Midterm review

Exam day: March 23, 2017, in class (Your TA will proctor the exam)

Guide line: Read lecture notes and related chapters. On the exam you will have to write some (correct) code and read some code (perhaps say what would be displayed by some code or say what values are left in some variables). You may also have some short answer questions which you will have to answer in English.

Topics (scope of the midterm exam): The following list is intended as a study aid, This is not as a complete list of material covered in the course up to this point.

Weiss, Data Structures & Problem Solving Using Java, Chapters 7, 15, 16, 17, 18, 19, 20, 21

- Data structure design
  - generic classes, recursion,
    asymptotic notations
- List
  - array and linked lists
- Stack and Queue
- Tree
  - BST, AVL
- Heap
  - Priority queue
- Hash
  - Hash function, collision handling
**Data structure design - generic classes**

Background: What do I do if I need a list (tree, stack, heap, hash) of strings (integers, shapes, colors, records, trees, graphs)? Do I need to re-design the whole list (or any data structures)?

What are the tools that can help you solve/address this question?

**Data structure design - recursion** (this is more an algorithm design technique)

We talked about Fibonacci numbers and looked at two implementations. One is using recursion and the other one is using dynamic programming. What are the other recursive functions that we studied?

Recursion is usual and elegant but need to be very very careful when you use it.

**Data structure design - Asymptotic notations**

Why do computer scientists use asymptotic notations?

Big O, Big Omega, Order of growth

```
Algorithm 3.3: SEARCH(A[1..n], K)

for i ← [1 · · · n]
do { if A[i] = K then return (i)
return (-1)
```

what is the time complexity of this algorithm?
List - array and linked lists

- What are the differences? (think this problem for ALL data structures you have learnt so far!!)
  - insert
    ‣ to the end
    ‣ to the front
  - delete
    ‣ from the end
    ‣ from the front
  - find
  - find min/max/i-th largest
  - build

List - stack and queue

- What is a stack and what are the applications?
- What is a queue and what are the applications?
**Tree - Definitions**

What is a tree?

What is NOT a tree?

What are depth, height, path, path length, root, leaves, parent, children?

What can be used to represent a tree?

**Tree - Traverse**

Traverse orders: Preorder, in order, post order

How do you implement these traverse orders? Why do we need to traverse a tree?

**Tree - BST (Binary search tree)**

What is a BST?

What binary tree is NOT a BST?

Why is BST useful?

What does NOT BST made for? (in terms of operations)
Tree - BST (Binary search tree)

How do insertion and deletion work?

**Example:** Before inserting 5

**Example:** Delete node 4 with one child: Make node 2 point to node 3

**Example:** Delete node 2 with two children: Replace value 2 with 3; delete 3 from 5’s tree.
**Tree - AVL tree**

What is AVL tree?

Why do we need AVL tree?

How do you insert/delete nodes in AVL tree?

**Tree - AVL tree - Single rotation**

![Single rotation diagram]

**Tree - AVL tree - Double rotation (why do we need to do double rotation?)**

![Double rotation diagram]

**Tree - Heap**

What is a heap? (What is not a heap)?

Why do we need heap? What is a heap good (and bad) for?

What kind of data structure can be used to represent a heap?

How do you insert and remove (and update) an element from a heap?

How much time does it take to (1) insert, delete, update an element and (2) build the heap?

Example: Insert 14
Hash

Why do we need hash?
What is the difference between direct access table and hash?
What are the most important properties when creating a hash table?
What is the time complexity for insertion/deletion/find

Hash - Hash function
The easiest hash function $h$ is modulo (usually by $TableSize$)
It is generally a good idea that $TableSize$ is not divisible by small numbers.
$TableSize$ is usually a prime number.

Hash - Open hashing

Table:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>81</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>49</td>
<td>9</td>
</tr>
</tbody>
</table>
Hash - Closed hashing

<table>
<thead>
<tr>
<th>Empty Table</th>
<th>After 89</th>
<th>After 18</th>
<th>After 49</th>
<th>After 58</th>
<th>After 69</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>49</td>
<td>49</td>
<td>49</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>58</td>
<td>58</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>

How do you resolve collision?
- linear probing
- quadratic probing

How do you delete an element?