Reminders and Topics

- **Project 11 is officially due Apr 27 (but we set autograder to accept submissions till Apr 29)**
- This lecture:
  - **More on Hash Table and Final Review**

Converting a String to a Key

- Say we want to use a hash table to store dictionary words. How do we compute the key for each word?
- Assume word is made of low-case letters (‘a’ to ‘z’), and we ignore spaces, hyphens etc.
- The basic requirement is that each word must convert to a unique key / integer. No two words should have the same key.
  - ‘cat’ -> ?
  - ‘science’ -> ?
  - ‘computer’ -> ?

Converting a String to a Key

- Think of each word as a ‘number’, except this is a base-27 number system (empty or space corresponds to 0):
  - ‘ ‘->0, ’a’->1, ‘b’->2, ‘c’->3, … …, ‘y’->25, ‘z’->26
  - So each letter in ‘cat’ would translate to [3] [1] [20]. What number is this?

- How does number systems work in general?
  - Decimal (base-10): $(321)_{10} = 3x10^2+2x10^1+1x10^0=321$
  - Binary (base-2): $(1011)_2 = 1x2^3+0x2^2+1x2^1+1x2^0=11$
  - Hex (base-16): $(2DFA)_{16} = 2x16^3+13x16^2+15x16^1+10x16^0 = 11770$

Converting a String to a Key

- Another representation (suitable for programming):
  - Decimal (base-10):
    $(321)_{10} = (3x10+2)x10+1=3x10^2+2x10^1+1x10^0$
  - Binary (base-2):
    $(1011)_2 = ((1x2+0)x2+1)x2+1 =1x2^3+0x2^2+1x2^1+1x2^0$
  - Hex (base-16):
    $(2DFA)_{16} = ((2x16+13)x16+15)x16+10 =$
    $2x16^3+13x16^2+15x16^1+10x16^0$
  - Ours (base-27): ('cat' = [3] [1] [20])
    $(cat)_{27} = (3x27+1)x27+20=3x27^2+1x27^1+20x27^0$

Converting a Word to a Hash Value

```
// convert word (made of lower-case
// letters) to an integer key, then
// compute the hash index
public int word2hash(String word,
                     int arraySize) {
  int key = 0;
  for(int j=0; j<word.length(); j++) {
    // 97 is the ascii code for ‘a’
    key = key*27 + (word.get(j)-96);
  }
  return key % arraySize;
}
```

Converting a String to a Key

- So each word converts to a unique key. But does each key value convert back to a valid word?
- What's the maximum possible value for a 8-letter stirng?

  $27^8 - 1 = 282,429,536,480$

  This is way more than the number of valid words! But it doesn't matter, because our hashing function will map it to an index at a much smaller range!
- There is a subtle problem here. What is it?

  Such a large integer exceeds the largest integer value in Java! In practice, we can apply the modulo operator inside the for loop (leveraging the properties of %).

Converting a Word to a Hash Value

```
// compute the hash index of a word
public int word2hash(String word,
                     int arraySize) {
  int key = 0;
  for(int j=0; j<word.length(); j++) {
    key = key*27 + (word.get(j)-96);
    key = key % arraySize;
  }
  return key;
}
// this works due to (a+b)%p = (a%p+b)%p
```

Java's hashCode() method

- All objects in Java has a **`hashCode()`** method that returns a 32-bit integer value. The requirement is:
  - Calling **`hashCode()`** on the same object must consistently return the same hash value.
  - If two objects are equal (w.r.t. **`equals()`** method), they must return the same hash value. However, two objects that are unequal may return the same hash value!
- By default, it's implemented by converting the object's memory address to a hash value. You can customize it (e.g. String class has a custom hashCode method).

Java's Hashtable class

- Java provides a `class Hashtable<K, V>` where K is key's type, V is the value's type
- It implements the `Map<K, V>` interface, and can re-hash dynamically based on a pre-defined load factor (e.g. 0.75)
- For example:

```
Hashtable<String, Float> table =
     new Hashtable<String, Float>();

table.put("John Smith", 6.0f);
table.put("Eric May", 5.8f);
table.put("Rose Ann", 5.9f);
System.out.println(table.get("Rose Ann"));
```

Final Exam

- Monday, May 2, 6-8pm, Totman Gym
- Cumulative, but will emphasize more on the second half of the semester (starting from BST)
- Same format as before: 20 MCs + 2 Programming Sections

Summary of What We've Learned

- **Fundamental Storage Data Structures**
  - Arrays
  - Linked Lists
  - Trees
- **High-Level, Abstract Data Structures**
  - List (sorted, unsorted), Stack, Queue
  - BST (Binary Search Tree)
  - Heap, Priority Queue
  - Graph
  - Hash Table

Comparison

data structure	add	search	remove
unsorted array	O(1)	O(N)	O(N)
sorted array	O(N)	O(log N)	O(N)
BST (avg / worst)	O(log N) / O(N)	O(log N) / O(N)	O(log N) / O(N)
balanced BST	O(log N)	O(log N)	O(log N)
heap	O(log N)	O(log N)	O(log N)
hash table	O(1)	O(1)	O(1)

Summary of What We've Learned

- **Fundamental Algorithms**
  - Recursion (three conditions of recursion)
  - Sorting (simple sorting, merge sort, heap sort, quick sort)
  - Graph Search (BFS and DFS)
  - Big-O Notation and Algorithm Analysis
- **Java-Specific Knowledge**
  - Classes, Interfaces, Exceptions, Generics, Iterators
  - Java provides implementations of many data structures we've learned (ArrayList, Stack, Queue…)

How to Prepare

- Study Lecture **Slides** and Textbook
  - Some algorithms presented in slides are different from the textbook (to provide you with more efficient / elegant solutions). So study lecture slides first. If anything is unclear, refer to the textbook
    - Make sure you understand all the **clicker** questions
- Study the two **Midterms** and the **Practice Final** Exam.
- Study discussion materials
- Study the projects

Second Half of the Semester

- **Binary Tree and BST**
  - Data structure of a binary tree
  - Binary tree terminologies
  - What's a full tree? What's a complete tree?
  - Tree traversals (in-order, post-order, pre-order)
  - What is a BST? What are its properties?
  - BST and in-order traversal.
  - How to perform search, insertion, deletion from BST? What is the cost of each of them? What about worst-case cost?

Second Half of the Semester

- **More on Binary Tree and BST**
  - What do (in-order) predecessor and successor mean? Write down code to find them.
  - How to create a balanced BST from a sorted array?
  - What is a self-balancing BST? What guarantees does it make?
  - Scapegoat tree: how does it balance the BST during insertion and deletion?
  - How to store binary tree in an array? Given a node at index i, where are its two children, and its parent?

Second Half of the Semester

- **Heap and Priority Queue**
  - What is priority queue? How is it different from a standard queue?
  - What is a heap? What are its properties? How is it different from a BST? How is a heap stored?
  - How to perform enqueue and dequeue in a heap?
    - Understand bubbleUp and bubbleDown, and be able to write down code for them.
  - What are the costs of enqueue and dequeue with a heap? How does this compare to using sorted array?

Second Half of the Semester

- **Graphs**
  - Graph terminologies
  - Directed / Undirected, Connected / Non-connected, Weighted / Unweighted
  - How is a graph typically stored?
  - Graph traversal
    - Depth-First Search (DFS): implementation
    - Breadth-First Search (BFS): implementation
    - Shortest-distance in a weighted graph (uniform cost search)

Second Half of the Semester

- **Simple Sorting (quadratic cost)**
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
- **Advanced Sorting (log linear cost)**
  - Merge Sort and the merge() operation
  - Quick Sort and the partition() operation
  - Heap Sort and the heapify() operation
  - Among them, Quick Sort does not guarantee log linear cost in the worst-case scenario!

Second Half of the Semester

- **Hash Table**
  - What is a hashing function and what is a hash table?
  - What are the advantages / disadvantages of hash table compared to other data structures?
  - Properties of the modulo operator
  - What is collision? How to handle collision?
    - Open addressing: linear, quadratic probing, double hashing
    - Separate chaining
  - How to compute the hash code?

Sample Programming Questions

- Find the index of the smallest number in a unsorted array.
- Implement binary search in a sorted array.
- Print out nodes in a binary tree in in-, post-, pre- order traversal.
- Search / Insertion in a BST
- Find the predecessor / successor of any node in a BST
- Implement bubbleUp / bubbleDown
- Check if a binary tree is a heap or not
- Print out vertices in a graph in DFS and BFS order
- Implement Bubble / Selection / Insertion sort
- Implement merge(), partition(), heapify() methods
- *Important to understand them, instead of just memorize them!*

Concluding Remarks and Questions