Heaps

02-713
Slides by Carl Kingsford
Priority Queue ADT

- Efficiently support the following operations on a set of keys:
  - findmin: return the smallest key
  - deletemin: return the smallest key & delete it
  - insert: add a new key to the set
  - delete: delete an arbitrary key

- Would like to be able to do findmin faster (say in time independent of the # of items in the set).
Job Scheduling: UNIX process priorities

PRI COMM
14 /System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/Versions/A/Support/mdworker
31 -bash
31 /Applications/iTunes.app/Contents/Resources/iTunesHelper.app/Contents/MacOS/iTunesHelper
31 /System/Library/CoreServices/Dock.app/Contents/MacOS/Dock
31 /System/Library/CoreServices/FileSyncAgent.app/Contents/MacOS/FileSyncAgent
31 /System/Library/CoreServices/RemoteManagement/AppleVNCServer.bundle/Contents/MacOS/AppleVNCServer
31 /System/Library/CoreServices/RemoteManagement/AppleVNCServer.bundle/Contents/Support/RFBRegisterMDNS
31 /System/Library/CoreServices/RemoteManagement/AppleVNCServer.bundle/Contents/Support/VNCPrivilegeProxy
31 /System/Library/CoreServices/Spotlight.app/Contents/MacOS/Spotlight
31 /System/Library/CoreServices/coreServicesd
...
31 /System/Library/PrivateFrameworks/MobileDevice.framework/Versions/A/Resources/usbmuxd
31 /System/Library/Services/AppleSpell.service/Contents/MacOS/AppleSpell
31 /sbin/launchd
31 /sbin/launchd
31 /usr/bin/ssh-agent
31 /usr/libexec/ApplicationFirewall/socketfilterfw
31 /usr/libexec/hidd
31 /usr/libexec/kextd
...
31 /usr/sbin/mDNSResponder
31 /usr/sbin/notifyd
31 /usr/sbin/ntpd
31 /usr/sbin/pboard
31 /usr/sbin/racoon
31 /usr/sbin/securityd
31 /usr/sbin/syslogd
31 /usr/sbin/update
31 autofsd
31 login
31 ps
31 sort
46 /Applications/Preview.app/Contents/MacOS/Preview
46 /Applications/iCal.app/Contents/MacOS/iCal
47 /Applications/Utilities/Terminal.app/Contents/MacOS/Terminal
50 /System/Library/Frameworks/CoreServices.framework/Frameworks/Metadata.framework/Support/mds
50 /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/CarbonCore.framework/Versions/A/Support/fseventsd
62 /System/Library/CoreServices/Finder.app/Contents/MacOS/Finder
63 /Applications/Safari.app/Contents/MacOS/Safari
63 /Applications/iWork '08/Keynote.app/Contents/MacOS/Keynote
63 /System/Library/CoreServices/Dock.app/Contents/Resources/DashboardClient.app/Contents/MacOS/DashboardClient
63 /System/Library/CoreServices/SystemUIServer.app/Contents/MacOS/SystemUIServer
63 /System/Library/CoreServices/loginwindow.app/Contents/MacOS/loginWindow
63 /System/Library/Frameworks/SystemUIServer.framework/Contents/MacOS/SystemUIServer
63 /System/Library/Frameworks/SystemUIServer.framework/Contents/MacOS/SystemUIServer
68 /sbin/dynamic_pager
68 /usr/sbin/UserEventAgent
68 /usr/sbin/coreaudiod

When scheduler asks “What should I run next?” it could findmin(H).
Plane Sweep: Process points left to right:

Store points in a priority queue, ordered by their $x$ coordinate.
Plane Sweep: Process points left to right:

Store points in a priority queue, ordered by their $x$ coordinate.
Plane Sweep: Process points left to right:

Store points in a priority queue, ordered by their $x$ coordinate.
Plane Sweep: Process points left to right:

Store points in a priority queue, ordered by their $x$ coordinate.
Plane Sweep: Process points left to right:

Store points in a priority queue, ordered by their $x$ coordinate.
Heap-Ordered Trees

- The keys of the children of $u$ are $\geq$ the key($u$), for all nodes $u$.
- (This “heap” has nothing to do with the “heap” part of computer memory.)
Heap-Ordered Trees

- The keys of the children of $u$ are $\geq$ the key($u$), for all nodes $u$.
- (This “heap” has nothing to do with the “heap” part of computer memory.)
Heap – Find min

The minimum element is always the root
Heap – Insert

![Heap Diagram]

Nodes: 2, 3, 12, 7, 10, 8, 12, 9, 7, 10
1. Add node as a leaf
   (we’ll see where later)
Heap – Insert

1. Add node as a leaf
   (we’ll see where later)

2. “sift up:” while current node is < its parent, swap them.
Heap – Insert

1. Add node as a leaf (we’ll see where later)

2. “sift up:” while current node is < its parent, swap them.
Heap – Insert

1. Add node as a leaf (we’ll see where later)

2. “sift up:” while current node is < its parent, swap them.
Heap – Delete($i$)
Heap – Delete\( (i) \)

1. need a pointer to node containing key \( i \)
Heap – Delete($i$)

1. need a pointer to node containing key $i$

2. replace key to delete $i$ with key $j$ at a leaf node (we’ll see how to find a leaf soon)
Heap – Delete($i$)

1. need a pointer to node containing key $i$

2. replace key to delete $i$ with key $j$ at a leaf node (we’ll see how to find a leaf soon)

3. Delete leaf
Heap – Delete($i$)

1. need a pointer to node containing key $i$

2. replace key to delete $i$ with key $j$ at a leaf node (we’ll see how to find a leaf soon)

3. Delete leaf

4. If $i > j$ then sift up, moving $j$ up the tree.

If $i < j$ then “sift down”: swap current node with smallest of children until its bigger than all of its children.
Running Times

- \textit{findmin} takes constant time $[O(1)]$

- \textit{insert}, \textit{delete} take time $\propto$ tree height plus the time to find the leaves.

- \textit{deletemin}: same as \textit{delete}

- But how do we find leaves used in \textit{insert} and \textit{delete}?
  - \textit{delete}: use the last inserted node.
  - \textit{insert}: choose node so tree remains complete.
Store Heap in a Complete Tree
Store Heap in a Complete Tree

```
     2
    / \   
  8    3
 / \   / \
12 9  7 10
/ \   /   /
15 21 A  7 10
```
Store Heap in a Complete Tree
Store Heap in a Complete Tree
Store Heap in a Complete Tree
Store Heap in a Complete Tree
Store Heap in a Complete Tree

```
2
/   \
8   3
/   /   \
12  9  7  10
/   /     /   \
15 21 A   B   C   D   E   F
```
Store Heap in a Complete Tree

left(i): \(2i\) if \(2i \leq n\) otherwise None
right(i): \((2i + 1)\) if \(2i + 1 \leq n\) otherwise None
parent(i): \([i/2]\) if \(i \geq 2\) otherwise None

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>8</th>
<th>3</th>
<th>12</th>
<th>9</th>
<th>7</th>
<th>10</th>
<th>15</th>
<th>21</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
left\left( i \right): 2i \text{ if } 2i \leq n \text{ otherwise } \text{None}
	right\left( i \right): (2i + 1) \text{ if } 2i + 1 \leq n \text{ otherwise } \text{None}

parent\left( i \right): \lfloor i/2 \rfloor \text{ if } i \geq 2 \text{ otherwise } \text{None}
Store Heap in a Complete Tree

left(i): $2i$ if $2i \leq n$ otherwise None
right(i): $(2i + 1)$ if $2i + 1 \leq n$ otherwise None
parent(i): $\lfloor i/2 \rfloor$ if $i \geq 2$ otherwise None
Store Heap in a Complete Tree

left(i): \(2i\) if \(2i \leq n\) otherwise None
right(i): \((2i + 1)\) if \(2i + 1 \leq n\) otherwise None
parent(i): \([i/2]\) if \(i \geq 2\) otherwise None
Store Heap in a Complete Tree

left(i): 2i if 2i ≤ n otherwise None
right(i): (2i + 1) if 2i + 1 ≤ n otherwise None
parent(i): ⌈i/2⌉ if i ≥ 2 otherwise None
Store Heap in a Complete Tree

left(i): $2i$ if $2i \leq n$ otherwise None
right(i): $(2i + 1)$ if $2i + 1 \leq n$ otherwise None
parent(i): $\lfloor i/2 \rfloor$ if $i \geq 2$ otherwise None
Store Heap in a Complete Tree

left(i): $2i$ if $2i \leq n$ otherwise None
right(i): $(2i + 1)$ if $2i + 1 \leq n$ otherwise None
parent(i): $\lfloor i/2 \rfloor$ if $i \geq 2$ otherwise None
Make Heap – Create a heap from $n$ items

- $n$ inserts take time $\propto n \log n$.
- Better:
  - put items into array arbitrarily.
  - for $i = n \ldots 1$, siftdown$(i)$.
- Each element trickles down to its correct place.

By the time you sift level $i$, all levels $i + 1$ and greater are already heap ordered.
Make Heap – Time Bound

There are at most \( n/2^h \) items at height \( h \).

Siftdown for all height \( h \) nodes is \( \approx hn/2^h \) time

Total time
\[
\approx \sum_h \frac{hn}{2^h}
\]

[sum of time for each height]
\[
= n \sum_h \left( \frac{h}{2^h} \right)
\]

[factor out the \( n \)]
\[
\approx n
\]

[sum bounded by const]
\textbf{\textit{d-Heaps}}

- What about complete non-binary trees (e.g. every node has \textit{d} children)?
  - \textit{insert} takes $O(\log_d n)$ \hfill [because height $O(\log_d n)$]
  - \textit{delete} takes $O(d \log_d n)$ \hfill [why?]

- Can still store in an array.

- If you have few deletions, make \textit{d} bigger so that tree is shorter.

- Can tune \textit{d} to fit the relative proportions of inserts / deletes.
d-Heap Runtime Summary

- `findmin` takes $O(1)$ time
- `insert` takes $O(\log_d n)$ time
- `delete` takes $O(d \log_d n)$ time
- `deletemin` takes time $O(d \log_d n)$
- `makeheap` takes $O(n)$ time
3-Heap Example
Heapsort – Another application of Heaps

Given unsorted array of integers

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>12</th>
<th>10</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>
Heapsort – Another application of Heaps

Given unsorted array of integers

makeheap – O(n)
Now first position has smallest item.

Swap first & last items.
Heapsort – Another application of Heaps

Given unsorted array of integers

makeheap – O(n)
Now first position has smallest item.

Swap first & last items.
Heapsort – Another application of Heaps

Given unsorted array of integers

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>12</th>
<th>10</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

makeheap – O(n)

Now first position has smallest item.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

Delete last item from heap.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>
Heapsort – Another application of Heaps

Given unsorted array of integers

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>12</th>
<th>10</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

makeheap – \(O(n)\)
Now first position has smallest item.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

Delete last item from heap.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>
Heapsort – Another application of Heaps

Given unsorted array of integers

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>12</th>
<th>10</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

makeheap – O(n)
Now first position has smallest item.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

Delete last item from heap.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

siftdown new root key down

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>
Heapsort – Another application of Heaps

Given unsorted array of integers

makeheap – O(n)
Now first position has smallest item.

Delete last item from heap.

siftdown new root key down
Heapsort – Another application of Heaps

Given unsorted array of integers

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>12</th>
<th>10</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

makeheap – O(n)
Now first position has smallest item.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

Delete last item from heap.

<table>
<thead>
<tr>
<th>10</th>
<th>3</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>

siftdown new root key
down

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>12</th>
<th>8</th>
<th>10</th>
<th>15</th>
<th>21</th>
<th>9</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</tbody>
</table>