CS2010: ALGORITHMS AND DATA STRUCTURES

Lecture 9: Priority Queues

Vasileios Koutavas

21 Oct 2015

School of Computer Science and Statistics
Trinity College Dublin

http://algs4.cs.princeton.edu

2.4 PRIORITY QUEUES

- *API and elementary implementations*
- *binary heaps*
- *heapsort*
- *event-driven simulation*

http://algs4.cs.princeton.edu

2.4 PRIORITY QUEUES

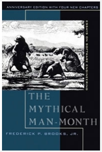
- *API and elementary implementations*
- *binary heaps*
- *heapsort*
- *event-driven simulation*

Collections

A collection is a data types that store groups of items.

data type	key operations	data structure
stack	Push, Pop	*linked list, resizing array*
queue	Enqueue, Dequeue	*linked list, resizing array*
priority queue	Insert, Delete-Max	*binary heap*
symbol table	Put, Get, Delete	*BST, hash table*
set	Add, Contains, Delete	*BST, hash 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.

operation	*argument*	*return value*
insert	P	
insert	Q	
insert	E	
remove max		Q
insert	X	
insert	A	
insert	M	
remove max		X
insert	P	
insert	L	
insert	E	
remove max		P

Priority queue API

Requirement. Generic items are `Comparable`.

Key must be Comparable
(bounded type parameter)

```
public class MaxPQ<Key extends Comparable<Key>>
```

	`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.

Priority queue client example

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

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

N huge, M large

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
```

sort key

Priority queue client example

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

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

N huge, M large

Constraint. Not enough memory to store N items.

```
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();
}
```

use a min-oriented pq

Transaction data type is Comparable (ordered by $$)

pq contains largest M items

Priority queue client example

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

implementation	time	space
sort	$N \log N$	N
elementary PQ	$M N$	M
binary heap	$N \log M$	M
best in theory	N	M

order of growth of finding the largest M in a stream of N items

Priority queue: unordered and ordered array implementation

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	P	
insert	E		7	P	E	M	A	P	L	E	A	E	E	L	M	P	P
remove max		P	6	E	M	A	P	L	E		A	E	E	L	M	P	

A sequence of operations on a priority queue

Priority queue: unordered array implementation

```
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];
   }
}
```

no generic array creation

less() and exch() similar to sorting methods (but don't pass pq[])

should null out entry to prevent loitering

Priority queue elementary implementations

Challenge. Implement all operations efficiently.

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
goal	$\log N$	$\log N$	$\log N$

order of growth of running time for priority queue with N items

http://algs4.cs.princeton.edu

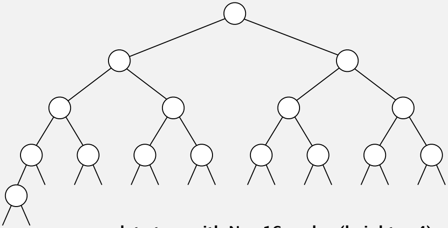
2.4 Priority Queues

- *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 N = 16 nodes (height = 4)

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

Binary heap representations

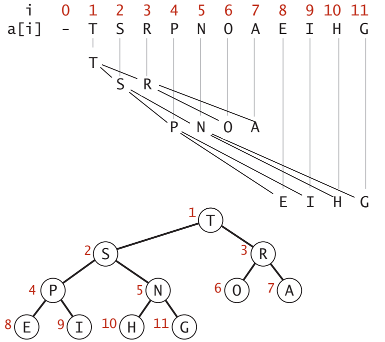
Binary heap. Array representation of a heap-ordered complete binary tree.

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!



Heap representations

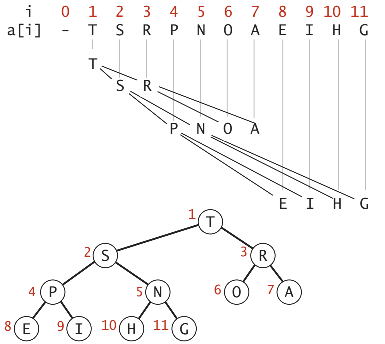
Binary heap properties

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