Discrete and Algorithmic Geometry

Julian Pfeifle, UPC, 2019

Sheet 0

Due on Thursday, November 7, 2019:

(-7) Learn about the versioning software `git`, and practice until you become comfortable using it.

(-6) `git` only supports versioning, but not issue tracking. However, since we will be writing code and talking about it, this is a desirable feature for us. Several solutions exist for this, and the UPC even has dedicated infrastructure for it. However, this infrastructure doesn't support teaching, so we have to use a commercial service. This year, we will try `gitlab.com`, so please sign up there. Be careful not to use the "Try Gitlab for Free" button (which makes you pay in a month's time), but the sign in link on the main page, which lets you use enough of the company's services for free. There, click on "register".

(-5) Send your chosen username to `julian.pfeifle@upc.edu` so I can add you to the repository.

(-4) Learn how to use `ssh` keys on gitlab, on your account's "settings" page.

(-3) When I reply to your email, you can check out the repository of this course using

```
git clone git@gitlab.com:julian-upc/2019-dag-upc .
```

(-2) Create a new branch `your-name-cv` in the repository, and edit the file `participants.tex` to include a short cv and some information about your mathematical interests. Then `commmit` and `push` your changes. When you feel your work is done, log in and create a merge request at the relevant issue page so that all the different stories may be merged.

(-1) Organize into teams of 2–3 people to work on the exercises, and edit `participants.tex` to reflect this. As always, `commit` and `push` your changes, and create a merge request at the gitlab instance.

Due on Tuesday, November 12, 2019:

(0) Read up on **two** programming languages of your choice that your team will use in this course. One of these should be a scripting language for rapid iteration, the other a compiled language for efficiency. If you have never programmed before, a good choice for a scripted language is `python`/`sage`, and a good choice for a compiled language is `julia`. If you already know some languages, take the opportunity to learn a new one! Some suggestions are `c++`, `perl`/`raku`, `rust`, `haskell`, `ocaml`, `ruby`, `lisp`/`scheme` (I myself have not used all of these).

Put your results and observations into the wiki of the class.

To submit your solutions to the next two exercises,

- ▷ create and switch to a branch `your-awesome-team-name-sheet-0`,
- ▷ create a subdirectory `exercises/sheet0/your-awesome-team-name/coding`,
- ▷ and `add` **all** files you create to your commits, **without encryption**.
- ▷ Record your results in `sheet0/results.tex`, and create a merge request.

This will make it possible to have conversations about your code in your merge requests.

When developing your programs, bear in mind that since we all share the same repository, your code is likely to be read by the members of other teams who would like to compare the language they're using to yours. Therefore, please take the opportunity to document your code very well, so as to make it as easy for them as possible to understand what you're doing! In return, you'll be rewarded by awesomely documented code in languages you didn't have time to learn, but where you know exactly what problems it is solving.

Use the facilities that `git` offers to synchronize and collaborate on your code across devices and operating systems.

(1) A set system $\mathscr{B}$ is the set of bases of a matroid if it satisfies the following axioms:
   (i) $\mathscr{B} \neq \emptyset$
   (ii) (*Exchange axiom*)
   Given $B_1, B_2 \in \mathscr{B}$ and $x \in B_1 \setminus B_2$, there exists $y \in B_2 \setminus B_1$ such that
   $$B_1 \setminus \{x\} \cup \{y\} \ \in \ \mathscr{B}.$$

In the programming languages of your choice, write code that checks whether the sets of integers contained in the directory `exercises/sheet0/matroid-or-not` satisfy the matroid basis axioms or not. What is the combinatorial complexity of your code? What parameters of the data does this combinatorial complexity depend on?

(2) Write (or use, or search for and download) code that given integers $n \geq k \geq 0$ creates all $\binom{n}{k}$ combinations of an n-set. They form the set of bases of the uniform matroid of rank k on n elements. Run your code on various instances of these matroids, and plot the execution time against reasonable parameters. Is your conclusion from part (1) borne out?