CSE 332: Data Structures and Parallelism — Winter 2019

P1 Write-up

P1 Due Date: Tuesday, January 22

Only ONE person from your group should submit the write-up to Gradescope. Please make sure that you select corresponding pages for each question when you are submitting your write-up to Gradescope. You should also add your partner to your group on Gradescope after you submitted your write-up.

Project Feedback

(1) Project Experience

Answer the following questions about your experience doing the project.

- How was your partnership? What worked well? What would you do differently next time?
- What was your favorite part of the project? What was your least favorite part of the project?
- Did you enjoy the project? Why or why not?
- How could the project be improved?

WorkLists

(2) `WorkList` Interface

We've implemented several data structures that all share the `WorkList` interface. What is the advantage of having several data structures with the same interface? Why not give each of them their own interface?

Tries

(3) `TrieMap` vs. (`HashMap` and `TreeMap`)

As we've described it, a `TrieMap` seems like a general-purpose replacement for `HashMap` or `TreeMap`. Why might we still want to use one of these other data structures instead?

(4) Applications of `TrieMap`

One of the applications of Tries is an IP address lookup. Network administrators, for instance, customarily need to ban some subset of IP addresses. Consequently, each IP address connecting to the network needs to be checked against this blacklist.

Explain (in very high-level pseudocode) how you might solve this problem with a `TrieSet` or a `TrieMap`. Make sure to detail how a similar solution that uses a `HashSet`/`HashMap` instead would be different and why using a Trie might improve the solution.

Zip

For this part of the write-up:

- You'll need a working `SuffixTrie` to run the ZIP experiments for this writeup. Implementing `SuffixTrie` yourself is part of the Above and Beyond section and not required for full credit. To use our provided SuffixTrie.jar (if you don't want to implement it yourself), right click on SuffixTrie.java in the Eclipse sidebar, choose Refactor > Rename, and name it something else (like SuffixTrieUnused.java) **with "update references" unchecked.**

- Put your test files in the root directory of the project (that is, at the same level as src, not in the src folder).

(5) Running Zip

One of the classes in the main package is called Zip. This class uses your `PriorityQueue` to do Huffman coding, your `FIFOQueue` as a buffer, your stack to calculate the `keyset` of a trie (using recursive backtracking), and your `SuffixTrie` to do LZ77Compression. Find some text file (a free book from https://www.gutenberg.org/ or even the HTML of some website) and use Zip.java to zip it into a zip file. Then, use a standard zip utility on your machine (Finder on OS X, zip on Linux, WinZip or the like on Windows) to UNZIP your file. Check that you got back the original. Congratulations! Your program correctly implements the same compression algorithm you have been using for years!

Discuss in a sentence or two how good the compression was and why you think it was good or bad.

(6) Zip Experiment

Now that you've played with Zip, we want you to do an experiment with Zip.

Notice that there is a constant called `BUFFER_LENGTH` in Zip.java. Higher values of this constant makes the compression algorithm that Zip uses use more memory and consequently more time. The "compression ratio" of a file is the uncompressed size divided by the compressed size.

Compare time, type of input file, and compression ratio by running your code on various inputs. We would like an in-depth analysis.

- You should try at least one "book-like" file, at least one "website-like" file, and some other input of your choice.
- You should also vary the `BUFFER_LENGTH` constant for each of these inputs.
- You should run multiple trials for each data point to help remove outliers.

We expect you to draw meaningful conclusions and have tables or graphs that convince us of your conclusions. In particular, you should explain WHY you think you may have gotten the results you did.
This single question is worth almost as much as the implementation of `ArrayStack`; **so, please take it seriously.**

Above and Beyond

(7) Extra Credit

Did you do any Above and Beyond? Describe exactly what you implemented.