COMS 3137 Data Structures and Algorithms. Homework 4

Submit your electronic files via `http://courseworks.columbia.edu`. Do the written assignments electronically if possible, since it is easier for all parties involved. Use the provided makefile for the programming problems.

Directory structure to be followed for submission:

```
Root folder : <Your UNI>_homework4
<all Java source files>
Readme File (describing all your files and notes to the graders)
Theory solutions (optional)
Makefile (optional if you modified the provided Makefile)
```

I.e., this directory structure should mean that typing "make" in your submission directory will compile your code with no errors. Archive your submission directory using the command (in the containing directory):

```
tar -czvf <Your UNI>_homework4.tar.gz <Your UNI>_homework4
```

1. If you are creating your classes within packages, please maintain the package directory structure during submission.
2. Include the makefile *only* if you need to change the makefile provided to you. However you are strongly advised to conform to the provided makefile.
3. The code should be commented as appropriate or the details should be explained in a readme.
4. If you are handing your theory on paper, do not print out your code.

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.

Late Submissions: Any unexcused late homework submissions will be penalized 10% each day it is late. The penalty begins at the beginning of class the day the assignment is due. The 10% penalties will continue for 3 full days at which point no more late submissions will be graded. If you are submitting late, do not use courseworks and mail your submission to all the TAs.

Upload your archive to the Class Files section of courseworks in the Homework 4 subdirectory. It is also recommended that you keep a pristine copy of your submission folder in case there is any submission error.

1 Written 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.

2 Programming Problems

1. (15 points) **Huffman Compression** Write a program that simulates compression using Huffman's Algorithm and draws the Huffman coding tree to a graphical window. Using your program, compress the dictionary text file `dictionary.txt` and output the resulting Huffman coding tree. Finally, display the space savings. You should write:

   - a main program `HuffmanCompress.java`
   - a Huffman coding tree class `HuffmanTree.java`.

   Your Huffman coding tree class should include a method that draws the tree using the new `Drawgraph.java`. Your main compression routine should read an input file (such as `dictionary.txt`), compute the character frequency of each character and generate an array of the initial forest of single-node coding trees. Then it should build a heap from the array (in linear time) using the textbook's heap class (or your own, if you prefer; not the mysterious built in Java PriorityQueue), and begin the Huffman Tree merging process.

   After the tree is built, your program should display the resulting tree. Then it should re-read the input file and encode it according to the coding tree. Rather than outputting a representation of the tree and the encoded input to a new file, we will only simulate the compression by counting the number of bits necessary to store the compressed file[1]. Finally, compare the resulting simulated file size to the original file size.

2. (15 points) (Based on **Weiss 5.18**) Implement a spell checker by using a hash table. Assume that the dictionary comes from an existing large dictionary (use the file from homework 3). Output all misspelled words and the line numbers in which they occur. Also, for each misspelled word, list any words in the dictionary that are obtainable by applying any of the following rules:

   (a) Add one character.
   (b) Remove one character.
   (c) Exchange adjacent characters.

   Some additional notes:

   - Call your program `SpellCheck.java`. If we are to run your program on our a text file called "`FinalPaper.txt`," should be called like this:

     ```
     java SpellCheck FinalPaper.txt dictionary.txt
     ```

   - Internally, your program should read the dictionary into a hash table. You may use the code from the textbook or the built in Java `HashMap` class.
   - Then you should perform lookups for each word, and if a word is not found, perform the appropriate lookups for the suggestions.
   - Note that while I stripped all non-alphabetical characters from the dictionary file, there are still both upper and lowercase letters.

[1] I'm not asking you to actually output the file because that only makes sense if you'll also write a program to read the file. This would double your work while the concepts you'd be practicing would be redundant.