Priority Queues

CS112 Recitation

Coding Files:
https://drive.google.com/drive/folders/1LLAhBd0LOvdpTvwbr47oDu5SJLhDLU9A?usp=sharing

Hoai-An Nguyen
hnn14@scarletmail.rutgers.edu

Let's Review

- A priority queue is a structure in which items have priorities
  - Many different data structures can be used to implement priority queues
- When we remove from a PQ, we remove the item with the highest priority (i.e. the largest item in a max PQ or the smallest in a min PQ)

operation	argument	return value	size	contents (unordered)	contents (ordered)
insert	P		1	P	P
insert	Q		2	P Q	P Q
insert	E		3	P Q E	E P Q
remove max		Q	2	P E	E P
insert	X		3	P E X	E P X
insert	A		4	P E X A	A E P X
insert	M		5	P E X A M	A E M P X
remove max		X	4	P E M A	A E M P
insert	P		5	P E M A P	A E M P P
insert	L		6	P E M A P L	A E L M P
insert	E		7	P E M A P L E	A E E L M
remove max		P	6	E E M A P L	A E E L M

Heaps!

- Popular data structure for priority queues
- Insertion, deletion (removing highest priority element) in O(log n)
- How are heaps implemented?
  - Represented by an array…
  - … but implicitly is a complete binary tree
  - ????

Binary heap: representation

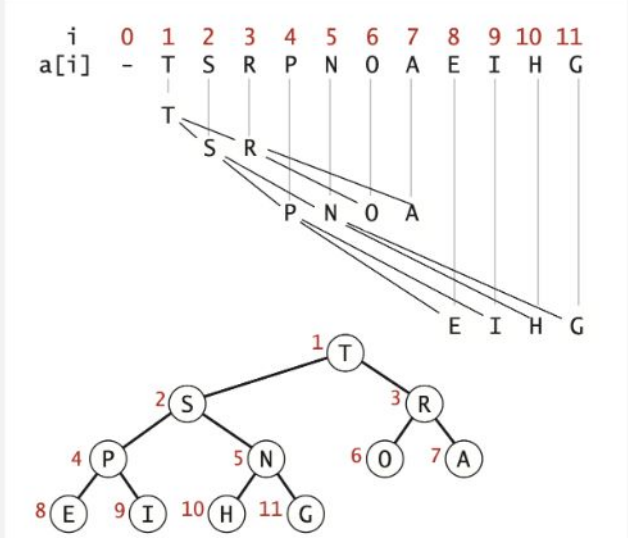
Binary heap. Array representation of a heap-ordered complete binary tree.

Heap-ordered binary tree.

- Keys in nodes.
- Parent's key no smaller than children's keys.

Array representation.

- Indices start at 1.
- Take nodes in level order.
- No explicit links needed!



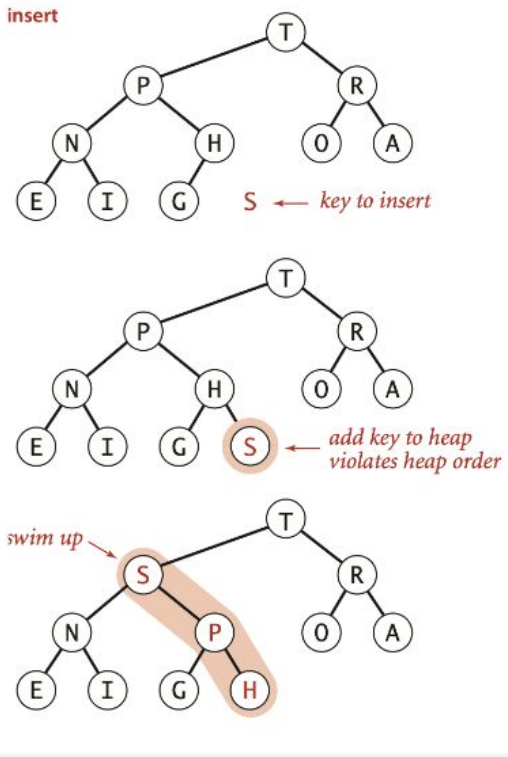
Insertion into Heap

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

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

Heap order invariant violated

parent of node at k is at k/2



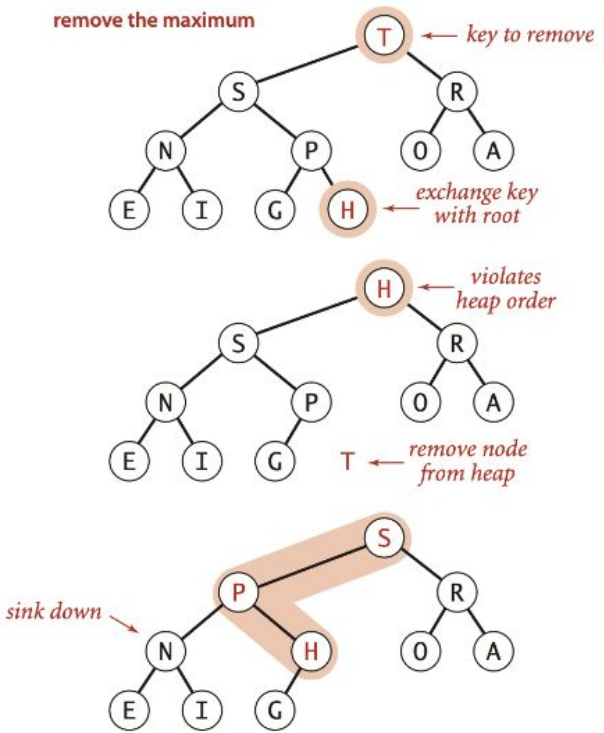
Deletion (Dequeue) from Heap

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

```
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 2*k and 2*k+1

Heap order invariant violated



Warm-Up

1. True or false? An array that is sorted in decreasing order a max-oriented heap.
2. Suppose that your application will have a huge number of insert operations, but only a few remove the maximum operations. Which priority-queue implementation is the most effective?
   a. heap
   b. unordered array
   c. ordered array
3. Why would we prefer using a PQ over sorting an array to find the top *x* maximum values of a collection of items? (Hint: consider a situation in which we need to deal with billions of items)

Warm-Up Solutions

1. True or false? An array that is sorted in decreasing order a max-oriented heap. **True!**
2. Suppose that your application will have a huge number of insert operations, but only a few remove the maximum operations. Which priority-queue implementation is the most effective?
   a. heap - worst case for insertion is O(logn)
   b. unordered array - worst case for insertion is O(1)
   c. ordered array - worst case for insertion is O(n)
3. Why would we prefer using a PQ over sorting an array to find the top *x* maximum values of a collection of items? (Hint: consider a situation in which we need to deal with billions of items)
   If we only need the top *x* maximum values, sorting the array completely is unnecessary work.

Q1 - Linear Certification

Design a linear-time certification algorithm to check whether an array pq[] is a min-oriented heap.

Write the code, write two test cases (and verify them), and verify the Big O runtime.

Note: Look at the coding file for a few test cases.

```
private static boolean certification(int[] pq) {

        for (int i = 1; i < pq.length; i++) {









        }


        return true;
}
```

Q1 Answer

```
private static boolean certification(Comparable[] pq) {

    for(int i = 1; i < pq.length; i++) {
        //Check left child
        if (i * 2 < pq.length && !ArrayUtil.less(pq[i], pq[i * 2])) {
            return false;
        }

        //Check right child
        if (i * 2 + 1 < pq.length && !ArrayUtil.less(pq[i], pq[i * 2 + 1])) {
            return false;
        }
    }

    return true;
}
```

Q2 - Insert and remove maximum

Suppose that the sequence

P R I O * R * * I * T Y *

(where a letter means insert and an asterisk means remove the maximum) is applied to an initially empty priority queue. Give the sequence of letters returned by the remove the maximum operations

Q2 - Insert and remove maximum

P R I O * R * * I * T Y *

Q3 - Dynamic Median-Finding

Design a data type that supports insert in logarithmic time, find the median in constant time, and delete the median in logarithmic time.

Verify the Big O for each of the methods.

Hint: Use a min-heap and a max-heap.

Note: Use the coding file to get started. It also has some driver code and test cases.

Q2 Answer - Initial Setup (Given)

```
private class DynamicMedianFindingHeap<Key extends Comparable<Key>> {

    private MinPQ<Key> minPriorityQueue;
    private MaxPQ<Key> maxPriorityQueue;

    private int size;

    DynamicMedianFindingHeap() {
        minPriorityQueue = new MaxPQ<>();
        maxPriorityQueue = new MinPQ<>();
        size = 0;
    }
```

Q2 Answer - Insert

```
//O(lg N)
public void insert(Key key) {

    if (size == 0 || ArrayUtil.less(key, maxPriorityQueue.peek())) {
        maxPriorityQueue.insert(key);
    } else {
        minPriorityQueue.insert(key);
    }

    if (minPriorityQueue.size() > maxPriorityQueue.size() + 1) {
        Key keyToBeMoved = minPriorityQueue.deleteTop();
        maxPriorityQueue.insert(keyToBeMoved);
    } else if (maxPriorityQueue.size() > minPriorityQueue.size() + 1) {
        Key keyToBeMoved = maxPriorityQueue.deleteTop();
        minPriorityQueue.insert(keyToBeMoved);
    }

    size++;
}
```

Q2 Answer - finding the median

```
//O(1)
public Key findTheMedian() {
    Key median;

    if (minPriorityQueue.size() > maxPriorityQueue.size()) {
        median = minPriorityQueue.peek();
    } else {
        median = maxPriorityQueue.peek();
    }

    return median;
}
```

Q2 Answer - Delete Median

```
//O(lg N)
public Key deleteMedian() {
    Key median;

    if (minPriorityQueue.size() > maxPriorityQueue.size()) {
        median = minPriorityQueue.deleteTop();
    } else {
        median = maxPriorityQueue.deleteTop();
    }

    size--;

    return median;
}
```

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code: Y9UF