Remove the following items: 53, 14, 5

Min-heap property violated!
(Percolate 43 down!)

No one to swap with
(done!)
Your Turn! Remove 5

- Remove the following items: 53, 14, 5
Your Turn! Remove 5

- Remove the following items: 53, 14, 5

```
5 32 18 43 41 23 90 50 64 87
```

Diagram representation: (Diagram of a heap with node values and structure)
Heap after remove(5)

- Remove the following items: 53, 14, 5
Outline

- (Binary) Heaps
- Min-Heap
  - Properties
  - Add
    - Percolate up
  - Remove
    - Percolate down
- Performance and Applications
- Conclusion
Observe: A Heap is a balanced binary tree!

- Percolating up and down are the time-dominant methods
  - Worst case: they just traverse all levels of the heap!

<table>
<thead>
<tr>
<th>Operations</th>
<th>ArrayList</th>
<th>LinkedList</th>
<th>BST</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>add an item</td>
<td>$O(n)$ avg/worst</td>
<td>$O(n)$ avg/worst</td>
<td>$O(h)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>contains an item</td>
<td>$O(n)$ avg/worst</td>
<td>$O(n)$ avg/worst</td>
<td>$O(h)$</td>
<td>$O(n)$ - why? **</td>
</tr>
<tr>
<td>remove an item</td>
<td>$O(n)$ avg/worst</td>
<td>$O(n)$ avg/worst</td>
<td>$O(h)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

** Heap doesn't help you search. An item could be down either left or right subtree.**
What good is a Heap if it doesn't help you search quickly?

- Min-Heap guarantee: smallest element always at top (root) of the heap
  - Takes $O(1)$ time to identify... just return: `the_data[0]`

These are the efficient (viable) operations...

- What can we do with them?

<table>
<thead>
<tr>
<th>Operations</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>add an item</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>contains an item</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>remove an item</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>find smallest item</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
One Application: HeapSort

- HeapSort is one interesting use of heaps...
  - Exploit the fact that it takes $O(1)$ to find min, $O(\log n)$ to remove it

Algorithm:

```java
/**
 * HeapSort!
 * @param list a list of ints
 * @return a sorted list of ints
 */
public static int[] heapSort(int[] list) {
    Heap<Integer> heap = new Heap<>();

    // build min-heap -- add all items from the list
    for (int i = 0; i < list.length; i++) {
        heap.add(list[i]);
    }

    // re-build the list by continuously pulling the min from the heap
    for (int i = 0; i < list.length; i++) {
        list[i] = heap.remove(0);
    }
    return list;
}
```
Another App: Java's PriorityQueue<E> Class

- Most priority queue implementations are heap based?
- Don't care how items are ordered
  - Only care that the smallest item is at the head of the queue
  - `offer() = add(), poll() = remove(0), peek() = get(0)`

<table>
<thead>
<tr>
<th>Signature</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public boolean offer(E item)</code></td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><code>public E remove()</code></td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><code>public E poll()</code></td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><code>public E peek()</code></td>
<td>$O(1)$</td>
</tr>
<tr>
<td><code>public E element()</code></td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Outline

- (Binary) Heaps
- Min-Heap
  - Properties
  - Add
    - Percolate up
  - Remove
    - Percolate down
- Performance and Applications
- Conclusion
Hwk 5 posted
  • Due Mon, 4/16
  • Find your partner

New slides up: Lec 8 Heaps

Last time...
  • Linked trees:
    - BinaryTree: good for modeling hierarchical structures
    - BST: good for performance if tree is balanced
  • Today: (Binary) Heaps
    - Array-based binary tree -- managed like an ArrayList (allowed to grow)
    - No BST properties, but something like it
    - Always balanced -- but what's the tradeoff? (Space)