AutoComplete*

Vishak Srikanth
Final Project for OCS15: AP Computer Science A
Stanford Online High School
May 2021

*http://nifty.stanford.edu/2016/wayne-autocomplete-me/
https://introcs.cs.princeton.edu/java/assignments/autocomplete.html

AutoComplete: Project Background

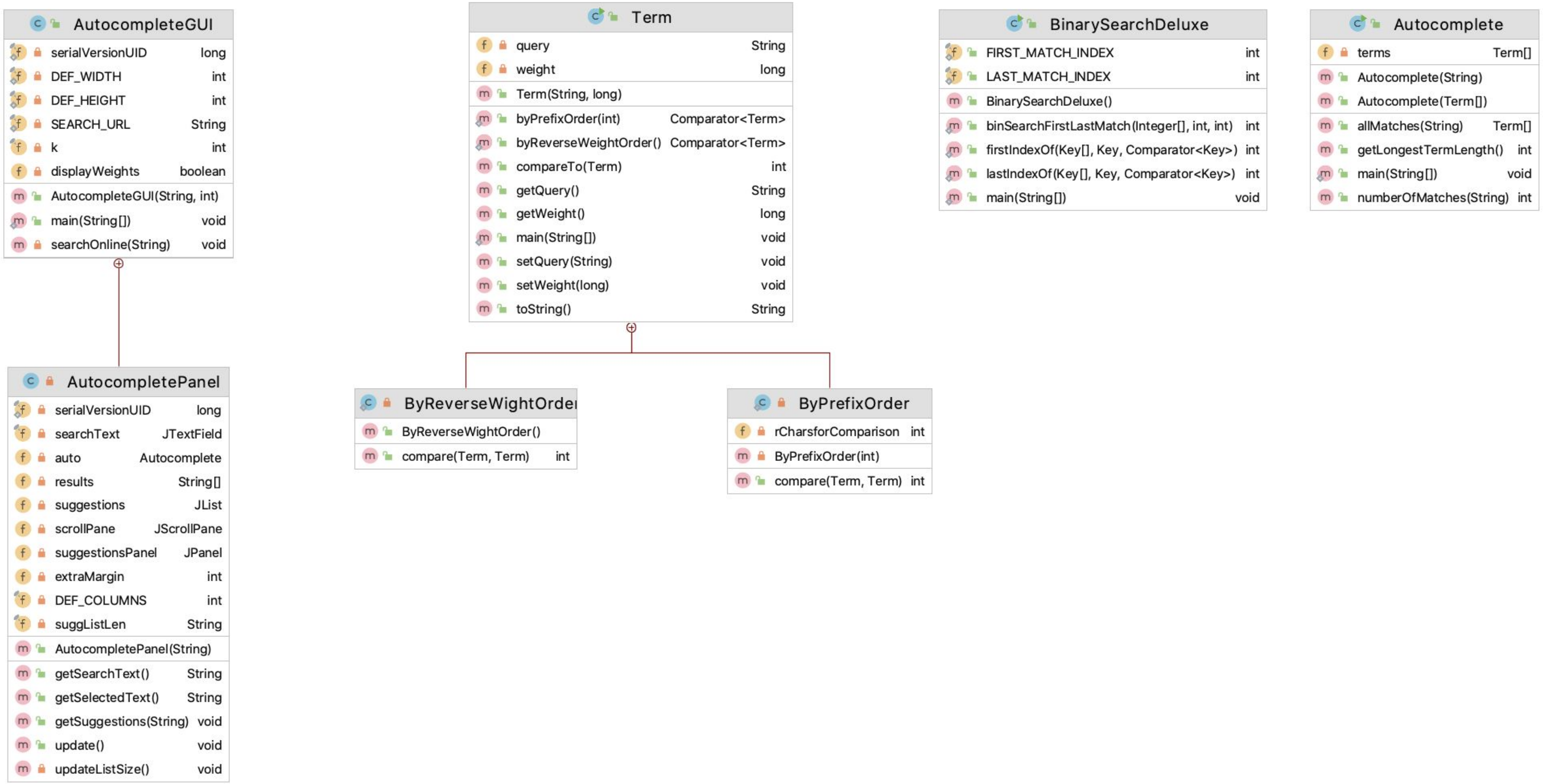
Goal

- Given a **prefix**, find all queries that **start** with the given prefix string, in descending order of weight (i.e priority of results)

  => Implement *Autocomplete* for a given set of *N terms*, where a term is a query string with associated non-negative weight

- Autocomplete is ubiquitous!
  - Google: Search search suggestions as user types search terms
  - Facebook: Find suggestions for Friends as user types
  - LinkedIn: People Search filters people as you type names
  - IMDB, Netflix: Show movies that match as the user types
  - Mobile phones: Intelligent text completions as user types

UML Diagram of Project Structure



C AutocompleteGUI	
f serialVersionUID	long
f DEF_WIDTH	int
f DEF_HEIGHT	int
f SEARCH_URL	String
f k	int
f displayWeights	boolean
m AutocompleteGUI(String, int)	
m main(String[])	void
m searchOnline(String)	void

C AutocompletePanel	
f serialVersionUID	long
f searchText	JTextField
f auto	Autocomplete
f results	String[]
f suggestions	JList
f scrollPane	JScrollPane
f suggestionsPanel	JPanel
f extraMargin	int
f DEF_COLUMNS	int
f suggListLen	String
m AutocompletePanel(String)	
m getSearchText()	String
m getSelectedText()	String
m getSuggestions(String)	void
m update()	void
m updateListSize()	void

C Term	
f query	String
f weight	long
m Term(String, long)	
m byPrefixOrder(int)	Comparator<Term>
m byReverseWeightOrder()	Comparator<Term>
m compareTo(Term)	int
m getQuery()	String
m getWeight()	long
m main(String[])	void
m setQuery(String)	void
m setWeight(long)	void
m toString()	String

C ByReverseWightOrde	
m ByReverseWightOrder()	
m compare(Term, Term)	int

C ByPrefixOrder	
f rCharsforComparison	int
m ByPrefixOrder(int)	
m compare(Term, Term)	int

C BinarySearchDeluxe	
f FIRST_MATCH_INDEX	int
f LAST_MATCH_INDEX	int
m BinarySearchDeluxe()	
m binSearchFirstLastMatch(Integer[], int, int)	int
m firstIndexOf(Key[], Key, Comparator<Key>)	int
m lastIndexOf(Key[], Key, Comparator<Key>)	int
m main(String[])	void

C Autocomplete	
f terms	Term[]
m Autocomplete(String)	
m Autocomplete(Term[])	
m allMatches(String)	Term[]
m getLongestTermLength()	int
m main(String[])	void
m numberOfMatches(String)	int

Key Classes & Their Use

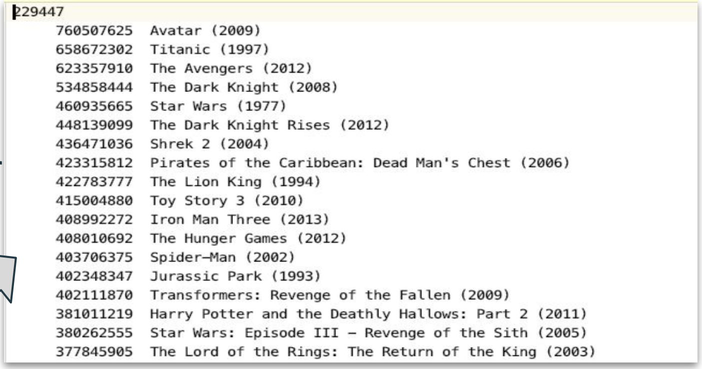
- Term: Data type that has a query string and an associated integer weight
  - Supports comparison of terms by three different orders:
    - lexicographic order by query string (natural order: alphabetic with ties (identical strings)ordered in their ordinal position in original array)
    - descending order by weight (i.e. priority of search results such as highest to lowest votes or box office collections for movies, or highest to lowest city population, highest to lowest word frequency in a corpus such as wikipedia)
    - lexicographic order by query string but using only the first r characters (used to query prefix string from a database)
- BinarySearchDeluxe: Enhanced Binary Search algorithm that has methods to return position of **first** and **last match** of a key in an array of sorted objects to be searched
  - Makes $1 + \lceil \log_2 N \rceil$ comparisons (worst case) where N is the length of the array. [compare is a single call to Comparator<Key>.compare()]

Key Classes & Their Use

- Autocomplete: Main data type that provides autocomplete functionality
  - Has a field that holds an array of Term objects
  - Uses static methods from BinarySearchDeluxe class
  - Flow:
    - Sort terms in lexicographic order
    - Use binary search to find the all query strings that start with a given prefix (for e.g as user types each character the prefix to search will change)
    - Sort the matching terms in descending order by weight
  - Methods:
    - Returns all terms that start with the given prefix, in descending order of weight [uses log N + M log M comparisons (worst case) if N is the number of terms and M is the number of matching terms].
    - Returns the number of terms that start with the given prefix [makes log N comparisons (worst case)
- AutocompleteGUI: Swing GUI reference implementation provided by Princeton CS department as part of CS226 course
  - Used as is except for one small change to remove the dependency on a custom input file reader jar and replaced with other generic classes and methods that are part of any standard Java 8 install

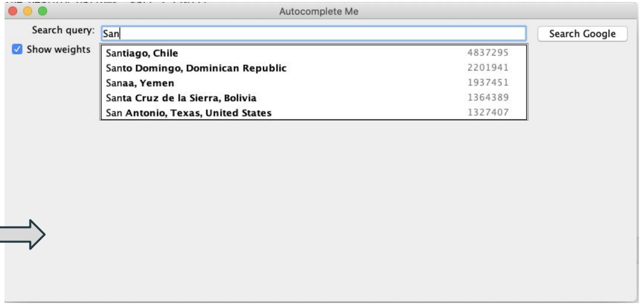
Testing

- All databases used are in the subfolder called ‘data’ in the working directory of IntelliJ project
- Variety of database files used for testing: movies (250K entries), cities (93K entries), google n-gram words (1m+ for 5-grams for example) etc.
- Each Database text file has following structure:
  - Header line with # of entries in DB
  - Each line has a weight, a tab, and then the query term for the string as shown



```
229447
      760507625   Avatar (2009)
      658672302   Titanic (1997)
      623357910   The Avengers (2012)
      534858444   The Dark Knight (2008)
      460935665   Star Wars (1977)
      448139099   The Dark Knight Rises (2012)
      436471036   Shrek 2 (2004)
      423315812   Pirates of the Caribbean: Dead Man's Chest (2006)
      422783777   The Lion King (1994)
      415004880   Toy Story 3 (2010)
      408992272   Iron Man Three (2013)
      408010692   The Hunger Games (2012)
      403706375   Spider-Man (2002)
      402348347   Jurassic Park (1993)
      402111870   Transformers: Revenge of the Fallen (2009)
      381011219   Harry Potter and the Deathly Hallows: Part 2 (2011)
      380262555   Star Wars: Episode III - Revenge of the Sith (2005)
      377845905   The Lord of the Rings: The Return of the King (2003)
```

- A term Array is created by parsing the input file using standard Java buffered reader and writer interfaces
- The Autocomplete data structure can be created once the terms array is created
- The GUI app is run against a specific database with a max hit count (i.e. limits matching results to a number specified)



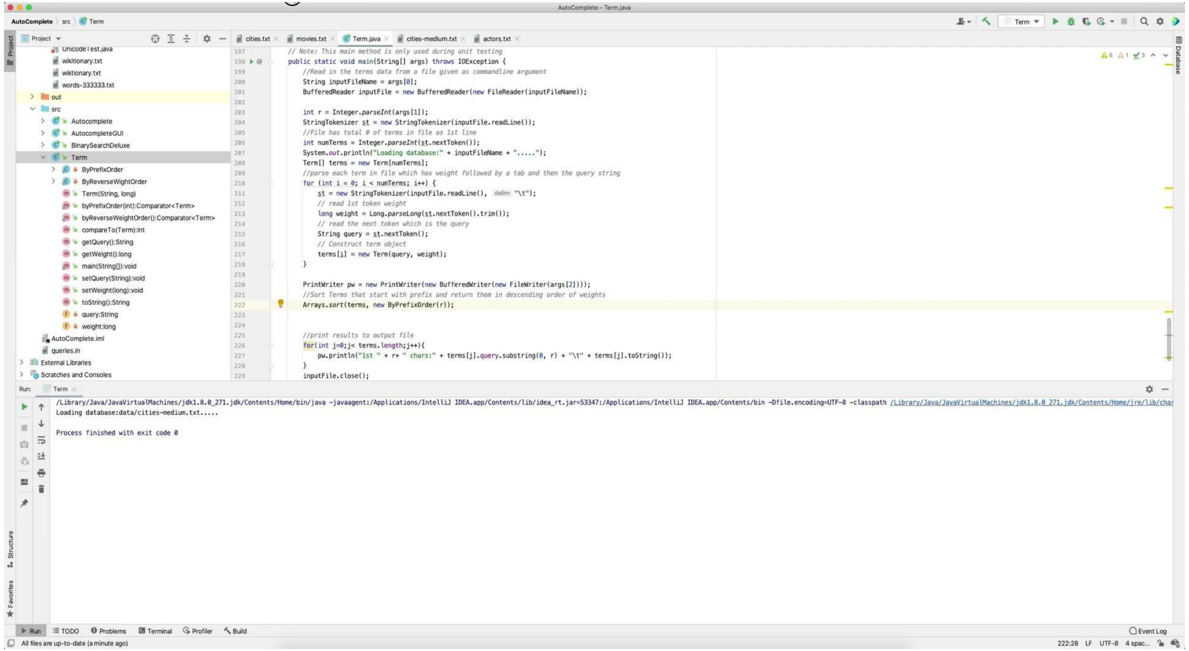
Search query: San

☑ Show weights

```
Santiago, Chile                           4837295
Santo Domingo, Dominican Republic         2201941
Sanaa, Yemen                              1937451
Santa Cruz de la Sierra, Bolivia          1364389
San Antonio, Texas, United States         1327407
```

Unit Testing Term Class

- A term Array is created by parsing cities-medium.txt
- The terms are sorted based on a prefix sort of 1st 3 characters
- The main method requires 3 command line arguments:
  - Database text input file (cities-medium.txt in demo)
  - Prefix length to sort terms entries by (3 in demo)
  - Output file (term-unittest-prefix-sorted-cities-medium.out in demo)
- Output file shows cities in sorted order based on 1st 3 characters

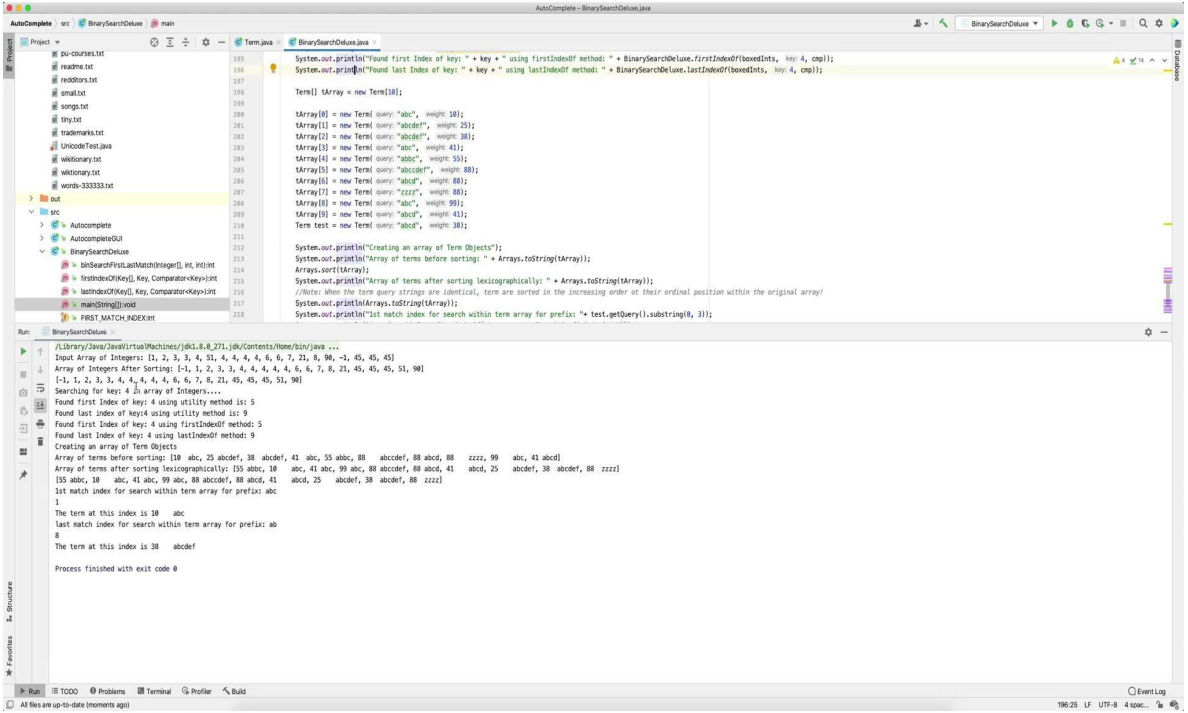


```java
// Note: This main method is only used during unit testing
public static void main(String[] args) throws IOException {
    //Read in the terms data from a file given as commandline argument
    String inputFileName = args[0];
    BufferedReader inputFile = new BufferedReader(new FileReader(inputFileName));

    int r = Integer.parseInt(args[1]);
    StringTokenizer st = new StringTokenizer(inputFile.readLine());
    //File has total # of terms in file as 1st line
    int numTerms = Integer.parseInt(st.nextToken());
    System.out.println("Loading database:" + inputFileName + ".....");
    Term[] terms = new Term[numTerms];
    //parse each term in file which has weight followed by a tab and then the query string
    for (int i = 0; i < numTerms; i++) {
        st = new StringTokenizer(inputFile.readLine(),  delim: "\t");
        // read 1st token weight
        long weight = Long.parseLong(st.nextToken().trim());
        // read the next token which is the query
        String query = st.nextToken();
        // Construct term object
        terms[i] = new Term(query, weight);
    }

    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(args[2])));
    //Sort Terms that start with prefix and return them in descending order of weights
    Arrays.sort(terms, new ByPrefixOrder(r));


    //print results to output file
    for(int j=0;j< terms.length;j++){
        pw.println("1st " + r+ " chars:" + terms[j].query.substring(0, r) + "\t" + terms[j].toString());
    }
    inputFile.close();
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53347:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/jre/lib/char
Loading database:data/cities-medium.txt.....

Process finished with exit code 0
```

Unit Testing BinarySearchDeluxe Class

- Main method creates an array of Integers having duplicates and an array of Terms having duplicate strings
- Boxed Integer array is sorted in natural order (ascending int value)
- Terms are sorted based on default comparator which sorts terms in lexicographic order (alphabetic)
- All output is printed locally to show the sorting and the binary search results to find the 1st and last match of the search key in both cases



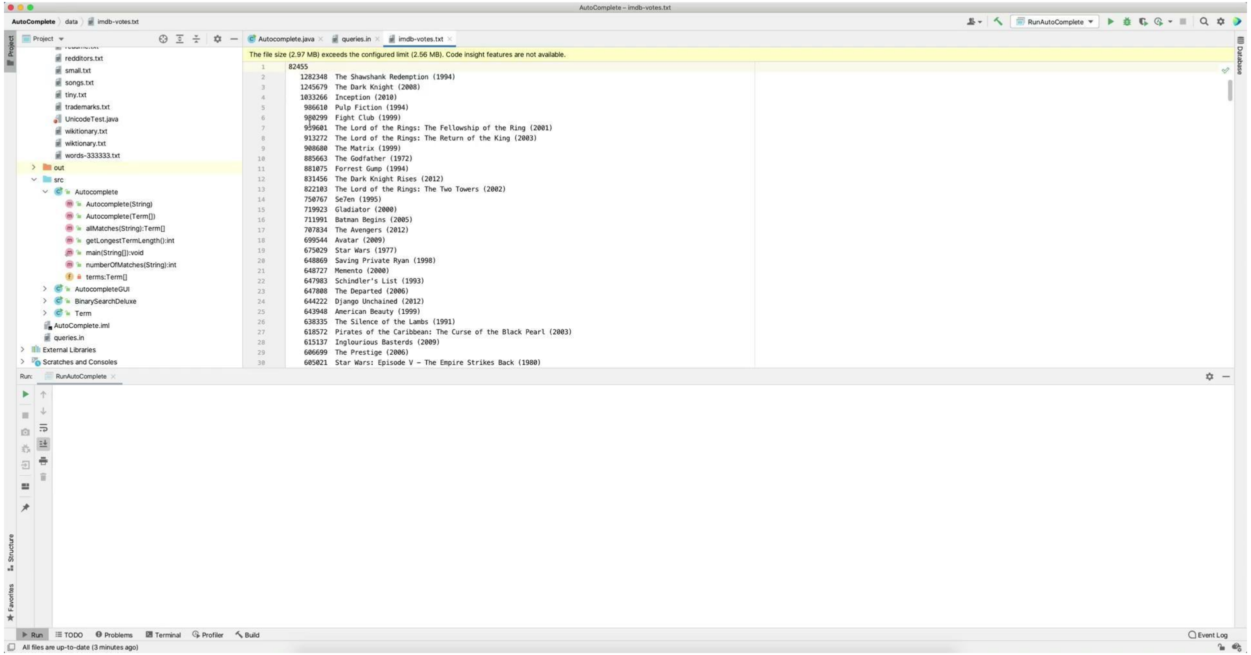
```java
System.out.println("Found first Index of key: " + key + " using firstIndexOf method: " + BinarySearchDeluxe.firstIndexOf(boxedInts, key: 4, cmp));
System.out.println("Found last Index of key: " + key + " using lastIndexOf method: " + BinarySearchDeluxe.lastIndexOf(boxedInts, key: 4, cmp));

Term[] tArray = new Term[10];

tArray[0] = new Term( query: "abc",  weight: 10);
tArray[1] = new Term( query: "abcdef",  weight: 25);
tArray[2] = new Term( query: "abcdef",  weight: 38);
tArray[3] = new Term( query: "abc",  weight: 41);
tArray[4] = new Term( query: "abbc",  weight: 55);
tArray[5] = new Term( query: "abccdef",  weight: 88);
tArray[6] = new Term( query: "abcd",  weight: 88);
tArray[7] = new Term( query: "zzzz",  weight: 88);
tArray[8] = new Term( query: "abc",  weight: 99);
tArray[9] = new Term( query: "abcd",  weight: 41);
Term test = new Term( query: "abcd",  weight: 38);

System.out.println("Creating an array of Term Objects");
System.out.println("Array of terms before sorting: " + Arrays.toString(tArray));
Arrays.sort(tArray);
System.out.println("Array of terms after sorting lexicographically: " + Arrays.toString(tArray));
//Note: When the term query strings are identical, term are sorted in the increasing order ot their ordinal position within the original array!
System.out.println(Arrays.toString(tArray));
System.out.println("1st match index for search within term array for prefix: "+ test.getQuery().substring(0, 3));
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java ...
Input Array of Integers: [1, 2, 3, 3, 4, 51, 4, 4, 4, 4, 6, 6, 7, 21, 8, 90, -1, 45, 45, 45]
Array of Integers After Sorting: [-1, 1, 2, 3, 3, 4, 4, 4, 4, 4, 6, 6, 7, 8, 21, 45, 45, 45, 51, 90]
[-1, 1, 2, 3, 3, 4, 4, 4, 4, 4, 6, 6, 7, 8, 21, 45, 45, 45, 51, 90]
Searching for key: 4 in array of Integers....
Found first Index of key: 4 using utility method is: 5
Found last index of key:4 using utility method is: 9
Found first Index of key: 4 using firstIndexOf method: 5
Found last Index of key: 4 using lastIndexOf method: 9
Creating an array of Term Objects
Array of terms before sorting: [10  abc, 25 abcdef, 38  abcdef, 41  abc, 55 abbc, 88    abccdef, 88 abcd, 88    zzzz, 99    abc, 41 abcd]
Array of terms after sorting lexicographically: [55 abbc, 10    abc, 41 abc, 99 abc, 88 abccdef, 88 abcd, 41    abcd, 25    abcdef, 38  abcdef, 88  zzzz]
[55 abbc, 10    abc, 41 abc, 99 abc, 88 abccdef, 88 abcd, 41    abcd, 25    abcdef, 38  abcdef, 88  zzzz]
1st match index for search within term array for prefix: abc
1
The term at this index is 10    abc
last match index for search within term array for prefix: ab
8
The term at this index is 38    abcdef

Process finished with exit code 0
```

Unit Testing AutoComplete Class

Main method takes 3 command line parameters:

- Reads a database input file (imdb-votes.txt) and creates an Autocomplete object loading the content into an array of Term objects
- Reads a query file (query.in) that contains # of results to be returned followed by each query to be against the loaded database returning the specified number of results
- Writes output to the specified file autocomplete-unit-test-results.out



The file size (2.97 MB) exceeds the configured limit (2.56 MB). Code insight features are not available.

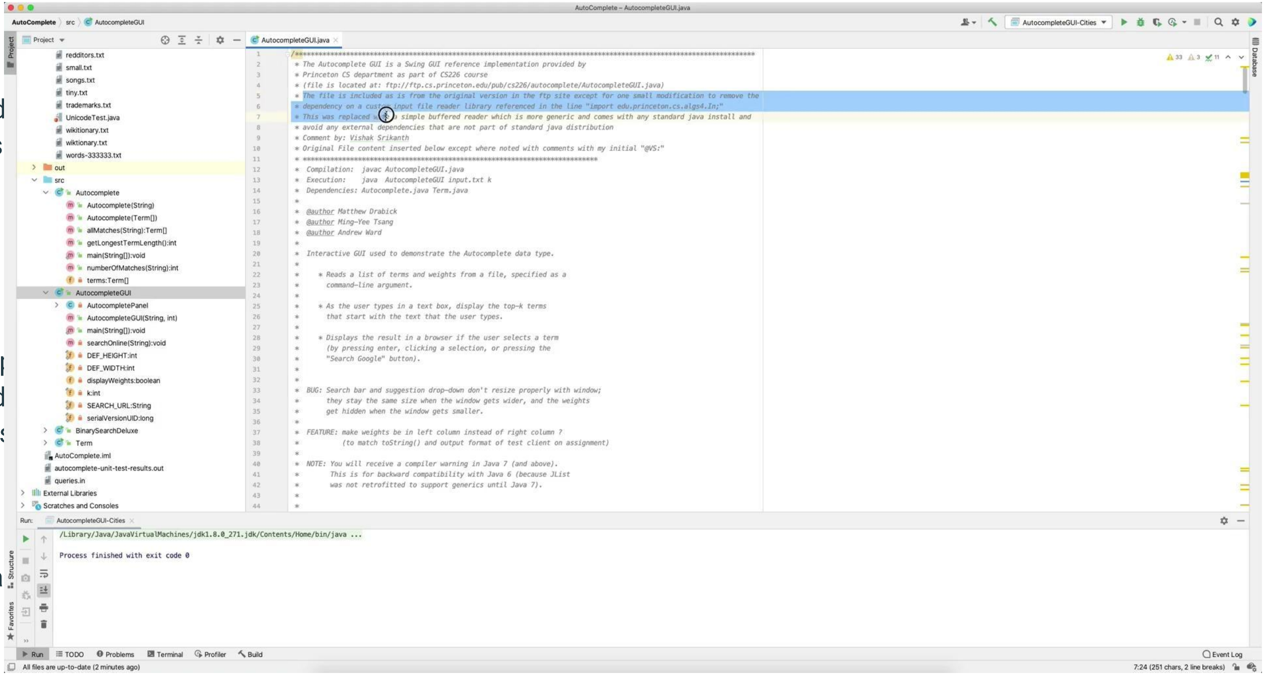
```
82455
    1282348  The Shawshank Redemption (1994)
    1245679  The Dark Knight (2008)
    1033266  Inception (2010)
     986610  Pulp Fiction (1994)
     980299  Fight Club (1999)
     939601  The Lord of the Rings: The Fellowship of the Ring (2001)
     913272  The Lord of the Rings: The Return of the King (2003)
     908680  The Matrix (1999)
     885663  The Godfather (1972)
     881075  Forrest Gump (1994)
     831456  The Dark Knight Rises (2012)
     822103  The Lord of the Rings: The Two Towers (2002)
     750767  Se7en (1995)
     719923  Gladiator (2000)
     711991  Batman Begins (2005)
     707834  The Avengers (2012)
     699544  Avatar (2009)
     675029  Star Wars (1977)
     648869  Saving Private Ryan (1998)
     648727  Memento (2000)
     647983  Schindler's List (1993)
     647808  The Departed (2006)
     644222  Django Unchained (2012)
     643948  American Beauty (1999)
     638335  The Silence of the Lambs (1991)
     618572  Pirates of the Caribbean: The Curse of the Black Pearl (2003)
     615137  Inglourious Basterds (2009)
     606699  The Prestige (2006)
     605021  Star Wars: Episode V – The Empire Strikes Back (1980)
```

All files are up-to-date (3 minutes ago)

Testing AutoCompleteGUI Class

Demo shows 2 different scenario:

1. Autocomplete that returns top 5 matches (search results are weighted in descending order of population) as user searches the cities database. When user double clicks on their selection it puts them on a Google search results page with selected term with search already performed.
2. Autocomplete that returns upto 10 top matches (search results are weighted in descending order of IMDB votes) as user searches the IMDB ratings database and then puts user on a Google search results page with selected term once the user selects a movie and presses Search on Google button in UI.



```java
/***************************************************************************************************
 * The Autocomplete GUI is a Swing GUI reference implementation provided by
 * Princeton CS department as part of CS226 course
 * (file is located at: ftp://ftp.cs.princeton.edu/pub/cs226/autocomplete/AutocompleteGUI.java)
 * The file is included as is from the original version in the ftp site except for one small modification to remove the
 * dependency on a custom input file reader library referenced in the line "import edu.princeton.cs.algs4.In;"
 * This was replaced with a simple buffered reader which is more generic and comes with any standard java install and
 * avoid any external dependencies that are not part of standard java distribution
 * Comment by: Vishak Srikanth
 * Original File content inserted below except where noted with comments with my initial "@VS:"
 * ************************************************************************
 *  Compilation:  javac AutocompleteGUI.java
 *  Execution:    java  AutocompleteGUI input.txt k
 *  Dependencies: Autocomplete.java Term.java
 *
 *  @author Matthew Drabick
 *  @author Ming-Yee Tsang
 *  @author Andrew Ward
 *
 *  Interactive GUI used to demonstrate the Autocomplete data type.
 *
 *     * Reads a list of terms and weights from a file, specified as a
 *       command-line argument.
 *
 *     * As the user types in a text box, display the top-k terms
 *       that start with the text that the user types.
 *
 *     * Displays the result in a browser if the user selects a term
 *       (by pressing enter, clicking a selection, or pressing the
 *       "Search Google" button).
 *
 *
 *  BUG: Search bar and suggestion drop-down don't resize properly with window;
 *       they stay the same size when the window gets wider, and the weights
 *       get hidden when the window gets smaller.
 *
 *  FEATURE: make weights be in left column instead of right column ?
 *           (to match toString() and output format of test client on assignment)
 *
 *  NOTE: You will receive a compiler warning in Java 7 (and above).
 *        This is for backward compatibility with Java 6 (because JList
 *        was not retrofitted to support generics until Java 7).
 *
 *
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_271.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```