Priority Queues

Chapter 6 Overview

- **The Concept**
- **Possible Implementations**
- **Binary Heaps**
- **Applications**

We will NOT cover

- d-Heaps
- Leftist Heaps
- Skew Heaps
- Binomial Queues

A priority queue is a 'queue' where the first element out is the one with the minimum key value, which we will take to mean the highest priority.

Operations:

DeleteMin ← Priority Queue ← Insert

Possible Implementations: complexity?

- linked list
  Insert adds to the end.
  DeleteMin has to search.

- binary search tree
  Use normal Insert and FindMin.

Binary Heaps

A heap is a binary tree that is full except for the bottom level, which is filled from left to right.

We use an **array implementation** of a binary tree, which saves memory and can be very efficient for this purpose.

A[0] is not used.
A[1] is the root of the tree.

A[i] has children at A[2i] and A[2i+1].
A[i] has parent at $A[\lfloor i / 2 \rfloor]$.

13	21	16	24	31	19
1	2	3	4	5	6

What tree is this?

Operations for Array Implementation

- Initialize (which also allocates)
- Destroy (which ought to delete)
- MakeEmpty
- Insert
- FindMin
- DeleteMin
- IsEmpty
- IsFull

How do we make the heap an **efficient** structure for priority queue operations?

Keep it in order according to the heap order property.

Heap Order Property

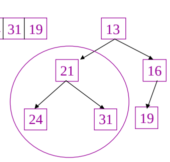
for every node **X** of the tree:

key(parent(X)) ≤ key(X)

This implies that

- the minimum value is at the root
- this is true of any subtree as well

13	21	16	24	31	19



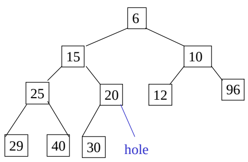
Inserting in a Binary Heap

```
Insert(X)  /* pseudo-code */

- let hole be the next unfilled node in the tree

- while ( X is not yet placed )
     if parent(hole) ≤ X, put X in hole
     else
        1. move the parent's data into hole
        2. let hole be the parent's node
```

We say that X **percolates up** the tree.

DS.P.7

Insertion Example



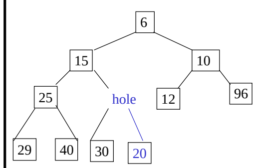
insert 13

Can I put 13 in the hole?

Will the order property still hold if I move the 20 into the hole?

DS.P.8

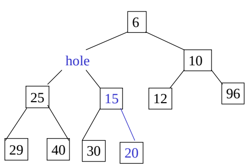
Insertion Example



Now can I put the 13 in the hole?

DS.P.9

Insertion Example



Now can I put the 13 in the hole?

DS.P.10

The implementation of Insert is very concise, using only simple array operations and with worst case complexity O(log n).

-∞	6	15	10	25	20	12	96	29	40	30	hole
0	1	2	3	4	5	6	7	8	9	10	11

-∞	6	15	10	25	hole	12	96	29	40	30	20
0	1	2	3	4	5	6	7	8	9	10	11

-∞	6	13	10	25	15	12	96	29	40	30	20
0	1	2	3	4	5	6	7	8	9	10	11

```
for (i = ++H->Size; H->Elements[ i/2 ] > X; i /= 2)
   H->Elements[i] = H->Elements[ i/2 ];
H->Elements[i] = X;
```

DS.P.11

DeleteMin

deleteMin

- Find the minimum value at the root
- Deleting it leaves a hole at the root.
- This hole must percolate down till we can place the last element in it.
- So swap it with its smaller child.
- Continue this process till the last element can be placed in the hole.

Try this: first draw it as a tree, then try the process.

6	15	10	25	20	12	96	29	40	30

DS.P.12

Building a Heap from Unordered Data

The idea:

- Put the data in an array.
- Start with the last internal node L.
- Percolate it down to its proper place.
- Continue this process for each internal node.

The code:

```
for (i = N / 2; i > 0; i-- )
    PercolateDown(i);
```

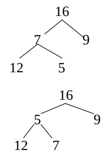
DS.P.13

Example

16	7	9	12	5
1	2	3	4	5

N = 5; N/5 = 2

1. PercolateDown(2)



2. PercolateDown(1)

Which is Node 1?
Try it!

DS.P.14

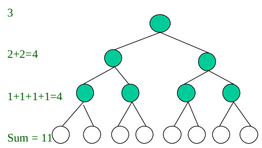
Complexity

- Insert ???
- DeleteMin ???
- BuildHeap

Th. The running time of BuildHeap is O(N).

- PercolateDown is called potentially for every nonleaf node.
- Each time, it can go all the way down, swapping the smallest child with its parent, if needed.
- This is approximately the sum of the of the heights of the nodes in the tree.

DS.P.15



Th. 6.1 For perfect binary tree of height h with 2^{h+1} - 1 nodes, the sum of the heights is

$2^{h+1} - 1 - (h+1)$

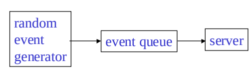
4. Therefore, for 2^{h+1} nodes, the sum is $O(2^{h+1})$,

and for N nodes, the sum is O(N).

DS.P.16

Applications of Heaps

- job and process queues
- event queues in simulation



Simulation clock starts at time 0.
Instead of checking what events happen at every tick, we just find the **next event** (the one with minimum starting time) and change the clock to that time.

Clock: 0, 5, 14, 23, 28, …

- sorting

DS.P.17

Heapsort (from Chapter 7)

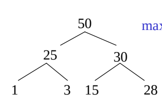
How can we use a heap to achieve sorting an array into ascending order?

- Build a max heap (instead of a min heap) in the array.
- Use DeleteMax (which is just like DeleteMin) to remove the largest element.
- Put that largest element at the end of the array, which will have become an empty spot through operation of DeleteMax.

DS.P.18

Heapsort Example

50	25	30	1	3	15	28



max element

What does the array look like after the max element is deleted?

Then what does it look like after the max element is put at the end?

DS.P.19

Heapsort Complexity

For n elements

It takes $2n$ comparisons to build the heap.

After that there are n sorting steps

with $2\log_2 i$ comparisons at the ith step.

$$2n + 2\sum_{i=1}^{n} \log_2 i$$

This term dominates.

$= O(n \log_2 n)$