Heaps & Other Trees

Heap

- A min-heap is a binary tree such that
  - the data contained in each node is less than (or equal to) the data in that node’s children.
  - the binary tree is complete

- A max-heap is a binary tree such that
  - the data contained in each node is greater than (or equal to) the data in that node’s children.
  - the binary tree is complete
Is it a min-heap?

Is it a min-heap?
Is it a min-heap?

Using heaps

What are min-heaps good for?
(What operation is extremely fast when using a min-heap?)

The difference in level between any two leaves in a heap is at most what?
Storage of a heap

- Use an array to hold the data.
- Store the root in position 1.
  - We won’t use index 0 for this implementation.
- For any node in position $i$,
  - its left child (if any) is in position $2i$
  - its right child (if any) is in position $2i + 1$
  - its parent (if any) is in position $\lfloor i/2 \rfloor$

(use integer division)
Inserting into a min-heap

- Place the new element in the next available position in the array.
- Compare the new element with its parent. If the new element is smaller, than swap it with its parent.
- Continue this process until either
  - the new element’s parent is smaller than or equal to the new element, or
  - the new element reaches the root (index 0 of the array)

Insert 43

```
5
  14
    32
     50
    64
    53
41
  87
  90
  23
43
```
Inserting into a min-heap
Insert 18

Inserting into a min-heap
Insert 2
Removing from a heap

- Place the root element in a variable to return later.
- Remove the last element in the deepest level and move it to the root.
- While the moved element has a value greater than at least one of its children, swap this value with the smaller-valued child.
- Return the original root that was saved.

Removing from a min-heap

Remove min

```
14
32 23
50 41 87 90
53 64 53
```

returnValue 5
Removing from a min-heap

Remove min

![Min-Heap Diagram]

```
returnValue  14
```

Efficiency of heaps

Assume the heap has N nodes.
Then the heap has \( \lceil \log_2(N+1) \rceil \) levels.

- **Insert**
  Since the insert swaps at most once per level, the order of complexity of insert is \( O(\log N) \)

- **Remove**
  Since the remove swaps at most once per level, the order of complexity of remove is also \( O(\log N) \)
Priority Queues

- A priority queue PQ is like an ordinary queue except that we can only remove the “maximum” element at any given time (not the “front” element necessarily).
- If we use an array to implement a PQ, enqueue is $O(\_\_\_\_\_\_\_)$ dequeue is $O(\_\_\_\_\_\_\_\_)$
- If we use a sorted array to implement a PQ enqueue is $O(\_\_\_\_\_)$ dequeue is $O(\_\_\_\_\_\_\_)$
- If we use a max-heap to implement a PQ enqueue is $O(\_\_\_\_\_)$ dequeue is $O(\_\_\_\_\_\_\_)$

General Trees

- A general tree consists of nodes that can have any number of children.
- Implementation using a binary tree:

  Each node has 2 fields: firstChild, nextSibling
Balanced Trees

- Binary search trees can become quite unbalanced, with some branches being much longer than others.
  - Searches can become O(n) operations
- These variants allow for searching while keeping the tree (nearly) balanced:
  - 2-3-4 trees
  - Red-black trees

2-3-4-trees

- A 2-3-4 Tree is a tree in which each internal node (nonleaf) has two, three, or four children, and all leaves are at the same depth.
  - A node with 2 children is called a "2-node".
  - A node with 3 children is called a "3-node".
  - A node with 4 children is called a "4-node".
Sample 2-3-4-tree

```
Sample 2-3-4-tree

10 15 20  40  60  80  90
10 15 20
```

Insert 100:

```
30 50 70
30 70
10 15 20 40 60 80 90 100
```

**Red-Black Trees**

- A red-black tree has the advantages of a 2-3-4 tree but requires less storage.
- Red-black tree rules:
  - Every node is colored either red or black.
  - The root is black.
  - If a node is red, its children must be black.
  - Every path from a node to a null link must contain the same number of black nodes.
2-3-4 Trees vs. Red-Black Trees

"2-node" in a 2-3-4 tree

<table>
<thead>
<tr>
<th>x</th>
<th>&lt;x</th>
<th>&gt;x</th>
</tr>
</thead>
</table>

equivalent red-black tree configuration

"3-node" in a 2-3-4 tree

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>&lt;x</th>
<th>&gt;x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>&lt;y</td>
<td>&gt;y</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>&lt;y</td>
<td>&gt;y</td>
</tr>
</tbody>
</table>

or

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>&lt;x</th>
<th>&gt;y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>&lt;y</td>
<td>&gt;y</td>
</tr>
</tbody>
</table>

equivalent red-black tree configurations
2-3-4 Trees vs. Red-Black Trees

"4-node" in a 2-3-4 tree

Sample Red-Black Tree

Original 2-3-4 tree:

Equivalent red-black tree:
Rotation

• Insert 85:

Rotation (cont'd)

See textbook for additional cases where rotation is required.
Additional Self-Balancing Trees

- AVL Trees
- 2-3 Trees
- B-Trees
- Splay Trees
  - (co-invented by Prof. Danny Sleator at CMU)