Hashing continued

cs206
Sep 25

Polynomial accumulation on Strings

Recommended by textbook

Handles really large numbers

```
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.value
Of((int)ss.charAt(i)));
            ll = ll.add(bb);
        }
        ll = ll.mod(BigInteger.valueOf(backingArray.length));
        return ll.intValue();
    }
```

33^15=599389454988654205434457

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

```
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

278

0	762
1	217
2	29
3	1
4	0
5	0
6	0
7	0
8	0
9	0

473

0	622
1	308
2	73
3	5
4	1
5	0
6	0
7	0
8	0
9	0

1550

0	210
1	342
2	252
3	136
4	55
5	9
6	4
7	1
8	0
9	0

2510

0	87
1	198
2	268
3	208
4	140
5	70
6	26
7	10
8	2
9	0

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 arround 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, …
    - Primary clustering – the bigger the cluster gets, the faster it grows
- Quadratic
  - try: h(x), h(x)+1, h(x)+4, h(x)+9, …
    - 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, …

Probing

Suppose:

Table size: 37

h(x) = 30

hh(x) = 7

h(y)=30

hh(y)=5

	Linear	Quadratic	Double x	Double y
First probe	30	30	30	30
Second	31	31	0	35
third	32	34	7	2
fourth	33	1	14	7

Probing Handling Deletions

Suppose:

Tablesize=11

h(t) = t%11

quadratic probing

put(2,A)

put(13,B)

put(24,C)

put(35,D)

get(35)

Locatio	Key	Value
0	35	D
1		
2	1	A
3	13	B
4		
5		
6	24	C
7		
8		
9		
10		

del(13)

get(24)

put(35,E)

get(24)

put(46,F)

Locatio	Key	Value
0	35	D
1		
2	1	A
3		
4		
5		
6	24	C
7		
8		
9		
10		

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

```
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/)"

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/)"

192.71.224.240 - - [31/Mar/2020:17:14:33 -0500] "GET /robots.txt HTTP/1.1" 404 - "-" "Go-http-
client/1.1"

192.71.38.71 - - [31/Mar/2020:17:14:33 -0500] "GET /humans.txt HTTP/1.1" 404 - "-" "Go-http-client/
1.1"

192.36.248.249 - - [31/Mar/2020:17:14:33 -0500] "GET /ads.txt HTTP/1.1" 404 - "-" "Go-http-client/
1.1"

192.71.2.171 - - [31/Mar/2020:17:14:34 -0500] "GET / HTTP/1.1" 200 2372 "-" "Go-http-client/1.1"

157.55.39.28 - - [31/Mar/2020:17:20:07 -0500] "GET /robots.txt HTTP/1.1" 404 - "-" "Mozilla/5.0
(compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)"

157.55.39.240 - - [31/Mar/2020:17:20:13 -0500] "GET /towers/ HTTP/1.1" 200 3436 "-" "Mozilla/5.0
(compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)"

103.15.226.14 - - [31/Mar/2020:17:56:43 -0500] "GET /admin/ HTTP/1.1" 404 - "-" "Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0"
```

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
```