Daily Coding Problem #59

Problem

This problem was asked by Google.

Implement a file syncing algorithm for two computers over a low-bandwidth network. What if we know the files in the two computers are mostly the same?

Solution

If the files on the two computers are radically different, then we have basically no choice: we must make the sender send over the whole file. We can compress it to save some space, but that's about it.

We can do a bit more if the files are similar. Ideally, we would like to just send over the deltas, i.e. the differences between the two files. However, the problem here is we don't know what's different, so we don't know what to send! So we're back to sending over the whole file.

We know that we can definitely send over deltas -- after all, it's the basis of utilities like `rsync`, and is also widely used for patching games and software! How is it done?

The basic idea is to have the receiver compute a small checksum or fingerprint for non-overlapping blocks of the file it has, and send that over. Then, the sender can just verify, using the same process, which blocks are different, and then send only the data required for those. Now, if the files are identical, we no longer need to send the whole file! We only need to send the fingerprints for the file over, which should be tiny.

Sounds great! But there's one problem: what if the files are of different lengths? Or

worse: they're different lengths, and the appended section is at the beginning of the file. Then all the checksums will be off, and we'll need to send over the whole file again, even if we just appended one section!

The solution to this problem is to change how we're matching blocks. After the receiver sends all the checksums, the sender can compute the checksum at every possible offset to find one that matches. If it does, then we just send all the data from the last point to the beginning of the current block, as well as some sort of signal that the block matched.

P.S. This algorithm is how the `rsync` utility is implemented, and was first described here. It's surprisingly short and easy to read! A few things in the paper that aren't mentioned here:

- Rolling checksums for efficiently computing checksums at every possible offset.
- Using a weak (rolling) checksum and a strong one for efficiency
- Storing the checksums in a hashtable for easier lookup