CSCI 421 Design and Analysis of Algorithms Spring 2019

Assignment 2

**Autocomplete Me**

Write a program to implement autocomplete for a given set of N terms, where a term is a query string and an associated nonnegative weight. That is, given a prefix, find all queries that start with the given prefix, in descending order of weight.

Please refer to the link as follow for details.

http://www.cs.princeton.edu/courses/archive/spring16/cos226/assignments/autocomplete.html

**Deliverables**. You need complete Part 1, 2, and 3 and submit **Autocomplete.java, BinarySearchDeluxe.java**, and **Term.java**. You may not use Arrays.sort() for this assignment, but rather should choose from among the sorts introduced in lecture. You may not call any library functions other than those in java.lang, java.util, and algs4.jar.

**Note**: If you want to use the textbook libraries ([stdlib.jar](http://algs4.cs.princeton.edu/code/stdlib.jar) and [algs4.jar](http://algs4.cs.princeton.edu/code/algs4.jar)) and make them accessible to Java, please follow the instruction shown in the first section of the link (section starts with “Install a Java programming environment”). https://introcs.cs.princeton.edu/java/assignments/percolation.html

////Term.java

import java.util.Comparator;

import java.lang.NullPointerException;

import java.lang.IllegalArgumentException;

public class Term implements Comparable<Term> {

private String query;

private double weight;

public Term(String query, double weight) {

if (query == null)

throw new NullPointerException("Your query cannot be null");

if (weight < 0)

throw new IllegalArgumentException("Weight cannot be negative");

this.query = query;

this.weight = weight;

}

public static Comparator<Term> byReverseWeightOrder() {

return (Comparator<Term>) (term1, term2) -> {

if (term1.weight > term2.weight)

return -1;

if (term1.weight == term2.weight)

return 0;

else

return 1;

};

}

public static Comparator<Term> byPrefixOrder(final int r) {

return new Comparator<Term>() {

@Override

public int compare(Term term1, Term term2) {

// the length of term1 and term2 must be less then r or be made length of r

if (term1.query.length() < r && term2.query.length() < r)

return term1.query.compareTo(term2.query);

else if (term1.query.length() < r)

return term1.query.compareTo(term2.query.substring(0, r));

else if (term2.query.length() < r)

return term1.query.substring(0, r).compareTo(term2.query);

else

return term1.query.substring(0, r).compareTo(term2.query.substring(0, r));

}

};

}

public int compareTo(Term that) {

String currQuery = this.query;

String prevQuery = that.query;

return currQuery.compareTo(prevQuery);

}

public String toString() {

return String.format("%.2f\t%s", this.weight, this.query);

}

}

///////BinarySearchDeluxe.java

import java.util.Comparator;

import java.lang.NullPointerException;

public class BinarySearchDeluxe {

public static <Key> int firstIndexOf(final Key[] a, final Key key, final Comparator<Key> comparator) {

if (a == null || key == null || comparator == null)

throw new NullPointerException("Cannot find first key index");

if (a.length == 0)

return -1;

int left = 0;

int right = a.length - 1;

while (left + 1 < right) {

int mid = left + (right - left) / 2;

if (comparator.compare(key, a[mid]) <= 0)

right = mid;

else

left = mid;

}

if (comparator.compare(key, a[left]) == 0)

return left;

if (comparator.compare(key, a[right]) == 0)

return right;

return -1;

}

public static <Key> int lastIndexOf(final Key[] a, final Key key, final Comparator<Key> comparator) {

if (a == null || key == null || comparator == null)

throw new NullPointerException("Cannot find Last key index");

if (a.length != 0) {// Binary Search Algorithm

int left = 0;

int right = a.length - 1;

while (left + 1 < right) {

int mid = left + (right - left) / 2;

if (comparator.compare(key, a[mid]) < 0)

right = mid;

else

left = mid;

}

if (comparator.compare(key, a[right]) == 0)

return right;

if (comparator.compare(key, a[left]) == 0)

return left;

return -1;

} else

return -1;

}

}

////////Autocomlete.java

import edu.princeton.cs.algs4.In;

import edu.princeton.cs.algs4.StdIn;

import edu.princeton.cs.algs4.StdOut;

import javax.swing.\*;

import java.lang.NullPointerException;

import java.util.Arrays;

public class Autocomplete {

private Term[] arrayTerms;

public Autocomplete(Term[] terms) {

this.arrayTerms = terms;

Arrays.sort(arrayTerms); // orders the array of terms

}

public Term[] allMatches(String prefix) {

if (prefix == null) {

throw new NullPointerException();

}

Term temp = new Term(prefix, 0);

int begin = BinarySearchDeluxe.firstIndexOf(arrayTerms, temp, Term.byPrefixOrder(prefix.length()));

int end = BinarySearchDeluxe.lastIndexOf(arrayTerms, temp, Term.byPrefixOrder(prefix.length()));

Term[] matches;

matches = Arrays.copyOfRange(arrayTerms, begin, end);

Arrays.sort(matches, Term.byReverseWeightOrder());

return matches;

}

public static void main(String[] args) {

String filename = "wiktionary.txt";

In in = new In(filename);

int N = in.readInt();

Term[] terms = new Term[N];

for (int i = 0; i < N; i++) {

double weight = in.readDouble(); // 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("10");

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]);

}

}

}

//////OUTPUT!!!!

A picture containing text

Description automatically generated