


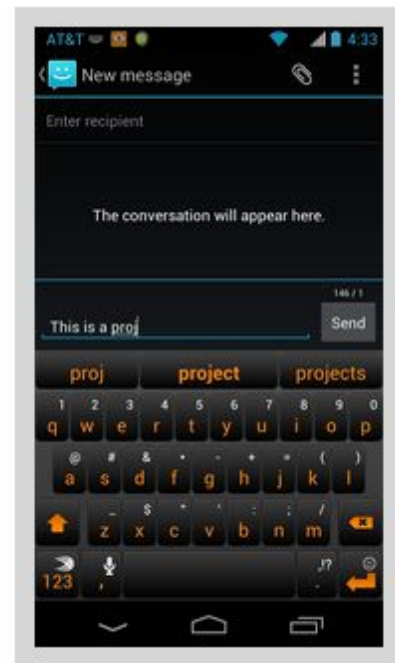
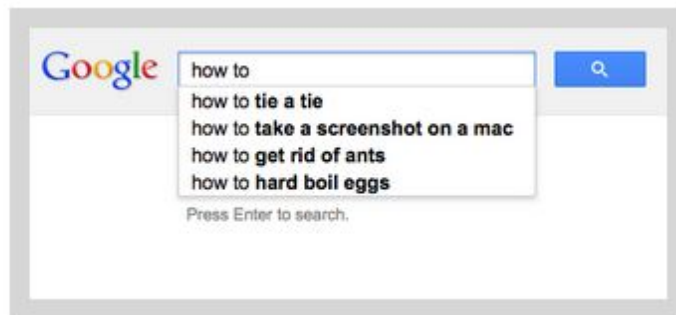


Discussion 6: Project 3

Kyle Hackett



AutoComplete Term
Binary Search Deluxe
Autocomplete





Problem 1. (Autocomplete Term)

Implement an immutable comparable data type called `Term` that represents an autocomplete

term: a string query and an associated real-valued weight. You must implement the following API, which supports comparing

terms by three different orders: lexicographic order by query; in descending order by weight; and lexicographic order by query

but using only the first r characters. The last order may seem a bit odd, but you will use it in Problem 3 to find all terms that start with a given prefix (of length r).



Term(String query) constructs a term given the associated query string, having weight 0

Term(String query, long weight) constructs a term given the associated query string and weight

String toString() returns a string representation of this term

int compareTo(Term that) returns a comparison of this term and other by query

static Comparator<Term> byReverseWeightOrder() returns a comparator for comparing two terms in reverse order of their weights

static Comparator<Term> byPrefixOrder(int r) returns a comparator for comparing two terms by their prefixes of length r



Instance Variables

String query

Long weight



Constructor(s)


2 cases - making a new Term by just passing a query, or passing a query and a weight

Throw appropriate errors depending on the constructor and assign the appropriate starting values also depending on the constructor



ToString - Print Weight then Query Separated by a Tab

CompareTo(Term other) - Return a negative, zero, or positive integer based on whether this.query is less than, equal to, or greater than other.query.



```
static Comparator<Term>  
byReverseWeightOrder()/byPrefixOrder(int r)
```

Return object of type ReverseWeightOrder/PrefixOrder
(Kind of like in the exercises)



ReverseWeightOrder

```
int compare(Term v, Term w)
```

* Return a negative, zero, or positive integer based on whether v.weight is less than, equal to, or greater than

w.weight.



Prefix Order

Instance variable:

- * Prefix length, int r.

- PrefixOrder(int r)

- * Initialize instance variable appropriately.

- int compare(Term v, Term w)

- * Return a negative, zero, or positive integer based on whether a is less than, equal to, or greater than b, where a is a substring of v of length $\min(r, v.\text{query.length}())$ and b is a substring of w of length $\min(r, w.\text{query.length}())$.



How the comparators are used

Used as a parameter in

```
Arrays.sort(terms , Term.byReverseWeightOrder ());
```



Problem 2. (Binary Search Deluxe)

When binary searching a sorted array that contains more than one key equal to the

search key, the client may want to know the index of either the first or the last such key. Accordingly, implement a library

called `BinarySearchDeluxe` with the following API:



```
static int firstIndexOf(Key[] a, Key key, Comparator<Key> c)
```

returns the index of the first key in a that equals the search key, or -1, according to the order induced by the comparator c

```
static int lastIndexOf(Key[] a, Key key, Comparator<Key> c)
```

returns the index of the last key in a that equals the search key, or -1, according to the order induced by the comparator c

Each method should should run in time $T(n) \sim \log n$, where n is the length of the array a

*Gonna especially pay attention to this during grading for the Code + Efficiency section



```
static int firstIndexOf(Key[] a, Key key, Comparator<Key> c)
```

- Modify the standard binary search such that when `a[mid]` matches `key`, instead of returning `mid`, remember it in, say

`index` (initialized to `-1`), and adjust `hi` appropriately.

- Return `index`

`static int lastIndexOf(Key[] a, Key key, Comparator<Key> c)` can be implemented similarly



Binary Search

- Start with full array, look at the middle
- Compare the middle element to the element you are searching for
- If your element is less than the middle
 - Move the high point to right before the middle
 - Find the middle of that new section
 - Rinse and repeat
- Same as above but opposite



Problem 3. (Autocomplete)

In this part, you will implement a data type that provides autocomplete functionality for a given set of string and weights, using `Term` and `BinarySearchDeluxe`. To do so, sort the terms in lexicographic order; use binary

search to find the set of terms that start with a given prefix; and sort the matching terms in descending order by weight.

Organize your program by creating an immutable data type called `Autocomplete` with the following API



`Autocomplete(Term[] terms)` constructs an autocomplete data structure from an array of terms

`Term[] allMatches(String prefix)` returns all terms that start with prefix, in descending order of their weights

`int numberOfMatches(String prefix)` returns the number of terms that start with prefix



The constructor should run in time $T(n) \sim n \log n$, where n is the number of terms.

- The `allMatches()` method should run in time $T(n) \sim \log n + m \log m$, where m is the number of matching terms.
- The `numberOfMatches()` method should run in time $T(n) \sim \log n$



Instance variables

Array of terms, `Term[]` terms



Autocomplete(Term[] terms)

Initialize this.terms to a defensive copy (ie, a fresh copy and not an alias) of. terms

- Sort this.terms in lexicographic order

(You made a defensive copy in the last project when creating the copy of the q so you could shuffle it)



Term[] allMatches(String prefix)

Find the index i of the first term in terms that starts with prefix.

- Find the number of terms (say n) in terms that start with prefix.
- Construct an array matches containing n elements from terms, starting at index i
- Sort matches in reverse order of weight and return the sorted array.

Hint* use one of the comparators you made in the last problem



int numberOfMatches(String prefix

- Find the indices i and j of the first and last term in terms that start with prefix.
- Using the indices, compute the number of terms that start with prefix, and return that value.