Homework #6: Parallelizing a software engineering analysis

Checkpoint due Monday, December 3rd
Due Wednesday, December 5th

In this assignment you will analyze existing Git repositories to compare projects' development histories. You will then parallelize your implementation to (hopefully) improve its performance. This assignment also serves, in part, as a capstone assignment in this course. It is intentionally under-specified. As you complete your solution you should be sure to demonstrate the software development best practices taught in this course, even though the requirements for the development process are not precisely described here.

Your goals in this assignment are to:

- Practice parallel and concurrent programming in Java.
- Demonstrate software development best practices, including (but not limited to) software design, use of a version control system, build automation, continuous integration, documentation, static analysis, and testing.
- Reuse existing, open-source third-party libraries.
- Learn more about the internal data representation of a Git repository.

This assignment includes a checkpoint deadline (Monday, December 3rd), and is due Wednesday, December 5th. The usual course late policy applies for the overall deadline, but you may **not** use any late days for the checkpoint deadline.

Comparing the history of Git repositories

By the checkpoint deadline (Monday, December 3rd), you must design and implement a sequential (single-threaded) program that, given two Git repositories, finds similar revisions in the two repositories. You must somehow use cosine similarity to measure the similarity of two revisions, but you have great flexibility in how you use cosine similarity. (In Homework 1 you computed a vector for each document based on the word frequencies within the document; here you must compute a vector for each *revision* that somehow represents the changes of the revision (the `git diff`), but we are not specifying how you compute that vector.) Note: For this assignment, we consider a revision to be a relationship between a parent and child commit. In Git it is possible for a child commit to have multiple parent commits, but you may treat each parent-child relationship as a separate revision.

Your solution may assume that the Git repositories are located on the local computer at the start of the analysis. At a minimum, your program must produce a list of the N most similar revisions from two Git repositories (for a configurable N), with each line being in the format:

```
child_hash_1, parent_hash_1; child_hash_2, parent_hash_2: similarity
```

including the SHA1 hashes for the parent and child commits from each repository, where `similarity` is the cosine similarity of the two revisions.

Your sequential implementation is due by the checkpoint deadline, Monday, December 3rd. Please label your work for this milestone with a helpful commit message so we can easily find and evaluate it.

Parallelizing your implementation

Parallelize your sequential implementation to use all cores when run on a CPU. Your solution should demonstrate a basic understanding of the Java concurrency tools; you may use primitive synchronization if appropriate, but your solution should show that you are aware of the tools in `java.util.concurrent` and its various subpackages. Your parallel implementation should be functionally equivalent to your sequential implementation. If possible, your design should allow your sequential and parallel implementations to reuse code without substantial code duplication.

After you initially complete your parallel implementation, benchmark and attempt to improve its performance compared to your sequential implementation. It's worth noting that this part of the assignment might require substantial experimentation to obtain a non-trivial speedup over your sequential implementation. You should plan to complete your solution well before the assignment deadline so that you have time to explore how you might improve the performance of your parallel implementation.

When you are done, describe your strategy for parallelization in a file `discussion.pdf` or `discussion.md`. Explicitly describe what constitutes a unit of work (to be done in parallel with other work) and describe how you achieve safety in the face of concurrency. You should also include a discussion of your benchmarking experiments. List which repositories you used and why you chose them. Describe your benchmarking experiments, how you used them to help improve the performance of your implementation, and any changes you made to your implementation and/or parallelization strategy as a result of those performance benchmarks.

Evaluating your work

When you are done, your solution should include at least one sequential program (with a `main` method) and at least one parallel program (with its own `main` method). You should also turn in a `discussion.pdf` or `discussion.md` file, any other files needed to document your build configuration and software development process. Please remember that we are evaluating your software design and development process in this assignment; include all artifacts needed for us to understand your work, and frequently commit your work so we can see how your solution evolved over time. We encourage you to also include a detailed `README.md` file explaining how we can run your solution.

This homework is worth 100 points. Your work will be evaluated approximately as follows:

- Checkpoint completion by Monday, December 3rd: 5 points.
- Sequential implementation: 20 points.
- Parallel implementation, including your performance discussion: 40 points.
- Demonstrating software development best practices: 35 points.

Hints

The following hints might be helpful:

- It's fine to use whitespace-delimited tokens of a revision to somehow compute the vector representing the revision. You do not need to parse source files or otherwise use complex algorithms to analyze a revision.
- jGit (https://www.eclipse.org/jgit/) is a Java library that provides programmatic access to Git repositories. The `Repository`, `Git`, `RevCommit`, and `DiffEntry` classes are all helpful. You are not required to use jGit, but learning how to programmatically access a Git repository is a non-trivial part of this assignment.
- It's good to test your solution on a variety of small and large repositories.
- For your discussion, the term "speedup" has a formal definition that was not introduced in class. Please use the term appropriately.