COMP 250

Lecture 23

heaps 2

Nov. 2, 2016
RECALL: **min Heap** (definition)

Complete binary tree with (unique) comparable elements, such that each node’s element is less than its children’s element(s).
Heap index relations

parent = child / 2
left = 2*parent
right = 2*parent + 1
add(element)

“upHeap”

removeMin()

“downHeap”
add(element ){
    size = size + 1 // number of elements in heap
    heap[ size ] = element // assuming array
        // has room for another element
    i = size

    // the following is sometimes called "upHeap"

    while ( i > 1 and heap[i] < heap[ i/2 ]){
        swapElements( i, i/2 )
        i = i/2
    }
}
e.g. \texttt{add(c)}
e.g. $\text{add}(c)$
e.g. add(c)
e.g. \( \text{add}(c) \)
Given a list with size elements:

```java
buildHeap(list){
    create new heap array        // length > list.size
    for (k = 0; k < list.size; k++)
        add( list[k] )        // add to heap[ ]
}
```
Best case: buildHeap is $\Omega(n)$

In the best case, the list is already a heap, and no swaps are necessary.
Worse case of buildHeap?

Thus,

\[ 2^{level} \leq i < 2^{level+1} \]

\[ level \leq \log_2 i < level + 1 \]

Thus, \( level = \text{floor}(\log_2 i) \)
Worse case of buildHeap

\[ t(n) = \sum_{i=1}^{n} \text{floor}( \log_2 i ) \]
The graph shows the function $\log_2 i$ and its floor function $\text{floor}(\log_2 i)$.
Area under the dashed curve is the total number of swaps (worst case) of buildHeap.

\[ t(n) = \sum_{i=1}^{n} \text{floor}(\log_2 i) \]
$t(n) \leq n \log_2 n$
Thus, worst case: buildHeap is $O(n \log_2 n)$

Next lecture I will show you a $O(n)$ algorithm.
add(element)

“upHeap”

removeMin()

“downHeap”
e.g. `removeMin()`
removeMin()

Let heap[ ] be the array.
Let size be the number of elements in the heap.

removeMin()
{
    heap[1] = heap[size]
    heap[size] = null
    size = size - 1
    downHeap(1, size)
    return element
}
downHeap( startIndex, maxIndex ){

    i = startIndex
    while (2*i <= maxIndex){   // if there is a left child
        child = 2*i
        if child < size {        // if there is a right sibling
            if (heap[child + 1] < heap[child])   // if rightchild < leftchild ?
                child = child + 1

        }
        if (heap[child] < heap[ i ]){   // Do we need to swap with child?
            swapElements(i, child)
            i = child
        }
    }
}
Heapsort

Given a list with size elements:

heap = buildHeap(list)
for k = 1 to size{
    list[ size - k ] = heap.removeMin()
}
Heapsort

Given a list with size elements:

```plaintext
heapsort( list ){
    buildheap(list)
    for i = 1 to size{
        swapElements( heap[1], heap[size + 1 - i])
        downHeap( 1, size – i )
    }
    return reverse(heap)
}
```
1 2 3 4 5 6 7 8 9
-------------------------------------
a  d  b  e  l  u  k  f  w |
b  d  k  e  l  u  w  f  | a
<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>k</td>
<td>f</td>
<td>w</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>f</td>
<td>a</td>
</tr>
<tr>
<td>f</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>f</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
<td>a</td>
</tr>
<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>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>k</td>
<td>f</td>
<td>w</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>f</td>
<td>a</td>
</tr>
<tr>
<td>f</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>f</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>k</td>
<td>f</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>k</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>k</td>
<td>f</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>k</td>
<td>f</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>f</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>k</td>
<td>f</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>k</td>
<td>w</td>
<td>l</td>
<td>u</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>k</td>
<td>f</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>k</td>
<td>e</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>f</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>k</td>
<td>f</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>b</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>k</td>
<td>w</td>
<td>l</td>
<td>u</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>f</td>
<td>l</td>
<td>k</td>
<td>w</td>
<td>u</td>
<td>e</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>k</td>
<td>l</td>
<td>u</td>
<td>w</td>
<td>f</td>
<td>e</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>l</td>
<td>w</td>
<td>u</td>
<td>k</td>
<td>f</td>
<td>e</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>u</td>
<td>w</td>
<td>l</td>
<td>k</td>
<td>f</td>
<td>e</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>w</td>
<td>u</td>
<td>l</td>
<td>k</td>
<td>f</td>
<td>e</td>
<td>d</td>
<td>b</td>
</tr>
<tr>
<td>w</td>
<td>u</td>
<td>l</td>
<td>k</td>
<td>f</td>
<td>e</td>
<td>d</td>
<td>b</td>
</tr>
</tbody>
</table>
Heapsort

heapsort(list){
    buildheap(list)
    for i = 1 to size{
        swapElements( heap[1], heap[size + 1 - i])
        downHeap( 1, size - i)
    }
    return reverse(heap)
}
Best and worst case of heapsort?

See Exercises.