CS51 Final Project: Team + Algorithm

Due Friday, April 3 at 5PM (Sunday, April 5 at midnight for Extension)

Checkpoint Requirement

Choose a team. For college students: we strongly recommend teams of 4 people. The minimum team size is 2. You will need to find large chunks of time to spend together working, so think about potential schedule incompatibilities.

For extension students: we still strongly encourage you to work with a group. However, we understand that this may not be possible, and we will accept smaller groups (including solo projects). We will adjust our expectations according to the size of the group.

Choose a couple algorithms. As we mentioned in the project overview, we are looking for projects with interesting algorithms or data structures as part of their core functionality.

In past years, the most successful project clearly identify one or more well-known algorithms or data structures that are then implemented. When brainstorming, we suggest you look for interesting algorithms and data structures before you look for a problem to which you could apply them.

We offer examples of past projects below that use algorithms and data structures in diverse ways. You may find even more interesting algorithms by browsing the syllabus of more advanced courses on the internet, such as algorithms, cryptography, compilers, graphics, natural language processing, artificial intelligence, optimization, numerical methods, etc. You're expected to put some thought into finding algorithms or data structures that are both interesting and neither too difficult nor too easy.

Additional places you may begin to look include:

- http://courses.csail.mit.edu/6.854/current/
- http://www.amazon.com/Introduction-Algorithms-Thomas-H-Cormen/dp/0262033844
- http://www.google.com/
- http://www.quora.com/

- http://www.stackoverflow.com/

If you're finding it difficult to gauge how difficult an algorithm is to implement, email your TF or post on Piazza.

You should design your project so that the front-end is optional. Getting up to speed with complicated web or mobile frameworks will really slow you down and ultimately take some focus away from the underlying content. Thus, if you choose to make a web or mobile interface, make it a secondary goal. Furthermore, if you decide on a web front-end, you should figure how you are going to host the website early so that it does not distract from the main goal of the project.

Once your team has met and discussed the above, fill out the Google Form (linked at the bottom of this document) to let us know what you've chosen. The most difficult questions should be along the lines of:

- **Give us a broad outline of your project.**
  What do you want to do? What algorithms or data structures do you plan on using?
- **What resources will you use get started with this project?**
  What books, people, websites, etc. will you reference to gain a better understanding of the algorithms or data structures you are interested in?

Examples of nice past projects

- A document search suite and plagarism detector. Done in Ocaml. [video]
- A version control system, focusing on the rsync and compression algorithms. Done in Java.
- Machine learning for games. Genetic algorithm + neural network to find optimum play for a strategy game. Done in Java.
- A simplex solver with a complete matrix module. Done in Ocaml.
- Collaborative filtering using five algorithms. Done in Ocaml.
- A spam filter using Naive Bayes classification. [writeup]
- Mini-WolframAlpha. Done in Ocaml.
- Comparison of dictionary performance with 6 different data structures. Done in Java. [presentation link]
- SAT solver. Done in Ocaml.
- Content-aware image resizing. Done in Python.
- BSP trees for representing 3D scenes

- Natural language parsing. Done in Python.

Projects to avoid

- Game AI that relies heavily on heuristics
- A web interface without an algorithmically significant backend

---

Submission

Just answer the questions in this Google Form!