CS 171 (Summer 2022)
Assignment 5: Binary Search Trees
Due: 8/3/2022 (11:59pm)

Goals:
Implement a Binary Search Tree (BST) based index for indexing/searching actors or movies in the IMDB dataset.

Details:
Our goal is to quickly find the information associated with an actor (or a movie), based on the movie or actor name (e.g. Arnold Schwartzeneger), or a prefix of a name (Arnold Schw*). The search should be case insensitive (e.g. arnold schw*) should also work. In other words, the search key is the short or simplified name of a movie or actor. The associated value of data is of type MovieInfo, which is defined in the following class:

```
class MovieInfo {
  public String shortName; // short or simplified name, e.g., "Tom Hanks"
  public String fullName;  // full name, e.g., "Hanks, Thomas III".
  public int ID;           // integer ID
}
```

The BST index should be implemented as a stand-alone class `BSTIndex`. Your `BSTIndex` class should have at least the following public methods (you can add as many additional private helper methods as needed). You will need to define a Node class within the BSTIndex class that contains four fields: key (of String type), data (of MovieInfo type), left and right link to the children nodes.

- `public BSTIndex()`: constructor to initialize the BST. The data element should be an object of the type `MovieInfo`, described above.
- `public MovieInfo findExact(String key)`: return the data element `MovieInfo` where the shortName matches the key exactly (can have different capitalization).
- `public MovieInfo findPrefix(String prefix)`: return the data element `MovieInfo` where the shortName starts with the prefix (can have different capitalization).
- `public void insert (MovieInfo data)`: insert the given data element into the proper place in the BST structure.

The `IndexTester` class is provided which will test your BSTIndex class. The `IndexTester` creates an empty `BSTIndex` object (using the constructor of the `BSTIndex` class), reads the data from an input movie or actor file, builds a `MovieInfo` object for each row, and inserts it into the BST (using the insert method of the `BSTIndex` class). At this point, the BST index is built for all of the movie or actor entries in the file. It will then ask for a search string from the user and search fo the `MovieInfo` object associated with the name, testing the search functionality of your `BSTIndex` (using the `findExact` or `findPrefix` methods of the `BSTIndex`).

Getting Started

1. Get the starter code `IndexTester.java` and `MovieInfo.java` from `~cs171001/share/hw5` directory and understand it.
2. The input data is also located in the `~cs171001/share/hw5` directory. Note that the full data files (`actors.txt` and `movies.txt` are relatively large). You can copy the smaller extracts (`actors100K.txt` and `movies100K.txt`) for debugging. They are all in the same format: one actor or movie per line, with fields ID, shortName, fullName separated by a tab character “`\t`”. Read `IndexTester` code to remind yourself how to read this data.

3. Implement `BSTIndex` class.
4. Test your program with IndexTester on the different datasets, starting with the smaller ones.

Honor Code
The assignment is governed by the College Honor Code and Departmental Policy. Please remember to have the following comment included at the top of the file.

```
/*
THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
CODE WRITTEN BY OTHER STUDENTS. _Your_Name_Here_
*/
```

Submission:
Place your completed BSTIndex.java file directly under your `~/cs171/hw5` directory. Then use the turnin command:

```
~cs171001/turnin BSTIndex.java hw5
```

Grading:

- If your code does not compile, it will get 0 points.
- Index building/insert (55 pts.)
- Exact search (20 pts.)
- Prefix search (20 pts.)
- Code clarity and style (5 pts.)