(Lecture 25) Priority Queues (Heaps)

A ***priority queue*** is a data structure that allows at least the following two operations:

- **Insert**: which does the obvious thing;
- **deleteMin (or deleteMax)**: which finds, returns, and removes the minimum (or maximum) element in the priority queue.

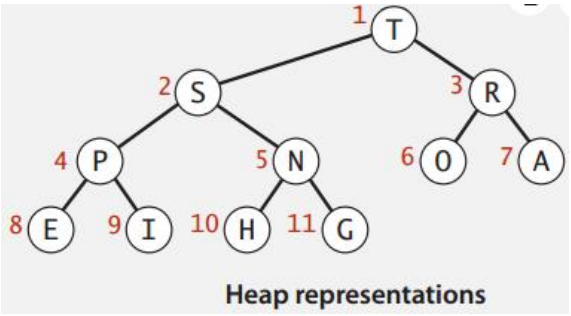
Simple Implementations:

- **Unsorted Linked list**, performing insertions at the front in **O(1)** and traversing the list, which requires **O(N)** time, to delete the minimum/maximum.
- **Sorted Linked list**, performing insertions in **O(N)** and **O(1)** to delete the minimum/maximum.
- **Binary search tree**: this gives an **O(log N)** average running time for both operations.

Binary Heap

A heap is a binary tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right.
Such a tree is known as a **complete binary tree**.

A complete binary tree of height ***h*** has between 2^h and $2^{h+1} - 1$ nodes.



Heap representations

As complete binary tree is so regular, it can be represented as an array:

i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	-	T	S	R	P	N	O	A	E	I	H	G

- Parent of node at ***i*** is at ***i/2***.
- Children of node at ***i*** are at ***2i*** (left child) and ***2i+1*** (right child).

Heap-order property:

- In a **min heap**, for every node **X**, the key in the parent of **X** is smaller than (*or equal to*) the key in **X**, with the exception of the root (which has no parent). Therefore, the minimum element can always be found at the root.
- In a **max heap**, for every node **X**, the key in the parent of **X** is larger than (*or equal to*) the key in **X**, with the exception of the root (which has no parent). Therefore, the maximum element can always be found at the root.

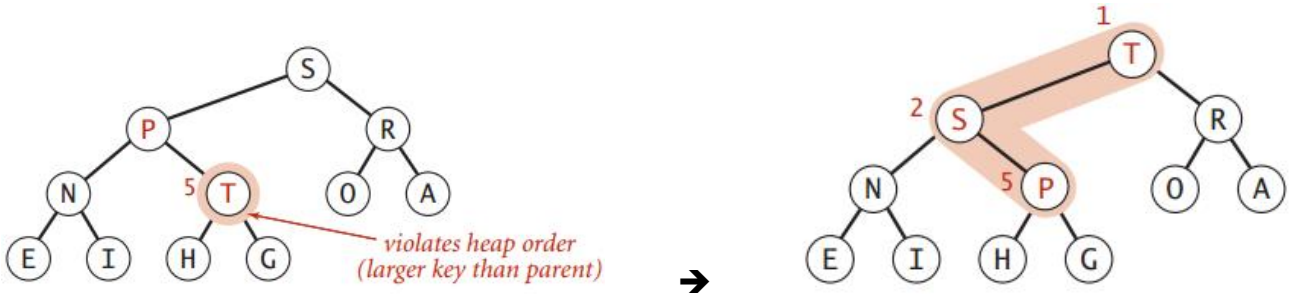
Promotion in a heap

Scenario 1: Child's key becomes larger than its parent's key.

To eliminate the violation:

- Exchange key in child with key in parent.
- Repeat until heap order restored.

Example:




```
private void swim(int k)
{
   while (k > 1 && less(k/2, k))
   {
      exch(k, k/2);
      k = k/2;
   }
}
```

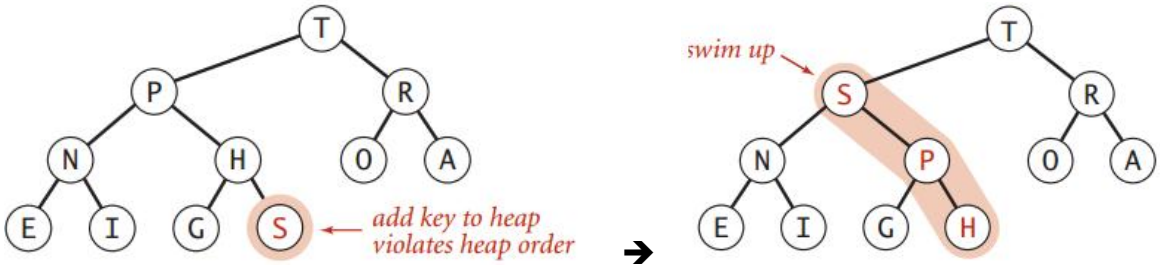
parent of node at k is at k/2

Insertion in a heap

Insert: Add node at end, then swim it up.

Cost: At most ***1 + lg N*** compares.

Example: Insert **S**




```
public void insert(Key x)
{
   pq[++N] = x;
   swim(N);
}
```

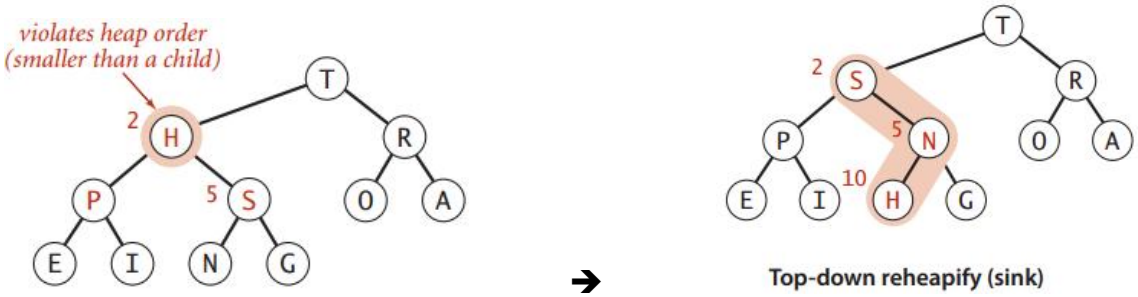
Demotion in a heap

Scenario 2: Parent's key becomes smaller than one (or both) of its children's.

To eliminate the violation:

- Exchange key in parent with key in larger child.
- Repeat until heap order restored.

Example:




```
private void sink(int k)
{
   while (2*k <= N)
   {
      int j = 2*k;
      if (j < N && less(j, j+1)) j++;
      if (!less(k, j)) break;
      exch(k, j);
      k = j;
   }
}
```

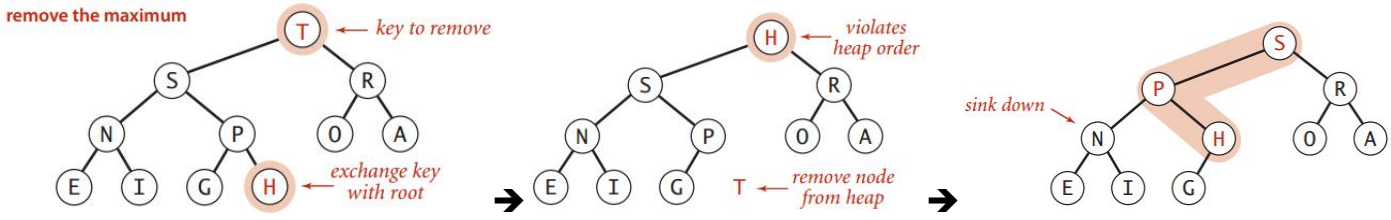
children of node at k are 2k and 2k+1

Delete the maximum in a heap

Delete max: Exchange root with node at end, and then sink it down.

Cost: At most ***2 lg N*** compares.

Example: delete **T**




```
public Key delMax()
{
   Key max = pq[1];
   exch(1, N--);
   sink(1);
   pq[N+1] = null;
   return max;
}
```

prevent loitering

Binary heap: Java implementation

```
public class MaxPQ<Key extends Comparable<Key>>
{
   private Key[] pq;
   private int N;

   public MaxPQ(int capacity)
   {  pq = (Key[]) new Comparable[capacity+1];  }

   public boolean isEmpty()
   {   return N == 0;   }
   public void insert(Key key)
   public Key delMax()
   {   /* see previous code */  }

   private void swim(int k)
   private void sink(int k)
   {   /* see previous code */  }

   private boolean less(int i, int j)
   {   return pq[i].compareTo(pq[j]) < 0;  }
   private void exch(int i, int j)
   {   Key t = pq[i]; pq[i] = pq[j]; pq[j] = t;  }
}
```

fixed capacity (for simplicity)

PQ ops

heap helper functions

array helper functions