Hashing Conclusions
Final Thoughts

cs206
April 30

Linear Probing

- Store only <K,V> at each location in array
  - No awkward linked lists
- If key is different and location is in use then go to next spot in array
  - if key is same, replace value
  - repeat until free location found

Probing Distance

- Given a hash value $h(x)$, linear probing generates $h(x),\ h(x)+1, h(x)+2, \ldots$
  - Primary clustering – the bigger the cluster gets, the faster it grows
- Quadratic probing – $h(x),\ h(x)+1, h(x)+4,\ h(x)+9, \ldots$
  - Quadratic probing leads to secondary clustering, more subtle, not as dramatic, but still systematic
- Double hashing
  - Use a second hash function to determine jumps

Performance Analysis for probing

- In the worst case, searches, insertions and removals take $O(n)$ time
  - when all the keys collide
- The load factor α affects the performance of a hash table
  - expected number of probes for an insertion with open addressing is $\frac{1}{1-\alpha}$
- Expected time of all operations is $O(1)$ provided α is not close to 1
    - NOTE: cheating here O() is about true worst case

Open Addressing vs Chaining

- Probing is significantly faster in practice
- locality of references – much faster to access a series of elements in an array than to follow the same number of pointers in a linked list
- Efficient probing requires soft/lazy deletions – tombstoning, why?
- de-tombstoning?

Performance of Hashtables

	Hash Expected	Hash Worst
search		
insert		
remove		
find min/ max		

	Unsorted array	Sorted array	Unsorted list	Sorted list	Tree (good)	Hash Expected
search						
insert						
remove						
find min/ max						

Using Hashtables

- No worries about hashing functions, rehashing, ...
  - Someone else responsibility
- Example: who is visiting my site, and how often?
  - for instance, hackers?
- web servers keep access logs

51.68.152.26 - - [31/Mar/2020:07:41:16 -0500] "GET / HTTP/1.1" 200 2372 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36 OPR/62.0.3331.99"

62.210.177.41 - - [31/Mar/2020:08:56:49 -0500] "GET /wp-json/wp/v2/users/ HTTP/1.1" 404 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36"

54.36.148.243 - - [31/Mar/2020:13:04:01 -0500] "GET /robots.txt HTTP/1.1" 404 - "-" "Mozilla/5.0 (compatible; AhrefsBot/6.1; +http://ahrefs.com/robot/)"

54.36.148.210 - - [31/Mar/2020:13:04:02 -0500] "GET /moon/14_1.jpg HTTP/1.1" 200 63064 "-" "Mozilla/5.0 (compatible; AhrefsBot/6.1; +http://ahrefs.com/robot/)"

Parsing a line

- A lot like the zip code task from the beginning of the semester

```
public class LogLine {
    /** The IP address extracted from the log line */
    private final String ipAddress;
    /** The line itself, stored here in case further processing is needed */
    private final String line;
    /** A counter, not properly a part of the line, but is data associaed with the line */
    private int count;
    public LogLine(String lin) throws Exception {
        if (lin==null || lin.length()==0)
            throw new Exception("Log lines should not be null or empty");
        line = lin;
        count = 1;
        String[] spl = lin.trim().split("\\s+");
        if (spl.length==0)
            throw new Exception("The line could not be split");
        ipAddress = spl[0];
    }
```

Read the file and accumulate data

```
public class LogAnalyzer {
   private HashMap<String, LogLine> lineMap;
   public LogAnalyzer() {
        lineMap = new HashMap<>();
    }
    public void readFileAndCount(String fileName) {
        try (BufferedReader br = new BufferedReader(new FileReader(fileName));) {
            String line;
            while ((null != (line=br.readLine()))) {
                LogLine ll = new LogLine(line);
                LogLine oll = lineMap.get(ll.getIP());
                if (oll!=null) {
                    oll.incCount();
                } else {
                    lineMap.put(ll.getIP(), ll);
                }
            }
        } catch (Exception eee) { // other exception handlers not shown
            System.err.println(eee.toString());
        }
    }
```

Print results

```
   public void printIPCount(int minCount) {
        ArrayList<LogLine> vvv = new ArrayList<LogLine>(lineMap.values());
        // if I wanted to sort, I now have the set in an array list,
        // from which sorting is fairly easy.
        int count=0;
        for (LogLine ll : vvv) {
            if (ll.getCount()>minCount) {
                System.out.println(ll.toStringLong());
                count++;
            }
        }
        System.out.println("Number of IPS seen " + lineMap.size());
        System.out.println("Number of IPS seen with count > " + minCount + ": "  + count);

    }
```

Run

```
    public static void main(String[] args) {
        LogAnalyzer la = new LogAnalyzer();
        la.readFileAndCount("fields43.com-Apr-2020");
        la.printIPCount(30);
    }
```

```
77.88.5.51 69
52.36.251.200 62
13.69.29.142 45
104.210.58.78 55
23.237.4.26 160
Number of IPS seen 893
Number of IPS seen with count > 30: 5
```

Course Goals (from day 1)

1. Become a better computer scientist
2. Learn about common data structures
   1. Implementation
   2. How and when to use each
3. Understand Object Oriented program design and its implementation in Java
4. Develop an understanding of UNIX
5. Become a better Java programmer