Software Architecture

Due Nov 7 by 11:59pm **Points** 20 **Submitting** a file upload **File Types** zip
Available Oct 17 at 12am - Nov 7 at 11:59pm 22 days

This assignment was locked Nov 7 at 11:59pm.

Overview

Software architecture is concerned with arranging the structures of the system (whether these are program elements, like modules, or runtime elements like services, processes, etc.) in order to achieve quality attributes. For example, we might set up the modules of a system so modules use only a few very simple interfaces to other modules, in order to make the system more modifiable (or more testable, if we want to replace one module with another). Architecture often consists of applying certain 'tactics' to arrange structures so as to achieve quality attributes. It's more abstract than design in that it focuses on "the big picture" and imposes constraints more than getting into "how" the aspects of a system work internally.

This assignment has two parts: first, using architecture to improve performance. Second, using architecture to improve testability. The specification for kwic is not changed from last time. You can use your kwic (fixed to do the right thing!) or my kwic code that I posted, as a starting point. The benefit of mine is that it's correct (by definition); the benefit of yours is that you understand it better, and it may be easier to change to have better performance or be more testable.

Because our kwic system is not large, and does not contain multiple modules, calling what we're doing "architecture" is stretching things, but we don't really have time to build a system requiring serious architecture in this class!

Performance

"The goal of performance tactics is to generate a response to an event arriving in a system within some time-based constraint."

Performance tactics are numerous, including ways to manage resource demands (limit rate of events, reduce overhead, increase efficiency) and to manage resources (increase the resources, introduce concurrency, cache results, schedule resource usage).

In this assignment, use performance tactics to modify a kwic implementation to improve performance. For benchmarking performance, I have uploaded in the Files section for Assignment 2, a few large texts on which to run kwic: Joyce's *Ulysses* (big), Proust's *Remembrance of Things Past* novels (bigger), a large chunk of the better-known work of G. K. Chesterton (really big). Proust takes a few minutes to index on my fast Macbook with two cores.

There are two parts of this assignment: implement fastkwic.py, containing a kwic implementation (with the same function name, kwic), that analyzes large texts faster than your kwic.py. You may assume your code will run on a machine with at least 3 cores and a reasonable amount of memory. You can focus on increasing the performance for all inputs, or on performance for inputs that have been seen before. Because it is important to be able to measure quality attributes to ensure and architecture does what it is meant to (and things we can't measure are hard to work with in engineering in general), you will also have to discuss experiments you did to measure your performance gains. You can also discuss what aspects of kwic made it hard to achieve good performance.

Testability:

A quality that is often in tension with performance is testability: how easy it is to control and observe the behavior of a system. Tactics for testability include specialized interfaces for control internal variables of a system, added assertions to check system state, limiting structural complexity, and limiting nondeterminism. Use some tactics for testability to make a version of kwic that is more "testable" in that individual requirements can be more easily tested. This is likely to also make modifiability (e.g. if we wanted to change how listPairs or periodsToBreaks works) better. Document your new testing features.

What to Turn In:

You will be turning in a few files, in a flat directory structure. The files are:

- kwic.py: a working, baseline implementation of kwic
- fastkwic.py: implements kwic, has the same function name and inputs, but has better performance (is faster to return an index)
- fastarch.txt: describes the (changes in) architecture for fastkwic.py, and how they use some tactic to improve the performance (minimum of 400 words); this is also where you describe experiments that checked you really had improved performance
- fastarch.pdf: you can supplement your text with diagrams/graphs (for performance) etc.; the only required content is a diagram showing the architecture of the fastkwic system
- testkwic.py: implements kwic, but improves testability and modifiability by allowing individual parts of kwic to be tested in isolation, corresponding to requirements or the structure of the system
- testarch.txt: describes the (changes in) architecture for testkwic.py, and how they use tactics to improve testability (minimum of 200 words)
- testarch.pdf: you can supplement with diagrams here, too; the only required content is again a simple testkwic architecture diagram

You can name your zip as before, except with assign2 instead of one. The main thing is to name these files correctly, and put them in a flat directory.

You can name your zip as before, except with assign2 instead of one. The main thing is to name these files correctly, and put them in a flat directory structure.

Again, stick to standard Python 2.7.11+ modules.