CSM 61B Spring 2017

Hashing & Heaps

Week 10: March 20, 2017

1 Motivation

1.1 (a) In the worst case, how long does it take to index into a linked list?

(b) In the worst case, how long does it take to index into an array?

(c) In the worst case, how long does it take to insert into a linked list?

(d) Assuming there's space, how long does it take to put a element in an array?

(e) What if we assume there is no more space in the array?

(f) Given what we know about linked lists and arrays, when would we choose to use one data structure over the other?

2 Hash Table Basics

2.1 Consider `BadHashMap`'s `put` implementation which assumes no collisions or negative hash codes.

```
public class BadHashMap<K, V> implements Map<K, V> {
    private V[] array;
    public void put(K key, V value) {
        this.array[key.hashCode() % this.array.length] = value;
    }
}
```

(a) Why do we use the `%` (modulo) operator?

(b) What are collisions? What data structure can we use to address them?

(c) Describe an implementation for `get` and `containsKey` assuming you applied the changes for handling collisions.

2.2 Consider a `HashMap<Integer, String>` with an underlying array of size 5. Draw the resulting structure after the following operations. `Integer::hashCode` returns the integer's value itself.

```
put(3, "monument");
put(8, "shrine");
put(3, "worker");
put(5, "granary");
put(13, "worker");
```

3 Hash Codes

3.1 Which of the following hashCodes are valid?

```
class Point {
    private int x;
    private int y;
    private static count = 0;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
        count += 1;
    }
```

(a)
```
    public void hashCode() {
        System.out.print(this.x + this.y);
    }
```

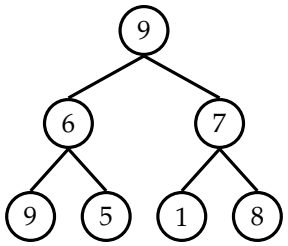
(b)
```
    public int hashCode() {
        Random random = new Random();
        return random.nextInt();
    }
```

(c)
```
    public int hashCode() {
        return this.x + this.y;
    }
```

(d)
```
    public int hashCode() {
        return count;
    }
```

(e)
```
    public int hashCode() {
        return 4;
    }
```

4 Min-Heapify This



4.1 (a) Show the heapification of the tree.

(b) Now, insert the value 2.

(c) Finally, remove the value 1.

5 Merging Sorted Lists *Extra for Experts*

5.1 Given the following sorted lists as inputs, how we can efficiently return a sorted list containing all the elements?

```
inputs: {1, 2, 3, 7}, {3, 4, 5, 6}, {2, 4, 6, 8}, {0, 1, 2, 4}, {1, 1, 6, 9}
output: {0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 6, 6, 6, 7, 8, 9}
```

5.2 Write `merge`, a procedure that merges K sorted lists of length N into a single sorted list in $O(NK \log K)$ time.

```
public class ListIterator<T extends Comparable<T>> implements Iterator<T>,
                                                              Comparable<ListIterator<T>> {
    private List<T> list;
    private int index;
    public boolean hasNext() {
        return index < list.size();
    }
    public T next() {
        T value = list.get(index);
        index += 1;
        return return value;
    }
    public T peek() {
        return list.get(index);
    }
    public int compareTo(ListIterator<T> other) {
        return this.peek().compareTo(other.peek());
    }
    public static <T extends Comparable<T>> List<T> merge(List<T>[] lists) {
        List<T> result = new ArrayList<>();
```

6 Caching *Extra for Experts*

6.1 You are a software engineer for a newspaper company! Your users are complaining about how slowly your website loads. After performing some performance profiling, you realize that the database queries are slowing the system down.

To fix the issue, you decide to implement a cache that contains the most recently accessed articles. The cache is only fast if it's small so you can only store a maximum of N articles. You want to keep only the N most recent articles that people have read. If a new, unique article is accessed, then the oldest article should be replaced.

Describe how you would implement this cache. What combinations of data structures would you use to build this efficiently?