Data and Information

Google Announces Plan To Destroy All Information It Can’t Index

MOUNTAIN VIEW, CA: Executives at Google, the rapidly growing online-search company that promised to “organize the world’s information,” announced Monday the latest step in their separation effort: a far-reaching plan to destroy all the information it is unable to index.

How and why do we organize data? Differences between data and information? What about knowledge?

Organizing Data: ideas and issues

- Often there is a time/space tradeoff
  - If we use more space (memory) we can solve a data/information problem in less time: time efficient
  - If we use more time, we can solve a data/information problem with less space: space efficient

- When do we repeat the same operation, e.g., search an array
  - Parameter changes, e.g., what we search for
  - We’re not “smart” enough to avoid the repetition
  - Learn new data structures or algorithms!
  - The problem is small enough or done infrequently enough that being efficient doesn’t matter
  - Markov illustrates this (next assignment)

From Google to Maps

- If we wanted to write a search engine we’d need to access lots of pages and keep lots of data
  - Given a word, on what pages does it appear?
  - This is a map of words->web pages

- In general a map associates a key with a value
  - Look up the key in the map, get the value
  - Google: key is word/words, value is list of web pages
  - DNS: key is hostname, value is IP address

- Interface issues
  - Lookup a key, return boolean: in map or value: associated with the key (what if key not in map?)
  - Insert a key/value pair into the map

Traceroute: where’s the map here?

traceroute www.cs.dartmouth.edu
ttraceroute to katahdin.cs.dartmouth.edu (129.170.213.101), 64 hops max,
1 lou (152.3.136.61) 2.566 ms
2 152.3.219.69 (152.3.219.69) 0.258 ms
3 telisp-roti.netcom.duke.edu (152.3.219.54) 0.336 ms
4 right67600-gw-to-duke67600-gw.ncrc.net (128.109.70.17) 184.752 ms
5 right67600-gw-to-right67600-gw.ncrc.net (128.109.70.37) 1.379 ms
6 right67600-gw-to-right67600-gw.ncrc.net (128.109.70.122) 1.647 ms
7 dep7600-gw2-to-dep7600-gw.ncrc.net (128.109.70.138) 2.273 ms
8 internet2-to-dep7600-gw2.ncrc.net (198.86.17.66) 10.494 ms
9 ge-0-1-0.10.ymlzq.abilene.unom.edu (64.57.28.10) 24.058 ms
10 so-0-0-0.0.rtr.newy.net.internet2.edu (64.57.28.10) 45.609 ms
11 nox300gw1-vl-110-nox-internet2.nox.org (192.5.89.221) 33.839 ms
12 " ..
13 ..
14 ..
15 border.ropeway-crt.dartmouth.edu (129.170.2.193) 50.991 ms
16 katahdin.cs.dartmouth.edu (129.170.213.101) 50.480 ms
Maps, another point of view

- An array is a map, consider array `arr`
  - The key is an index, say `i`, the value is `arr[i]`
  - Values stored sequentially/consecutively, not so good if the keys/indexes are 1, 100, and 1000, great if 0, 1, 2, 3, 4, 5

- Time/space trade-offs in map implementations, we’ll see more of this later
  - TreeMap: most operations take time $O(\log(N))$ for $N$-elements
  - HashMap: most operations are $O(1)$ time on average
    - Time for insert, get, … doesn’t depend on $N$ (wow!)
  - But! Elements in TreeMap are in order and TreeMap uses less memory than HashMap

Map (foreshadowing or preview)

- Any kind of Object can be inserted as a key in a HashMap
  - But, performance might be terrible if `hashValue` isn’t calculated well
  - Every object has a different number associated with it, we don’t want every object to be associated with 37, we want things spread out

- Only `Comparable` object can be key in TreeMap
  - Basically compare for less than, equal, or greater
  - Some objects are naturally comparable: `String`, `Integer`
  - Sometimes we want to change how objects are compared
  - Sometimes we want to invent `Comparable` things

Interface at work: `Frequencies.java`

- Similar to `WordCount` from before

- What clues are there for prototype of `map.get` and `map.put`?
  - What if a key is not in map, what value returned?
  - What kind of objects can be put in a map?
  - Kinds of maps?

```java
for(String s : words) {
    s = s.toLowerCase();
    Integer count = map.get(s);
    if (count == null) {
        map.put(s, 1);
    } else {
        map.put(s, count + 1);
    }
}
```

Interlude: What can an Object do (to itself)?

  - Look at `java.lang.Object`
  - What is this class? What is its purpose?

- `toString()`
  - Used to print `(System.out.println)` an object
  - overriding `toString()` useful in new classes
  - String concatenation: `String s = "value " + x;`
  - Default is basically a pointer-value
**What else can you do to an Object?**

- **equals(Object o)**
  - Determines if guts of two objects are the same, must override, e.g., for using `a.indexOf(o)` in `ArrayList` `a`
  - Default is `==`, pointer equality

- **hashCode()**
  - Hashes object (guts) to value for efficient lookup

- If you're implementing a new class, to play nice with others you must
  - Override `equals` and `hashCode`
  - Ensure that equal objects return same `hashCode` value

**Objects and values**

- **Primitive variables are boxes**
  - think memory location with value

- **Object variables are labels that are put on boxes**
  - String `s = new String("genome")`
  - String `t = new String("genome")`
  - if (`s == t`) {they label **the same box**}
  - if (`s.equals(t)`) {**contents of boxes the same**}

**Objects, values, classes**

- For primitive types: `int`, `char`, `double`, `boolean`
  - Variables have names and are themselves boxes (metaphorically)
  - Two int variables assigned 17 are equal with `==`

- For object types: `String`, `ArrayList`, others
  - Variables have names and are labels for boxes
  - If no box assigned, created, then label applied to `null`
  - Can assign label to existing box (via another label)
  - Can create new box using built-in `new`

- Object types are references or pointers or labels to storage

**Tomato and Tomato, how to code**

- **java.util.Collection and java.util.Collections**
  - one is an interface
    - `add()`, `addAll()`, `remove()`, `removeAll()`, `clear()`
    - `toArray()`, `size()`, `iterator()`
  - one is a collection of static methods
    - `sort()`, `shuffle()`, `reverse()`, `max()`
    - `frequency()`, `indexOfSubList()`

- **java.util.Arrays**
  - Also a collection of static methods
    - `sort()`, `fill()`, `binarySearch()`, `asList()`