CS2010: ALGORITHMS AND DATA STRUCTURES
Lecture 9: Priority Queues

Vasileios Koutavas
21 Oct 2015

School of Computer Science and Statistics
Trinity College Dublin
2.4 Priority Queues

- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation
2.4 PRIORITY QUEUES

- API and elementary implementations
- binary heaps
- heapsort
- event-driven simulation
A **collection** is a data types that store groups of items.

<table>
<thead>
<tr>
<th>data type</th>
<th>key operations</th>
<th>data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
<td><strong>PUSH, POP</strong></td>
<td>linked list, resizing array</td>
</tr>
<tr>
<td>queue</td>
<td><strong>ENQUEUE, DEQUEUE</strong></td>
<td>linked list, resizing array</td>
</tr>
<tr>
<td>priority queue</td>
<td><strong>INSERT, DELETE-MAX</strong></td>
<td>binary heap</td>
</tr>
<tr>
<td>symbol table</td>
<td><strong>PUT, GET, DELETE</strong></td>
<td><strong>BST, hash table</strong></td>
</tr>
<tr>
<td>set</td>
<td><strong>ADD, CONTAINS, DELETE</strong></td>
<td><strong>BST, hash table</strong></td>
</tr>
</tbody>
</table>

“Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious.” — Fred Brooks
Priority queue

Collections. Insert and delete items. Which item to delete?

Stack. Remove the item most recently added.
Queue. Remove the item least recently added.
Randomized queue. Remove a random item.

Priority queue. Remove the largest (or smallest) item.

<table>
<thead>
<tr>
<th>operation</th>
<th>argument</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>remove max</td>
<td>E</td>
<td>Q</td>
</tr>
<tr>
<td>insert</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>remove max</td>
<td>M</td>
<td>X</td>
</tr>
<tr>
<td>insert</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>remove max</td>
<td>E</td>
<td>P</td>
</tr>
</tbody>
</table>
Priority queue API

**Requirement.** Generic items are Comparable.

```java
public class MaxPQ<Key extends Comparable<Key>> {
    // Constructor
    MaxPQ()
    MaxPQ(Key[] a)

    // Operations
    void insert(Key v)
    Key delMax()
    boolean isEmpty()
    Key max()
    int size()
}
```

- **MaxPQ()**
  - create an empty priority queue
- **MaxPQ(Key[] a)**
  - create a priority queue with given keys
- **void insert(Key v)**
  - insert a key into the priority queue
- **Key delMax()**
  - return and remove the largest key
- **boolean isEmpty()**
  - is the priority queue empty?
- **Key max()**
  - return the largest key
- **int size()**
  - number of entries in the priority queue
**Priority queue applications**

- Event-driven simulation.  [ customers in a line, colliding particles ]
- Numerical computation.  [ reducing roundoff error ]
- Data compression.  [ Huffman codes ]
- Graph searching.  [ Dijkstra's algorithm, Prim's algorithm ]
- Number theory.  [ sum of powers ]
- Artificial intelligence.  [ A* search ]
- Statistics.  [ online median in data stream ]
- Operating systems.  [ load balancing, interrupt handling ]
- Computer networks.  [ web cache ]
- Discrete optimization.  [ bin packing, scheduling ]
- Spam filtering.  [ Bayesian spam filter ]

**Generalizes:** stack, queue, randomized queue.
Challenge. Find the largest $M$ items in a stream of $N$ items.

- Fraud detection: isolate $$ transactions.
- NSA monitoring: flag most suspicious documents.

Constraint. Not enough memory to store $N$ items.

```
% more tinyBatch.txt
Turing       6/17/1990   644.08
vonNeumann   3/26/2002  4121.85
Dijkstra     8/22/2007  2678.40
vonNeumann   1/11/1999  4409.74
Dijkstra     11/18/1995 837.42
Hoare        5/10/1993  3229.27
vonNeumann   2/12/1994  4732.35
Hoare        8/18/1992  4381.21
Turing       1/11/2002  66.10
Thompson     2/27/2000  4747.08
Turing       2/11/1991  2156.86
Hoare        8/12/2003  1025.70
vonNeumann   10/13/1993 2520.97
Dijkstra     9/10/2000  708.95
Turing       10/12/1993 3532.36
Hoare        2/10/2005  4050.20
```

```
% java TopM 5 < tinyBatch.txt
Thompson     2/27/2000  4747.08
vonNeumann   2/12/1994  4732.35
vonNeumann   1/11/1999  4409.74
Hoare        8/18/1992  4381.21
vonNeumann   3/26/2002  4121.85
```
Challenge. Find the largest $M$ items in a stream of $N$ items.
- Fraud detection: isolate $\$\$ transactions.
- NSA monitoring: flag most suspicious documents.

Constraint. Not enough memory to store $N$ items.

```java
MinPQ<Transaction> pq = new MinPQ<Transaction>();
while (StdIn.hasNextLine())
{
    String line = StdIn.readLine();
    Transaction item = new Transaction(line);
    pq.insert(item);
    if (pq.size() > M)
        pq.delMin();
}
```

Transaction data type is Comparable (ordered by $\$\$)

pq contains largest $M$ items

use a min-oriented pq

N huge, M large
Priority queue client example

**Challenge.** Find the largest $M$ items in a stream of $N$ items.

<table>
<thead>
<tr>
<th>implementation</th>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>$N \log N$</td>
<td>$N$</td>
</tr>
<tr>
<td>elementary PQ</td>
<td>$M \cdot N$</td>
<td>$M$</td>
</tr>
<tr>
<td>binary heap</td>
<td>$N \log M$</td>
<td>$M$</td>
</tr>
<tr>
<td>best in theory</td>
<td>$N$</td>
<td>$M$</td>
</tr>
</tbody>
</table>

order of growth of finding the largest $M$ in a stream of $N$ items
### Priority queue: unordered and ordered array implementation

<table>
<thead>
<tr>
<th>operation</th>
<th>argument</th>
<th>return value</th>
<th>size</th>
<th>contents (unordered)</th>
<th>contents (ordered)</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>P</td>
<td>1</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>insert</td>
<td>Q</td>
<td>2</td>
<td>P Q</td>
<td>P Q</td>
<td>P Q</td>
</tr>
<tr>
<td>insert</td>
<td>E</td>
<td>3</td>
<td>P Q E</td>
<td>E P Q</td>
<td>E P Q</td>
</tr>
<tr>
<td>remove max</td>
<td>Q</td>
<td>2</td>
<td>P E</td>
<td>E P</td>
<td>E P</td>
</tr>
<tr>
<td>insert</td>
<td>X</td>
<td>3</td>
<td>P E X</td>
<td>E P X</td>
<td>E P X</td>
</tr>
<tr>
<td>insert</td>
<td>A</td>
<td>4</td>
<td>P E X</td>
<td>A E P X</td>
<td>A E P X</td>
</tr>
<tr>
<td>insert</td>
<td>M</td>
<td>5</td>
<td>P E X A M</td>
<td>A E M P X</td>
<td>A E M P X</td>
</tr>
<tr>
<td>remove max</td>
<td>X</td>
<td>4</td>
<td>P E M A</td>
<td>A E M P</td>
<td>A E M P</td>
</tr>
<tr>
<td>insert</td>
<td>P</td>
<td>5</td>
<td>P E M A P</td>
<td>A E M P</td>
<td>A E M P</td>
</tr>
<tr>
<td>insert</td>
<td>L</td>
<td>6</td>
<td>P E M A P L</td>
<td>A E L M P</td>
<td>A E L M P</td>
</tr>
<tr>
<td>insert</td>
<td>E</td>
<td>7</td>
<td>P E M A P L E</td>
<td>A E E L M P</td>
<td>A E E L M P</td>
</tr>
<tr>
<td>remove max</td>
<td>P</td>
<td>6</td>
<td>E M A P L E</td>
<td>A E E L M P</td>
<td>A E E L M P</td>
</tr>
</tbody>
</table>

A sequence of operations on a priority queue
public class UnorderedArrayMaxPQ<Key extends Comparable<Key>>
{
    private Key[] pq; // pq[i] = ith element on pq
    private int N; // number of elements on pq

    public UnorderedArrayMaxPQ(int capacity)
    {  pq = (Key[]) new Comparable[capacity];  }

    public boolean isEmpty()
    {  return N == 0;  }

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

    public Key delMax()
    {
        int max = 0;
        for (int i = 1; i < N; i++)
            if (less(max, i)) max = i;
        exch(max, N-1);
        return pq[--N];
    }
}
Priority queue elementary implementations

Challenge. Implement all operations efficiently.

<table>
<thead>
<tr>
<th>implementation</th>
<th>insert</th>
<th>del max</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>1</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>ordered array</td>
<td>$N$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>goal</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
</tr>
</tbody>
</table>

order of growth of running time for priority queue with $N$ items
2.4 Priority Queues

- API and elementary implementations
- Binary heaps
- Heapsort
- Event-driven simulation
Complete binary tree

**Binary tree.** Empty or node with links to left and right binary trees.

**Complete tree.** Perfectly balanced, except for bottom level.

![Complete tree with 16 nodes (height = 4)](image)

**Property.** Height of complete tree with $N$ nodes is $\lfloor \lg N \rfloor$.

**Pf.** Height increases only when $N$ is a power of 2.
A complete binary tree in nature

Hyphaene Compressa - Doum Palm

© Shlomit Pinter
Binary heap representations

**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!
Binary heap properties

**Proposition.** Largest key is $a[1]$, which is root of binary tree.

**Proposition.** Can use array indices to move through tree.
- Parent of node at $k$ is at $k/2$.
- Children of node at $k$ are at $2k$ and $2k+1$.

- left subtree of $k$ is empty if $2k > N$.
- right subtree of $k$ is empty if $(2k+1) > N$.
- $k$ is a leaf node if $2k > N$. 

Heap representations