Class Meeting #6

COS 226 — Spring 2018

Mark Braverman

(based on slides by

Jérémie Lumbroso)


AUTOCOMPLETE ME
AUTOCOMPLETE MENU
AUTOCOMPLETE ME COS
AUTOCOMPLETE ME PRINCETON

A note on collaboration

- Read collaboration policy.
- 10 total attempts per team.
- Keep team's submission in one account, and erase from the other.

Assignment Goal

Write a client that takes list of terms with weights and produces autocomplete
Manipulate **Comparator** / **Comparable**
Reimplement binary search

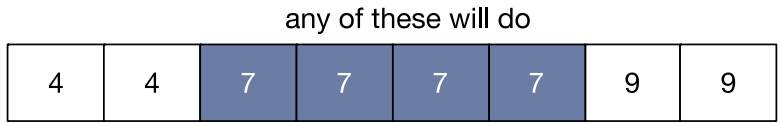
number of terms in file

```
   93827                          one term per line
         14608512  Shanghai, China
         13076300  Buenos Aires, Argentina
         12691836  Mumbai, India
         12294193  Mexico City, Distrito Federal, Mexico
         11624219  Karachi, Pakistan
weight per term 11174257  İstanbul, Turkey
```

Main challenge

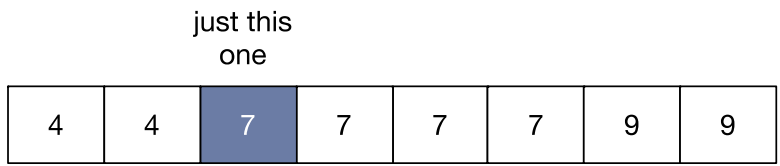
Problem 1: finding *any* index in a list

ANY



Problem 2: finding the *first* index in a list

FIRST



Test Client #1

(in assignment description)

It reads the data from the file; then it repeatedly reads autocomplete queries from standard input, and prints out the top k matching terms in descending order of weight.

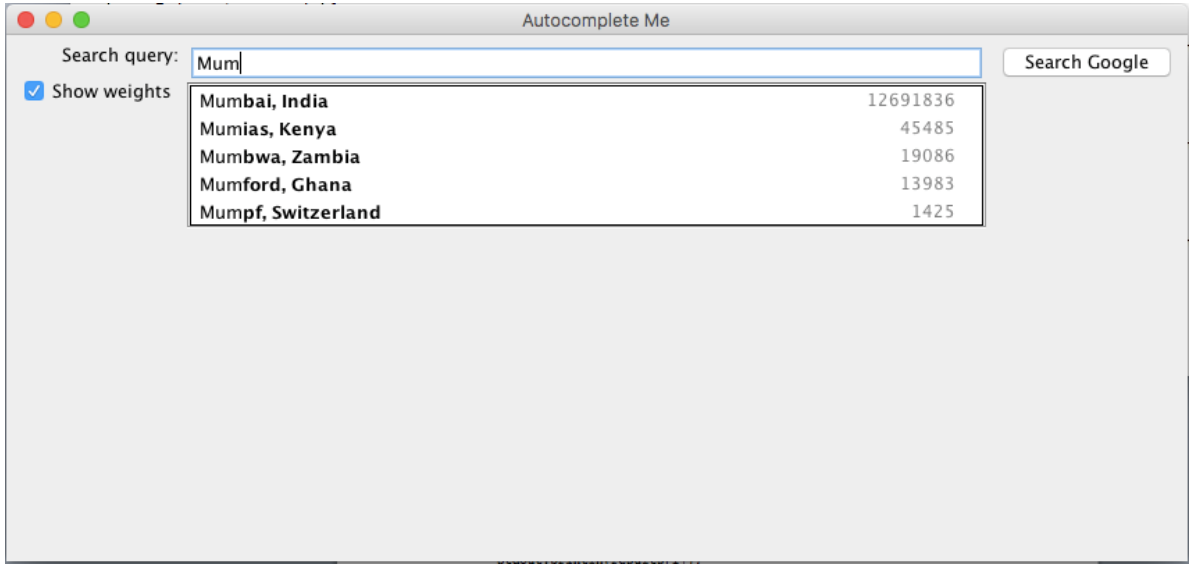
```
public static void main(String[] args) {

    // read in the terms from a file
    String filename = args[0];
    In in = new In(filename);
    int N = in.readInt();
    Term[] terms = new Term[N];
    for (int i = 0; i < N; i++) {
        long weight = in.readLong();           // read the next weight
        in.readChar();                         // scan past the tab
        String query = in.readLine();          // read the next query
        terms[i] = new Term(query, weight);    // construct the term
    }

    // read in queries from standard input and print out the top k matching terms
    int k = Integer.parseInt(args[1]);
    Autocomplete autocomplete = new Autocomplete(terms);
    while (StdIn.hasNextLine()) {
        String prefix = StdIn.readLine();
        Term[] results = autocomplete.allMatches(prefix);
        for (int i = 0; i < Math.min(k, results.length); i++)
            StdOut.println(results[i]);
    }
}
```

Test Client #2

AutocompleteGUI from FTP



COMPARATORS

Comparison function

compares two objects

instance version (where this == a):

    compareTo(Object b)

comparator version:

    compare(Object a, Object b)

returns 0 if objects are equal

returns -1 if a < b // or any negative int

returns 1 if a > b // or any positive int

```
int comparator(int a, int b) { return a-b; }
```

Two new interfaces

Comparable<T> applied to a class

- says that it can be compared to objects of type T
- add **compareTo(T other)**

Comparator<T> declares that an external class can be used to compare two objects of type T

- add **compare(T first, T second)**
- can be given to a function as a parameter

Comparator interface: using with our sorting libraries

To support comparators in our sort implementations:

- Use `Object` instead of `Comparable`.
- Pass `Comparator` to `sort()` and `less()` and use it in `less()`.

insertion sort using a Comparator

```
public static void sort(Object[] a, Comparator comparator)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0 && less(comparator, a[j], a[j-1]); j--)
         exch(a, j, j-1);
}

private static boolean less(Comparator c, Object v, Object w)
{  return c.compare(v, w) < 0;   }

private static void exch(Object[] a, int i, int j)
{  Object swap = a[i]; a[i] = a[j]; a[j] = swap;  }
```

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the `Comparator` interface.
- Implement the `compare()` method.

```
public class Student
{
   public static final Comparator<Student> BY_NAME    = new ByName();
   public static final Comparator<Student> BY_SECTION = new BySection();
   private final String name;
   private final int section;
   ...

   private static class ByName implements Comparator<Student>
   {
      public int compare(Student v, Student w)
      {  return v.name.compareTo(w.name);  }
   }

   private static class BySection implements Comparator<Student>
   {
      public int compare(Student v, Student w)
      {  return v.section - w.section;  }
   }
}
```

one Comparator for the class

this technique works here since no danger of overflow

What does <Key> mean?

```
// Returns the index of the first key in a[] that equals the search key, or -1 if no such key.
public static <Key> int firstIndexOf(Key[] a, Key key, Comparator<Key> comparator)

// Returns the index of the last key in a[] that equals the search key, or -1 if no such key.
public static <Key> int lastIndexOf(Key[] a, Key key, Comparator<Key> comparator)
```

Generic method: placeholder for argument type.
Enforces the condtition that *a* and *key* are of the same type:

Apple [] a;
Apple b;
….
firstIndexOf(a, b, Apples.bySweetness());

What does <Key> mean?

```
// Returns the index of the first key in a[] that equals the search key, or -1 if no such key.
public static <Key> int firstIndexOf(Key[] a, Key key, Comparator<Key> comparator)

// Returns the index of the last key in a[] that equals the search key, or -1 if no such key.
public static <Key> int lastIndexOf(Key[] a, Key key, Comparator<Key> comparator)
```

Generic method: placeholder for argument type.
Enforces the condtition that *a* and *key* are of the same type:

Apples [] a;
Orange b;
….
firstIndexOf(a, b, Apples.bySweetness());

TIPS

Don't reinvent the wheel

Look at Arrays java module

- Arrays.sort(…)
- Arrays.copyOfRange(T[] orig, int from, int to)

Use String.substring(int beginIndex)
String.substring(int begin, int end);

Two tips

Search for the first/last position efficiently

- no linear scan to find edge of array
- ideally, $1 + \lceil \log_2 n \rceil$ (how to test??)

Immutable (when resorting arrays)

- when you copy the select results
- resort them according to weight
- make sure the new array is a copy

This week's analysis question

```
/*****************************************************************************
 *  What is the order of growth of the number of compares (in the
 *  worst case) that each of the operations in the Autocomplete
 *  data type make, as a function of the number of terms N and the
 *  number of matching terms M?
 *
 *  Recall that with order-of-growth notation, you should discard
 *  leading coefficients and lower order terms, e.g., M^2 + M log N.
 *****************************************************************************/

constructor:

allMatches():

numberOfMatches():
```

Order of growth = no leading constants
Expecting N log N, or M log N, etc.
Pencil-and-paper analysis

TECHNICAL INTERVIEW QUESTIONS

Data streaming model

Sequence of elements (integers)

x[1] x[2] x[3] x[4] x[5]

Like an array

In some problems, we hope to make as few passes as possible

Ideally only 1 pass (read each element only once, sequentially from x[1] to x[N])

Use memory substantially smaller than N

Data streaming — Part 1

FINDING MISSING ELEMENTS

Problem #1: One Missing

Stream: x[1] x[2] x[3] x[4] … x[N-1]
with all elements from 1 to N appear once **except one (which is missing)**

Problem: find missing element

Requirements:

- one pass (scan elements from x[1] to x[N-1] and read each exactly once)
- linear time
- one word of memory

Problem #2: Two Missing

Stream: x[1] x[2] x[3] x[4] … x[N-2]
with all elements from 1 to N appear once **except two (which are missing)**

Problem: find **the two** missing element

Requirements:

- one pass (scan elements from x[1] to x[N-2] and read each exactly once)
- linear time
- (~) one word of memory