Topic 24
Heaps

"You think you know when you can learn, are more sure when you can write even more when you can teach, but certain when you can program."
- Alan Perlis

Priority Queue

- Recall priority queue
  - elements enqueued based on priority
  - dequeue removes the highest priority item

- Options?
  - List? Binary Search Tree?

  Linked List enqueue  BST enqueue
  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>O(N)</td>
</tr>
<tr>
<td>B</td>
<td>O(N)</td>
</tr>
<tr>
<td>C</td>
<td>O(N)</td>
</tr>
<tr>
<td>D</td>
<td>O(logN)</td>
</tr>
<tr>
<td>E</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Another Option

- A heap
  - not to be confused with the runtime heap (portion of memory for dynamically allocated variables)

- A complete binary tree
  - all levels have maximum number of nodes except deepest where nodes are filled in from left to right

- Maintains the heap order property
  - in a min heap the value in the root of any subtree is less than or equal to all other values in the subtree

Clicker Question 2

- In a max heap with no duplicates where is the largest value?
  
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
</tbody>
</table>
Example Min Heap

12
/   \
17   16
/ |  / |
19 52 37 25
/ \
21 45

Internal Storage

- Interestingly heaps are often implemented with an array instead of nodes

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>17</td>
<td>16</td>
<td>19</td>
<td>52</td>
<td>37</td>
<td>25</td>
<td>21</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for element at position $i$:
- parent index: $i / 2$
- left child index: $i * 2$
- right child index: $i * 2 + 1$

Enqueue Operation

- Add new element to next open spot in array
- Swap with parent if new value is less than parent
- Continue back up the tree as long as the new value is less than new parent node

Enqueue Example

- Add 15 to heap (initially next left most node)
Enqueue Example

\* Swap 15 and 52

\* Swap 15 and 17, then stop

PriorityQueue Class

```java
public class PriorityQueue<E extends Comparable<E>> {

    private int size;
    private E[] con;

    public PriorityQueue() {
        heap = getArray(2);
    }

    private E[] getArray(int size) {
        return (E[]) (new Comparable[size]);
    }

    public void enqueue(E val) {
        if (size == con.length - 1) {
            con = expandArray(con.length * 2 + 1);
        }
        size++;
        int indexToPlace = size;
        while (indexToPlace > 1 && con[indexToPlace / 2].compareTo(val) > 0) {
            con[indexToPlace] = con[indexToPlace / 2];
            indexToPlace /= 2;
        }
        con[indexToPlace] = val;
    }

    private void expandArray(int newSize) {
        E[] temp = getArray(newSize);
        System.arraycopy(con, 1, temp, 1, size);
        con = temp;
    }
}
```

PriorityQueue enqueue

```java
public void enqueue(E val) {
    if (size == con.length - 1) {
        con = expandArray(con.length * 2 + 1);
    }
    size++;
    int indexToPlace = size;
    while (indexToPlace > 1 && con[indexToPlace / 2].compareTo(val) > 0) {
        con[indexToPlace] = con[indexToPlace / 2];
        indexToPlace /= 2;
    }
    con[indexToPlace] = val;
}
```

PriorityQueue Enqueue Example

\* Swap 15 and 52

\* Swap 15 and 17, then stop
Dequeue

- min value / front of queue is in root of tree
- swap value from last node to root and move down swapping with smaller child unless values is smaller than both children
Deque Example

- Swap 35 with smaller child (17)

```
15
  /  \
17   16
 /  \
35  23 37 25
 |    |    |
21  45
```

Deque Code

```java
public E dequeue() {
    E top = con[1];
    int hole = 1;
    boolean done = false;
    while ( hole * 2 < size && !done ) {
        int child = hole * 2;
        // see which child is smaller
        if ( con[child].compareTo( con[child + 1] ) > 0 ) {
            child++;
            // child now points to smaller
        }
        // is replacement value bigger than child?
        if (con[size].compareTo( con[child] ) > 0 ) {
            con[hole] = con[child];
            hole = child;
        }
        else {
            done = true;
        }
    }
    con[hole] = con[size];
    size--;
    return top;
}
```

PriorityQueue Comparison

- Run a Stress test of PQ implemented with Heap and PQ implemented with BinarySearchTree
- What will result be?
  A. Heap takes half the time or less of BST
  B. Heap faster, but not twice as fast
  C. About the same
  D. BST faster, but not twice as fast
  E. BST takes half the time or less of Heap
Data Structures

- Data structures we have studied
  - arrays, array based lists, linked lists, maps, sets, stacks, queue, trees, binary search trees, graphs, hash tables, red-black trees, priority queues, heaps
- Most program languages have some built in data structures, native or library
- Must be familiar with performance of data structures
  - best learned by implementing them yourself

We have not covered every data structure

- Abstract data types
  - Array
    - Linear list
      - Array
      - Bit array
      - Bit vector
      - Bit mask
    - Circular buffer
      - Constant
      - Image
      - Dynamic array
      - Global buffer
      - Shared array
      - Uniform array
      - High memory
      - Lookup table
    - Matrix
    - Parallel array
    - Serial array
    - Sparse array
    - Sparse matrix
    - Life matrix
    - Variable-length array
  - List
    - Doubly linked list
    - Lined list
    - Suffix list
    - Skip list
    - Unsorted linked list
    - List
      - For linked list
      - Interleaved list
    - Stack
    - Queue
    - Doubly connected list
    - Difference list

Some properties of abstract data types:

<table>
<thead>
<tr>
<th>Structure</th>
<th>Container</th>
<th>Map/Associative array/Dictionary</th>
<th>Multimap</th>
<th>List</th>
<th>Set</th>
<th>Multiset</th>
<th>Priority queue</th>
<th>Queue</th>
<th>Deque</th>
<th>Stack</th>
<th>String</th>
<th>Tree</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bag (multiset)</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Stable" means that input order is retained. Other properties may be specified.

Heaps

- Graph
- Adjacency list
- Adjacency matrix
- Graph-structured stack
- Scene graph
- Binary decision diagram
- Zero suppressed decision diagram
- And-inverter graph
- Directed graph
- Directed acyclic graph
- Propositional directed acyclic graph
- Multigraph
- Hypergraph

Other

- Lightmap
- Weighted edge
- Doubly connected edge list
- Quad-edge
- Rosting table
- Symbol table

Data Structures

- deque, b-trees, quad-trees, binary space partition trees, skip list, sparse list, sparse matrix, union-find data structure, Bloom filters, AVL trees, trie, 2-3-4 trees, and more!
- Must be able to learn new and apply new data structures