COMS 3134 Homework 4

Submission instructions

All programs must compile and run on CUNIX to receive credit. Submit your electronic files via `http://courseworks.columbia.edu`. We prefer electronic submission of theory, though you will not be penalized for paper submissions. (Do **not** print out your programs.) I recommend learning LaTeX for typesetting math. Include a README file that explains exactly what each file in your submission is. Place all the files you want to submit into a submission directory with the following naming scheme.

`<your_uni>_hw<number>`

So if my UNI is `uni1234` am submitting homework 6, my directory would be `uni1234_hw6`. Archive your submission directory using

```
tar -czvf uni1234_hw6.tar.gz uni1234_hw6
```

and upload `uni1234_hw6.tar.gz` to courseworks. (You will probably need to first download the file to a local directory using an FTP program. See CUNIX tutorial for more info.)

Multiple Submissions: You can submit multiple times, but we will only consider the latest submission based on the timestamp in courseworks. Please give at least 1-2 minutes between two submissions so we can tell which is the newest submission.

I recommend that you also keep a pristine copy of your submission folder in case there is any submission error.

Theory Problems

Make sure your solutions are clear. Diagrams and math are often insufficient to convey exactly what you mean, so supplement with some text. Either pseudocode or Java are acceptable when asked to provide algorithms. Nevertheless, clear, concise English is often preferable.

1. (4 points) A d-heap is a priority queue data structure that generalizes the binary heap. In the tree containing the data, nodes have d children instead of 2 two children. What are the (big-Oh) running times of the `insert` and `deleteMin` operations? Prove your answer.

2. (8 points) **Weiss 5.1** Given input $\{4371, 1323, 6173, 4199, 4344, 9679, 1989\}$ and a hash function $h(x) = (x \bmod 10)$, show the resulting:

   (a) separate chaining hash table.

   (b) hash table using linear probing.

(c) hash table using quadratic probing.

(d) hash table with second hash function $h_2(x) = 7 - (x \bmod 7)$.

3. (8 points) (Based on **Weiss 5.2**) Show the result of rehashing the hash tables in the previous exercise. Rehash using a new table size of 19, and a new hash function $h(x) = (x \bmod 19)$.

4. (4 points) **Weiss 5.6** In the quadratic probing hash table, suppose that instead of inserting a new item into the location suggested by `findPos`, we insert into the first inactive cell on the search path (thus, it is possible to reclaim a cell that is marked "deleted," potentially saving space).

   (a) Rewrite the insertion algorithm to use this observation. Do this by having `findPos` maintain, with an additional variable, the location of the first inactive cell it encounters.

   (b) Explain the circumstances under which the revised algorithm is faster than the original algorithm. Can it be slower?

5. (6 points) Describe the advantages and disadvantages of Binary Search Trees versus Binary Heaps versus Hash Tables. What operations are efficient in some but not others? What are the tradeoffs between the three data structures? Give three example applications, one for each of these data structures, where it makes the most sense to use that structure and not the other two. Explain your choices.

Programming Problem

1. (30 points) **Word Frequency Counter**

   Write a program `WordCounter.java` that reads a text file and counts the number of times each word appears. Your program should then output words that are used more often than some threshold value given by the user. To avoid boring results, compare against the provided list `commonwords.txt` of the 100 most commonly used words in the English language[1].

   Clarification: output only words that are not the 100 most commonly used words. Otherwise no matter what text you run the code on you will see pretty much the same output. This way we can isolate words that are more specific to the text.

   - Your program should use the built in Java `HashMap` and `HashSet` classes. I leave it up to you to decide how to use them reasonably. Make sure to read the API documentation to understand the differences between the two hash table classes. The only $O(N)$ operation in your program should be iterating through the text file. E.g., your entire program should run in $O(N)$ time to count the words, and $O(k)$ time to output the k words.
   - You should consolidate words by removing capitalization with `toLowerCase()` and split words by spaces and punctuation with the `String` method:

     ```
     split([^a-zA-Z'])
     ```

[1] according to `http://www.duboislc.org/EducationWatch/First100Words.html`

We want to remove any punctuation that splits words, but keep the apostrophe since it often signifies contractions. You may optionally read more about regular expressions at `http://java.sun.com/docs/books/tutorial/essential/regex/` if you want to do something more advanced.

- Try your program on the included text files which were downloaded from Project Gutenberg's public domain e-book archive `http://www.gutenberg.org/`.

```
totc.txt          // "Tale of Two Cities" by Charles Dickens
dyssy10.txt // "The Odyssey" by Homer
tomsawyer.txt // "The Adventures of Tom Sawyer" by Mark Twain
```

- An efficient implementation should take at most 10 seconds or so to complete the word count on a large book like Charles Dickens', Tale of Two Cities.

Here is an example output that you can compare your program to:

```
java WordCounter totc.txt 200
defarge: 282
night: 219
am: 225
me: 529
mr: 622
know: 232
little: 267
lorry: 339
again: 228
before: 233
himself: 222
upon: 291
good: 217
any: 261
hand: 251
very: 218
miss: 233
man: 284
```