

CSE250, Fall 2011 Assignment 10 Due Wed. 12/7 & Fri. 12/9

Including final project task and report questions

The **Final Exam** will be held on **Monday, Dec. 12, 11:45am–2:45pm** in **NSC 220(!)**. None of this is ideal—while the exam will stay open-book, please plan with the awareness that there won't be room for much shuffling of papers and books, and laptops are forbidden. I am at the moment unsure whether I can arrange a review session just before; in any event this graded homework will be returned by exam time.

Reading:

For next week, read Chapter 10 on insertion sort, merge sort, quicksort, and heap sort, and read Chapter 11 just on red-black trees.

The first three problems are due in hardcopy in class on Wed. 12/7

(1) Write on paper the `operator++()` method for a `WordTree::iterator` that would do a *postorder* transversal, as opposed to the preorder transversal executed by the method given to you. Note that the string in the root (which could be the empty string or `ROOT`) will be printed last now. (24 pts.)

(2) Consider open-address hashing with the standard quadratic-probing rule that if the initial slot $k = \text{hashFun}(\text{item})$ is occupied, the i -th retry is in slot $k + i^2$ (modulo the table size). Use the (admittedly poor) hash function that simply adds up the number values of letters $a = 1, b = 2, \text{etc.}$, as on Prelim II.

- (a) For table-size 8, attempt to insert the words `bad bed bid dad fed gag` in that order. Does your table get `fed` up? Or does it gag on `gag`?
- (b) Now change the table size to the prime number 7 and try again.
- (c) Show and compare to the result of hashing with chaining for both table sizes. (15 pts. total)

(3) Text, section 10.9 of Chapter 10, “Self-Check Exercise 1” on p614, which asks you to diagram how `quicksort` partitions, recurses on, and sorts the array `[55 50 10 40 80 90 60 100 70 80 20]`. Also show how `MergeSort` works on the same array—use list sizes 2 or 3 as the base case and in case of an odd number of elements, make the first list the longer one. (15 pts. total, for 54 hardcopy points)

Final project task and report questions:

Show the task of Assignment 6, using the new phrase file `Guardian998novels.txt` as well as the previous `RS500songs.txt`, and using the main-article-text portion of the following posts:

<http://rjlipton.wordpress.com/2011/10/12/empirical-humility/>
<http://rjlipton.wordpress.com/2011/10/31/an-interview-with-kurt-gdel/>
<http://rjlipton.wordpress.com/2011/10/29/poker-and-cantors-proof/>

You may either manually put the main text into a file or use `wget` as described in labs; for the latter you may prune down to the words between by `rjlipton` or by `KWRegan` and `Open Problems` near the end. [Using `wget` and handling HTML cleanup such as “bleeping over” HTML tags and LaTeX will bring 30 pts. extra credit to both collaborators. Thanks also to a student for suggesting the alternative (inside a `system` command using `#include <cstdlib>` as with `wget`, and note this may work only on `timberlake`):

```
lynx -dump http://rjlipton.wordpress.com > tmp.txt
```

But again, it is AOK to take the text manually.] Use your heap to get the top 20 scores, both by `numChars` and `numWords`, for each of the six combinations, and the overall hot score (the alternative of summing the top 20 scores is fine too). You may break ties arbitrarily, even ties at zero.

Then revise your `WordTree` class to exclude the “stop words” `a`, `an`, `the` and re-run. Note that you are skipping them in the text as well—thus the text words `shining star is born` will match `The Shining` and `A Star Is Born`. Is it quicker? Do the scores change much? Write a report after `main` in the file `HotPhrMMMNNN.cpp` where `MMM` and `NNN` are your initials, along with `WordTreeMMM.h`, `PhraseHeapNNN.h`, `PhraseHeapNNN.cpp` and `HotPhrMMMNNN.make`, with executable name `hotphr`. Submit the word-tree class and `main` with the stop-words feature enabled. Any additional files should have similar naming conventions, `MMM` for word-tree person and/or `NNN` for heaper, both for a collaboration. The report questions enumerated:

- (a) Analyze the running time of the basic Assignment 6 task (not caring about the different combinations yet) analytically using the `WordTree` and the heaps, contrasted to the prior way the text was sliced into phrases (duplicating words) and sorting.
- (b) Show the results with and without the stop-words feature. Did it make any perceptible effect on the running time? Implementing the `HiResTimer`, which works on `timberlake` only, is 15 pts. extra credit for both collaborators.
- (c) Do the scores change much when the stop-words feature is implemented?
- (d) Suppose we were to allow near-matches of individual words, such as cases where the next word in the text is `man` or `towards` while the current word-tree node compiled from phrases has `men` or `toward`. For instance, we could say we’ve matched a child next-word if the edit-distance is at most 1, as in these two cases, or we could use a more-general “similarity metric” to choose a close-enough child—choosing the closest one in case several are near. The question here is: How does that affect the idea (which this project chose not to require) of using binary-search to choose the child?
- (e) Now a larger question on the same theme, for 15 pts. extra credit (both, joint answer is fine on all these questions): Suppose we made our hot-scores depend on a similarity metric of *entire phrases*. This might allow deleting a non-stop word, like considering `Gatsby` a match for the title `The Great Gatsby`. How would that work against the whole `WordTree` idea and its greater efficiency in part (a)?

(60 pts. total for task and report questions, making 300 on the project of which 120 is individual, plus 60 pts. possible extra credit which is separate)