13—Priority Queues

§9.1, §11.3

May 4, 2002

Frequency Assignment

$\Downarrow$

$\Downarrow$

970	the
708	and
666	of
632	to
521	I
466	a
466	in
466	my
383	you

The Text Engine

$\Downarrow$

$\longrightarrow$

BST

AVL Tree

Hashtable(s)

Splay Tree

⋮

$\Downarrow$

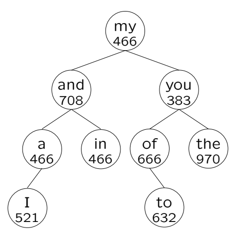
970	the
708	and
666	of
632	to
521	I
466	a
466	in
466	my
383	you

The Main Loop

D = **new** DictAVL or DictHash or DictWhatever;
while (still words left) {
 w = next word;

}
D−>PrintSorted(num_to_print);

Printing the List



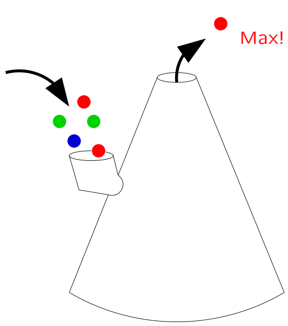
⇒

970	the
708	and
666	of
632	to
521	I
466	a
466	in
466	my
383	you

- What if we want to print all n words?

- What if we want to print only the top ten words?

Priority Queues



- MakeEmpty(), IsEmpty()
- Insert(key, info)
- FindMax(), DeleteMax()

How to Use

```
DictWhatever::PrintSorted(num_to_print)
{
  PQ pq;
  for (each record in dict)
   pq->Insert(record.freq, record.str);

  for (i = 1... num_to_print)
   Print pq->DeleteMax();
}
```


Implementations of PQs

Linked List

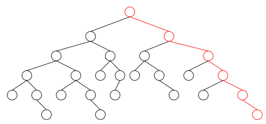
- Insert
- FindMax
- DeleteMax

Implementations of PQs

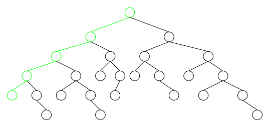
AVL Tree

- Insert
- FindMax
- DeleteMax

AVL Tree PQ



FindMax()



FindMin()

Removing the First k Items

Hashtable

of	708
forsooth	200
my	466
to	632
I	521
a	466
codpiece	382
in	466
the	970
nunnery	2
and	708
frier	279
you	383

$\Rightarrow$

Top k Freqs

970	the
708	and
666	of
632	to
521	I
466	a

Running time for...

- Sorting and printing first k?
- Linked List PQ?
- AVL PQ?

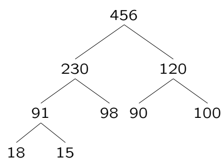
Detailed AVL PQ Creation

- $\approx n/2^h$ nodes inserted when tree is height h, for $h = 0, \ldots, \log n$
- The height 0 nodes take constant time to insert
- Time h to insert node at height h, for $h \geq 1$
- Total time for rest is

$$\begin{aligned}
\sum_{h=1}^{\log n} h \cdot \frac{n}{2^h} &= n \cdot \sum_{h=1}^{\log n} \frac{h}{2^h} \\
&\leq n \cdot \left(\frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \cdots\right) \\
&= n \cdot \Big(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \\
&\qquad + \frac{1}{4} + \frac{1}{8} + \cdots \\
&\qquad\qquad + \frac{1}{8} + \cdots\Big) \\
&= n \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right) \\
&= 2 \cdot n
\end{aligned}$$

Towards an Easier Implementation

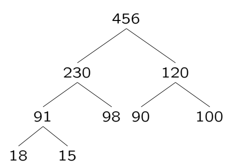
Partially Ordered Tree



- Parent *bigger* than children

A Hard Operation

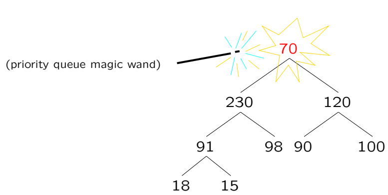
FindMax()?



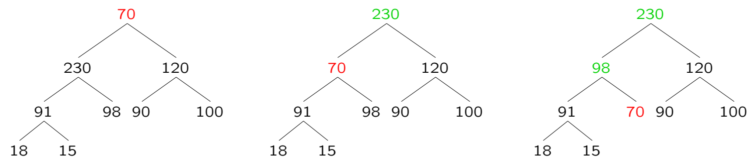
DeleteMax()? Insert()?

A Strange Operation

Decrease the Root



A Strange Operation

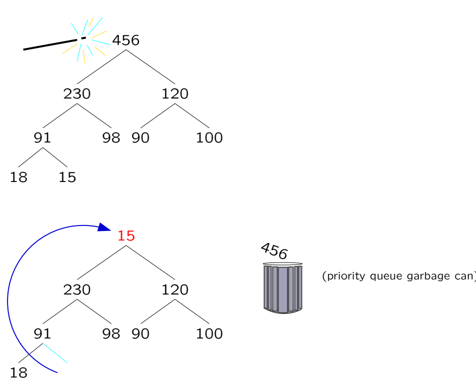


Let's Write Some Code

```
DecreaseRoot(Node *n, int val)
{
    n->key = val;









}
```

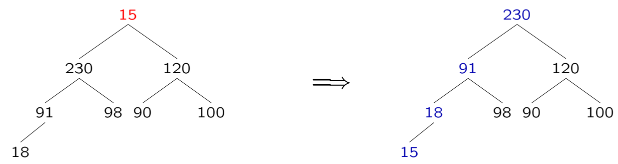

More Useful Than You Think

DeleteMin()



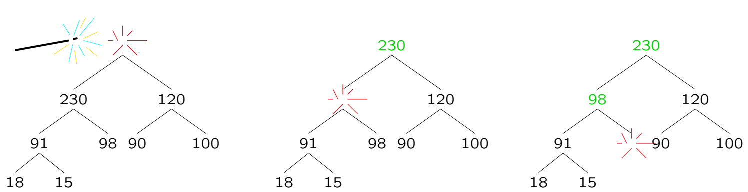
How do we fix the root?

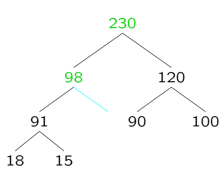
Deletion



Deletion

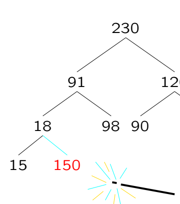
Another way to do it





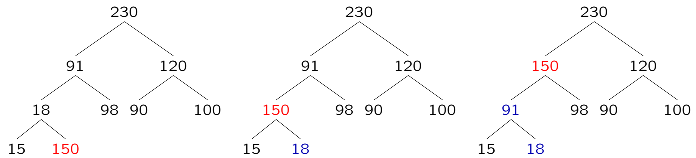
Why do we do it the other way?

Insertion



Fill out complete binary tree

Insertion

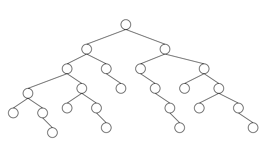


Swap *up*

Running Time?

Talking About Complete Trees

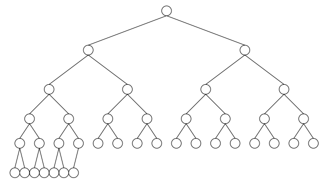
The space overhead of trees is annoying. . .



. . . but trees can be complicated, so it's necessary

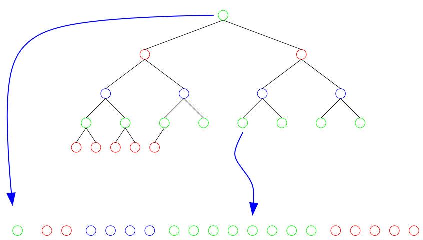
Talking About Complete Trees

Complete trees are very *regular*



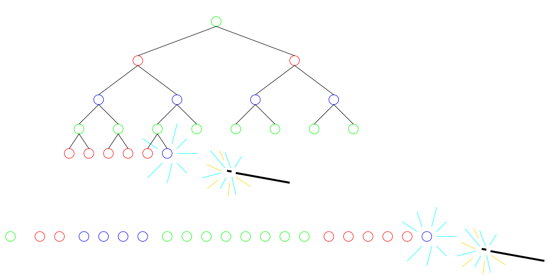
Do we really need all those pointers?

The Answer is No



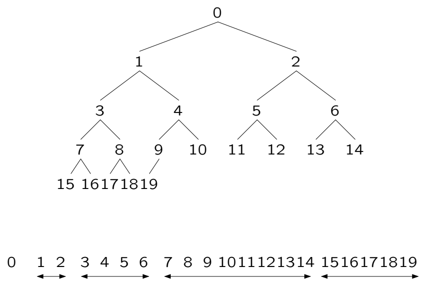
Store as an array: *Heap* implementation of PQs

Adding and Removing



Heaps are easy!

Indicies of Heap in Array



These are array indicies, not key values

Implementing Heaps


```
int Heap::LeftIndex( int i )
{
  if (2*i+1 < array_size )
   return 2*i+1;
  return -1;
}
```

```
int Heap::RightIndex( int i )
{
  if (2*i+2 < array_size )
   return 2*i+2;
  return -1;
}
```

```
void DecreaseRoot(Key new_root)
{
  array [0] = new_root;










}
```

Our Application

```
void PrintTopFreq(
    FreqHashTable& hash;
    int k)
{






}
```

Hashtable

of	708
forsooth	200
my	466
to	632
I	521
a	466
codpiece	382
in	466
the	970
nunnery	2
and	708
frier	279
you	383

$\Rightarrow$

Top k Freqs

970	the
708	and
666	of
632	to
521	I
466	a

In-Place Heap Creation

708 200 466 632 521 466 382 466 970 2 708 299 383

Heap Sort

1. Make heap, in-place
2. DeleteMax() by swapping root of heap to *back* of array

   31 27 29 18 25 28 19 2 16 5
   5 27 29 18 25 28 19 2 16 31

3. Swap down to fix heap

   5 27 29 18 25 28 19 2 16 31
   29 27 5 18 25 28 19 2 16 31
   29 27 28 18 25 5 19 2 16 31

4. Continue until sorted

   Swap 29,16:
   16 27 28 18 25 5 19 2 29 31
   28 27 19 18 25 5 16 2 29 31
   Swap 28,2:
   2 27 19 18 25 5 16 28 29 31
   27 25 19 18 2 5 16 28 29 31
   ⋮

Heap Sort Summary

- $O(n \log n)$ time, guaranteed
- In-place and efficient operations
  - No extra memory, only doing simple swaps
- What if initial array partially sorted?
- How is cache performance?