Hashing continued
Polynomial accumulation on Strings

```java
static int POLY_MULT=33;
public int stringHasher(String ss) {
    BigInteger ll = new BigInteger("0");
    for (int i=0; i<ss.length(); i++) {
        BigInteger bb = BigInteger.valueOf(POLY_MULT).pow(i).multiply(BigInteger.valueOf(((int)ss.charAt(i))));
        ll = ll.add(bb);
    }
    ll = ll.mod(BigInteger.valueOf(backingArray.length));
    return ll.intValue();
}
```

33^15 = 59938945498865420543457

Recommended by textbook

Handles really large numbers

Array storing the hashtable
Collisions

drawing 500 unique words from Oliver Twist and assuming a hashtable size of 1009, get these collisions

16 probable child when
42 fagins xxix importance that xv administering
104 stage pledge near
132 surgeon can night
271 things fang birth
341 alone sequel life
415 maylie check circumstances
418 mentioning containing growth
625 meet she first
732 there affording encounters
749 possible out acquainted
761 never xviii after goaded where
833 marks jew gentleman
985 adventures inseparable experience
Realistic hash codes computation in Java

• Use the hashcode function defined on Java Object.

• So put into hashtable is just

```java
private int h(Object k) {
    return k.hashCode() % backingArray.length;
}
public void put(Object key, Object value) {
    backingArray[h(key)] = value;
}
```
Collisions

• Handling of collisions is one of the most important topics for hashtables

• Rehashing
  • make the table bigger
  • O(n) time so want to avoid

• Alternative to rehashing
  • Separate Chaining
  • Probing
Separate Chaining

- Idea: each spot in hashtable holds a array list of key value pairs when the key maps to that hashvalue.
- Replace the item if the key is the same
- Otherwise, add to list
- Generally do not want more than about number of objects as size of table
- Chains can get long
Hash tables get crowded,
chains get long

HT_SIZE=1009

Using unique words drawn from “Oliver Twist”.
Unique count at top of table

<table>
<thead>
<tr>
<th>Unique Count</th>
<th>Hash Table 1</th>
<th>Hash Table 2</th>
<th>Hash Table 3</th>
<th>Hash Table 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>278</td>
<td>762</td>
<td>0</td>
<td>622</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>217</td>
<td>1</td>
<td>308</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>2</td>
<td>73</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

HT_SIZE=1009

<table>
<thead>
<tr>
<th>Unique Count</th>
<th>Hash Table 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>473</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
In class exercise

- Show the final contents of the hashtable using separate chaining assuming
  - table size is 7
  - \( h(t) = t \% 7 \)
- Data: <0,a> <32,b> <39,c> <12,d> <14,e> <35,f> <27,g> <13,h> <15,i> <5,j> <12,k> <13,l> <4,m> <0,n> <35,o>
- What is the longest chain?
Separate Chaining Code

SepChainHT.java
Open Addressing
Linear Probing

- Store only <K,V> at each location in array
- No awkward lists
- If key is different and location is in use then go to next spot in array
- repeat until free location found
- May need to wrap around end of array
In class exercise

• Show the final contents of the hashtable using linear probing assuming
  • table size is 13
  • \( h(t) = t \% 13 \)
  • Data: <0,a> <32,b> <39,c> <12,d> <14,e> <35,f> <27,g> <13,h> <15,i> <5,j> <12,k> <13,l> <4,m> <0,n> <35,o>
  • What is the most number of steps you needed to take to find a free location?
Probing Systems

- **Linear**
  - try: \( h(x), h(x)+1, h(x)+2, h(x)+3, \ldots \)
  - Primary clustering – the bigger the cluster gets, the faster it grows

- **Quadratic**
  - try: \( h(x), h(x)+1, h(x)+4, h(x)+9, \ldots \)
  - Quadratic probing leads to secondary clustering, more subtle, not as dramatic, but still systematic

- **Double hashing**
  - Compute a second hash function \( y=hh(x) \)
  - try: \( h(x), h(x)+y, h(x)+2y, h(x)+3y, \ldots \)
Suppose:

Table size: 37

$h(x) = 30$

$hh(x) = 7$

$h(y) = 30$

$hh(y) = 5$

<table>
<thead>
<tr>
<th>probe</th>
<th>Linear</th>
<th>Quadratic</th>
<th>Double x</th>
<th>Double y</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Second</td>
<td>31</td>
<td>31</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Third</td>
<td>32</td>
<td>34</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Fourth</td>
<td>33</td>
<td>1</td>
<td>14</td>
<td>7</td>
</tr>
</tbody>
</table>
Suppose:

Tablesize=11

h(t) = t%11

quadratic probing

put(2,A)
put(13,B)
put(24,C)
put(35,D)

get(35)

del(13)
get(24)
put(35,E)
get(24)
put(46,F)

tombstones!
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
- de-tombstoning
Using Hashtables

- No worries about hashing functions, rehashing, ...
- Someone else’s responsibility
- Example: who is visiting my site, and how often?
- for instance, hackers?
- web servers keep access logs
- java.util.HashMap
Parsing a line

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

```java
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 associated 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

```java
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());
        }
    }
}
```
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);
}
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