Priority Queues

Data. Items that can be compared.

Basic operations.
- Insert.
- Remove largest.
- Copy.
- Create.
- Destroy.
- Test if empty.

Priority Queue Applications

Applications:
- Event-driven simulation.
- Numerical computation.
- Data compression.
- Graph searching.
- Computational number theory.
- Artificial intelligence.
- Statistics.
- Operating systems.
- Discrete optimization.
- Spam filtering.

Customers in a line, colliding particles
reducing roundoff error
Huffman codes
Dijkstra’s algorithm, Prim’s algorithm
sum of powers
A* search
maintain largest M values in a sequence
load balancing, interrupt handling
bin packing, scheduling
Bayesian spam filter

Generalizes: stack, queue, randomized queue.

Priority Queue Client Example

Problem: Find the largest M of a stream of N elements.
- Fraud detection: isolate $$\$\$$ transactions.
- File maintenance: find biggest files or directories.

Constraint. Not enough memory to store N elements.
Solution. Use a priority queue.

<table>
<thead>
<tr>
<th>Operation</th>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>( N \lg N )</td>
<td>( N )</td>
</tr>
<tr>
<td>elementary PQ</td>
<td>( MN )</td>
<td>( M )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( N \lg M )</td>
<td>( M )</td>
</tr>
<tr>
<td>best in theory</td>
<td>( N )</td>
<td>( M )</td>
</tr>
</tbody>
</table>

```java
MaxPQ<String> pq = new MaxPQ<String>();
while(!StdIn.isEmpty()) {
    String s = StdIn.readString();
    pq.insert(s);
    if (pq.size() > M) pq.delMax();
}
while (!pq.isEmpty())
    System.out.println(pq.delMax());
```
**Priority Queue: Elementary Implementations**

*Challenge.* Implement both operations efficiently.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Insert</th>
<th>Delmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>ordered array</td>
<td>N</td>
<td>1</td>
</tr>
</tbody>
</table>

worst-case asymptotic costs for PQ with N items.

**Priority Queue: Unordered Array Implementation**

```java
public class UnorderedPQ<Item extends Comparable> {
    private Comparable[] pq;
    private int N;

    public UnorderedPQ(int maxN) { pq = new Comparable[maxN]; }

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

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

    public Item delMax() {
        int max = 0;
        for (int i = 1; i < N; i++)
            if (less(max, i)) max = i;
        exch(max, N-1);
        return (Item) pq[--N];
    }
}
```

**Binary Heap**

**Heap:** Array representation of a heap-ordered complete binary tree.

**Binary tree.**
- Empty or
- Node with links to left and right trees.

**Heap-ordered binary tree.**
- Keys in nodes.
- No smaller than children’s keys.

**Array representation.**
- Take nodes in level order.
- No explicit links needed since tree is complete.

**Binary Heap Properties**

**Property A.** Largest key is at root.

**Property B.** Can use array indices to move through tree.
- Note: indices start at 1.
- Parent of node at k is at k/2.
- Children of node at k are at 2k and 2k+1.

**Property C.** Height of heap is \( h = 1 + \lfloor \log N \rfloor \).
- Level \( i \) has at most \( 2^i \) nodes.
  \[ 1 + 2 + 4 + \ldots + 2^{h-1} \geq N. \]
Promotion In a Heap

**Scenario.** Exactly one node is bigger than its parent.

**To eliminate the violation:**
- Exchange with its parent.
- Repeat until heap order restored.

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

Peter principle: node promoted to level of incompetence.

Demotion In a Heap

**Scenario.** Exactly one node is smaller than a child.

**To eliminate the violation:**
- Exchange with larger child.
- Repeat until heap order restored.

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

Power struggle: better subordinate promoted.

Insert

**Insert.** Add node at end, then promote.

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

Remove the Maximum

**Remove max.** Exchange root with node at end, then demote.

```java
public Item delMax() {   Item max = (Item) pq[1];   exch(1, N--);   sink(1);   pq[N+1] = null;   return max;}
```
Binary Heap: Skeleton

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

  public MaxPQ(int maxN) {}  
  private void build() {}   
  public MaxPQ(int maxN) { } same as array-based PQ,
  public boolean isEmpty() { } but allocate one extra element in array
  public void insert(Item x) {} PQ ops
  public Item delMax() {} 
  public void swim(int k) {} heap helper functions
  private void sink(int k) {} array helper functions
  private boolean less(int i, int j) {}  
  private void each(int i, int j) {} 
}
```

Priority Queues Implementation Cost Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Insert</th>
<th>Remove Max</th>
<th>Find Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>ordered array</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ordered list</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>unordered array</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>unordered list</td>
<td>1</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>binary heap</td>
<td>(\lg N)</td>
<td>(\lg N)</td>
<td>1</td>
</tr>
</tbody>
</table>

worst-case asymptotic costs for PQ with \(N\) items

Hopeless challenge: get all ops \(O(1)\). Why hopeless?

Minimum oriented priority queue. Replace `less` with `greater` and implement `greater`.

Array resizing. Support no-argument constructor, and implement repeated doubling so that operations take \(O(\log N)\) amortized time.

Immutability of keys. We assume client does not change keys while they’re on the PQ. It’s a good idea for client to use immutable objects.

Other operations.
- Remove an arbitrary item.
- Change the priority of an item.
- Can implement using `sink` and `swim` abstractions, but we defer.

Digression: Heapsort

First pass: build heap.
- Insert items into heap, one at a time.
- Or can use faster bottom-up method; see book.

```java
for (int k = N / 2; k >= 1; k--)
  sink(a, k, N);
```

Second pass: sort.
- Remove maximum items, one at a time.
- Leave in array, instead of nulling out.

```java
while (N > 1) {
  each(a, 1, N--);
  sink(a, 1, N);
}
```

Property D. At most \(2N \lg N\) comparisons.
Significance of Heapsort

Q: Sort in O(N log N) worst-case without using extra memory?
A: Yes. Heapsort.

Not mergesort? Linear extra space.
Not quicksort? Quadratic time in worst case.

Heapsort is optimal for both time and space, but:
- Inner loop longer than quicksort’s.
- Makes poor use of cache memory.

In the wild: g++ STL uses introsort.

1
combo of quicksort, heapsort, and insertion

A* Algorithm

<table>
<thead>
<tr>
<th>In-Place</th>
<th>Stable</th>
<th>Worst</th>
<th>Average</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble sort</td>
<td>X</td>
<td>X</td>
<td>N² / 2</td>
<td>N² / 2</td>
</tr>
<tr>
<td>Selection sort</td>
<td>X</td>
<td></td>
<td>N² / 2</td>
<td>N² / 2</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>X</td>
<td>X</td>
<td>N² / 2</td>
<td>N² / 4</td>
</tr>
<tr>
<td>Shellsort</td>
<td>X</td>
<td></td>
<td>N²/2</td>
<td>N²/2</td>
</tr>
<tr>
<td>Quicksort</td>
<td>X</td>
<td></td>
<td>N² / 2</td>
<td>2N log N</td>
</tr>
<tr>
<td>Mergesort</td>
<td>X</td>
<td></td>
<td>N log N</td>
<td>N log N</td>
</tr>
<tr>
<td>Heapsort</td>
<td>X</td>
<td></td>
<td>2 N log N</td>
<td>2 N log N</td>
</tr>
</tbody>
</table>

Sam Loyd’s 15-Slider Puzzle

15 puzzle.
- Legal move: slide neighboring tile into blank square.
- Challenge: sequence of legal moves to put tiles in increasing order.
- Win $1,000 prize for solution.

http://www.jossenthebresk.com/puzz15/
Breadth First Search of 8-Puzzle Game Tree

Priority first search.
- Basic idea: explore positions in a more intelligent order.
- Ex 1: number of tiles out of order.
- Ex 2: sum of Manhattan distances + depth.

Implement A* algorithm with PQ.

Slider Puzzle: Unsolvable Instances

Unsolvable instances.

8-slider invariant. Parity of number of pairs of pieces in reverse order.

Event-Driven Simulation
Molecular Dynamics Simulation of Hard Spheres

**Goal.** Simulate the motion of N moving particles that behave according to the laws of elastic collision.

**Hard sphere model.**
- Moving particles interact via elastic collisions with each other, and with fixed walls.
- Each particle is a sphere with known position, velocity, mass, and radius.
- No other forces are exerted.

**Significance.** Relates macroscopic observables to microscopic dynamics.
- Maxwell and Boltzmann: derive distribution of speeds of interacting molecules as a function of temperature.
- Einstein: explain Brownian motion of pollen grains.

---

Time-Driven Simulation

**Main drawbacks.**
- \( N^2 \) overlap checks per time quantum.
- May miss collisions if \( dt \) is too large and colliding particles fail to overlap when we are looking.
- Simulation is too slow if \( dt \) is very small.

---

Event-Driven Simulation

**Event-driven simulation.**
- Between collisions, particles move in straight-line trajectories.
- Focus only on times when collisions occur.
- Maintain priority queue of collision events, prioritized by time.
- Remove the minimum \( = \) get next collision.

**Collision prediction.** Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?

**Collision resolution.** If collision occurs, update colliding particle(s) according to laws of elastic collisions.
Particle-Particle Collision Prediction

Collision prediction.
- Particle $i$: radius $r_i$, position $(r_{xi}, r_{yi})$, velocity $(v_{xi}, v_{yi})$.
- Particle $j$: radius $r_j$, position $(r_{xj}, r_{yj})$, velocity $(v_{xj}, v_{yj})$.
- Will particles $i$ and $j$ collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } v_{yj} = 0 \\ \frac{(r_{yj} - r_{yi})}{v_{yj}} & \text{if } v_{yj} < 0 \\ \frac{(1 - r_{yj} - r_{yi})}{v_{yj}} & \text{if } v_{yj} > 0 \end{cases}$$

Collision resolution. $(v_{xj}', v_{yj}') = (v_{xj}, -v_{yj})$.

Particle-Wall Collision

Collision prediction.
- Particle of radius $r$ at position $(r_x, r_y)$, moving with velocity $(v_x, v_y)$.
- Will it collide with a horizontal wall? If so, when?

$$v' = \frac{r}{v_y} \text{ if } v_y > 0$$

Collision resolution. $(v_x', v_y') = (v_x, -v_y)$.

Particle-Particle Collision Prediction

Collision prediction.
- Particle $i$: radius $r_i$, position $(r_{xi}, r_{yi})$, velocity $(v_{xi}, v_{yi})$.
- Particle $j$: radius $r_j$, position $(r_{xj}, r_{yj})$, velocity $(v_{xj}, v_{yj})$.
- Will particles $i$ and $j$ collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } v_{yj} = 0 \\ \frac{\Delta v \cdot \Delta t}{\Delta v \cdot \Delta t + \sqrt{d}} & \text{if } d < 0 \\ \frac{\Delta v \cdot \Delta t}{\Delta v \cdot \Delta t - \Delta z^2} & \text{otherwise} \end{cases}$$

$$d = (\Delta v \cdot \Delta t)^2 - (\Delta v \cdot \Delta t) (\Delta r \cdot \Delta t) - \Delta z^2$$

$$\Delta v = (\Delta v_x, \Delta v_y) = (v_x - v_{xj}, v_y - v_{yj})$$

$$\Delta r = (\Delta r_x, \Delta r_y) = (r_x - r_{xj}, r_y - r_{yj})$$

$$\Delta r \cdot \Delta r = (\Delta r_x)^2 + (\Delta r_y)^2$$

Collision resolution. When two particles collide, how does velocity change?

$$v_x' = v_x + J x / m_i$$
$$v_y' = v_y + J y / m_i$$
$$v_x' = v_x - J x / m_j$$
$$v_y' = v_y - J y / m_j$$

Impulse due to normal force:
- Conservation of energy.
- Conservation of momentum.

$$J_x = \frac{J \Delta r_x}{\sigma}, \quad J_y = \frac{J \Delta r_y}{\sigma}, \quad J = \frac{2m_i m_j (\Delta v \cdot \Delta t)}{\sigma(m_i + m_j)}$$
Event-Driven Simulation

Initialization. Fill PQ with all potential particle-wall and particle-particle collisions.

Main loop.
- Delete the impending event from PQ (min priority = t).
- If the event is no longer valid, ignore it.
- Advance all particles to time t, on a straight-line trajectory.
- Update the velocities of the colliding particle(s).
- Predict future particle-wall and particle-particle collisions involving the colliding particle(s) and insert events onto PQ.

potential since collision may not happen if some other collision intervenes.