Outlab 2: LaTeX and gdb

Your names

Autumn 2021

Contents

1 Introduction

In this assignment, we will learn our way around with two useful tools: LaTeX, to typeset technical documents (and especially make resumes look *cool*[1]), and `gdb`, to introduce some method to the madness of debugging code.

LaTeX, a markup language, is part of the TeX typesetting system created by the immortal Donald Knuth. You put "code" - which is really markup commands and tags that determine how your content is displayed, in a "source file", which has a `.tex` extension. You then need a "compiler", or software that is formally called a TeX distribution, to actually render the marked up content into a portable document - literally, a PDF. Although PDF is indeed most common, other formats are possible.

Why go through all this effort? Admittedly, plain old Google Docs serves most of our purposes, and there is a reason why it is the most popular alternative. However, LaTeX proves its edge when we want to write involved mathematics. Its "environments" are handy to automate the process of organising large documents. It's indispensable when we want to render domain-specific scientific illustrations.

The TeX distribution and packages you will need for this assignment are already installed in the Docker image provided to you. Since you're doing the assignment in teams, you can also collaborate online on Overleaf, with minimal setup. Refer here if you want to install independently.

`gdb` - The GNU Debugger is a tool everyone wishes they'd learnt to operate earlier. You'll be writing a lot of C++ code this semester, and running your programs with `gdb` will make your job of tracing and squashing bugs a lot easier. `gdb` gives you the power of the gods - you can set breakpoints to pause executables, run them line by line, step into functions, see the program stack, print out variables at any point in time... we'll learn that with experience, this assignment just has a small (and hopefully fun) introduction.

`gdb` is already installed on the Docker image we provided, but it's fairly standard software, and if you don't have it, it's just an `apt-get install` (or whatever equivalent) away. You can look up `man gdb` on the command line, or on the internet, if you need any help.

1.1 Evaluation Logistics

This assignment is **manually graded** and is intended to be chill - it is meant for you to get acquainted with software you'll be using a lot, and of course you're going to get a lot better at them on the job.

For the assignment, you have to create a PDF, following the instructions in the next section. You will turn in the `.tex` file, `.bib` file, and your images, so that we can generate the PDF without any errors.

In your submission directory, write a shell script called `typeset.sh`, which, when run in the Docker container, correctly produces your intended PDF output. You'd have to invoke `pdflatex` more than once to get the cross references rendered correctly. This script is to prevent any misunderstandings: we will look at the shell script, so please don't suppress the output of the commands you invoke.

[1] Ok, you're going to jugaad a template from a senior, who jugaaded it from another senior, who... but you need to know LaTeX to edit it comfortably

2 LaTeX features we demonstrate

Before we jump in to the heart of the assignment, here's a list of the LaTeX features we demonstrate. In order to adjust the spacing between bullets, you can specify the `itemsep` parameter as an optional argument to the `itemize` environment. You will need the `enumitem` package for that.

- Making lists
- "Writing **bold**, *italicized*, `verbatim` (hint: `\verb`) text in quotes"
- Cross references within the document, hyperlinks[2]and footnotes
- Making a title, sections, and automatically generating a table of contents
- Setting up the page layout with header and footer
- Typesetting mathematics
- Theorem environments with the `amsthm` package
- Managing citations with a `bib` file
- Writing down algorithms
- Inserting tables and pictures
- (Bonus) Custom environments, "listing" code

2.1 Creating the PDF

The paper size for the document is A4. The left and right margins are 1.2 inches each; the lower margin is 1.5 inches. The `geometry` package is very convenient to set up and manipulate these dimensions. Week 2 Notes in the tutorial will be helpful to set this up.

Your sections will have the same names as the original, however the numbering will obviously be different. In the section **"Basic Theory..."** you have to copy the original as it is. Typesetting Algorithm 2: `bottomup` counts towards bonus credit, and you can choose to omit it if you're sure you've done everything else perfectly, or if you'd rather spend your time on something else, like other courses, or having a life.

In the section **"Programming"**, you just have to follow the guidelines and report what is asked for in the PDF. It is summarised in bold at the end of that section.

Section **"Hints and tips"** is for bonus credit: should you choose to attempt it, you only need to copy any one LaTeX listing and make a reference to it; no need to copy the other text.

Most of what you would need is covered in the notes for Week 3 in the tutorial.

We of course don't have the time and resources to check every tiny detail of your PDF, and we certainly don't intend to grade strictly but try to be as faithful as you can, because it's not that hard to notice if something is seriously off either. In particular, make sure that there are no errors while typesetting (Overfull hbox kind of warnings don't count as errors)

The above list is more or less what we're looking for. The credit is capped at full marks, but attempting the bonus well will compensate if you happen to make mistakes elsewhere.

One suggestion: in the following section, there will be long maths commands you'd have to type over and over again. To avoid the drudgery, make your source more readable and your job more convenient, define "macros" with `\newcommand`

[2]The link is an Easter Egg, and shameless self promotion. You needn't copy this page and the Introduction in your submission.

3 Basic Theory of Linear Recurrences

We all know about the Fibonacci Sequence, given by the recurrence

$$f_{n+2} = f_{n+1} + f_n$$

and initialised as $f_0 = 0$, $f_1 = 1$. This is an instance of what is called a Linear Recurrence Sequence, or LRS.

Definition 3.1. *LRS $(u_n)_{n=0}^{\infty}$ of order k over field $\mathbb{F}$ is the sequence determined by the recurrence*

$$u_{n+k} = \sum_{j=0}^{k-1} a_j u_{n+j} \tag{1}$$

for integers $n \geq 0$ with $a_0, \ldots, a_{k-1} \in \mathbb{F}$, $a_0 \neq 0$, and an initialisation vector

$$\begin{bmatrix} u_0 & \ldots & u_{k-1} \end{bmatrix}^T \in \mathbb{F}^{k \times 1}$$

Thanks to Definition 3.1, when we say that we are given $(u_n)_{n=0}^{\infty}$ of order k over $\mathbb{F}$, we actually mean that we are given the $2k$ constants $a_0, \ldots a_{k-1}, u_0, \ldots, u_{k-1} \in \mathbb{F}$.

The Fibonacci sequence shows up a lot in pop culture and DSA assignments, however, the study of LRS is a vast subject in its own right, with applications in software verification, quantum computing, formal languages, statistical physics, combinatorics, and theoretical biology, to name a few.

3.1 Computing terms of an LRS

For a fixed k, assuming we can do arithmetic operations in $\mathbb{F}$ in constant time, how long does it take to compute the n^{th} term of a given LRS?

Algorithm 1: Naive first attempt naive

Data: LRS $(u_n)_{n=0}^{\infty}$, n
Result: u_n

```
1 if n < k then
2 |   return u_n
3 end
4 return sum_{j=0}^{k-1} a_j · naive((u_n)_{n=0}^{∞}, n − k + j)
```

By the way, notice the use of `\mathsf` to write "naive". This algorithm, unfortunately, is painfully inefficient: as you can see, the number of recursive calls will be exponentially many in n. This immediately prompts a more "bottom-up" [3]approach: we start from the first k terms, use them to compute the next term, and iterate this way upto n. This takes $\mathcal{O}(n)$ iterations.

But we can do better, in fact, we only need $\mathcal{O}(\log n)$ iterations.[4]

Technically, even Algorithm 2 is *inefficient*: when we analyse complexity, we do it with respect to the size of the *bit representation* of the input, i.e. how many bits it takes to specify the input. An integer n takes $\log n$ bits to represent, so the parameter for describing complexity is not n, but $\eta = \log n$.

[3]Not to be confused with the drinking phrase
[4]In complexity analysis, the base of the logarithm is implicitly taken as 2

Algorithm 2: Bottom up dynamic programming approach bottomup

Data: LRS $(u_n)_{n=0}^{\infty}$, n
Result: u_n
if $n < k$ **then**
  return u_n
end
$\mathsf{circularArray} \leftarrow \{u_0, \dots u_{k-1}\}$
$\mathsf{arrayStartIndex} \leftarrow 0$
$n_{last} \leftarrow k - 1$
while $n_{last} < n$ **do**
  $\mathsf{nextTerm} \leftarrow \sum_{j=0}^{k-1} a_j \cdot \mathsf{circularArray}[(\mathsf{arrayStartIndex} + j)\%k]$
  $\mathsf{circularArray}[\mathsf{arrayStartIndex}] \leftarrow \mathsf{nextTerm}$
  $\mathsf{arrayStartIndex} \leftarrow (\mathsf{arrayStartIndex} + 1)\%k$
  $n_{last} \leftarrow n_{last} + 1$
end
return $\mathsf{circularArray}[(\mathsf{arrayStartIndex} + k - 1)\%k]$

Yes, we can compute the n^{th} term with $\mathcal{O}(\eta)$ operations, and the trick here is a method called **iterated squaring**. Given an LRS $(u_n)_{n=0}^{\infty}$, define its companion matrix $\mathbf{M} \in \mathbb{F}^{k \times k}$ as

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ a_0 & a_1 & a_2 & \dots & a_{k-1} \end{bmatrix} \tag{2}$$

Compare equations 1 and 2 and observe that

$$\mathbf{M}^n \begin{bmatrix} u_0 \\ \vdots \\ u_{k-1} \end{bmatrix} = \begin{bmatrix} u_n \\ \vdots \\ u_{n+k-1} \end{bmatrix} \tag{3}$$

Iterated squaring hinges on this ridiculously trivial observation: consider the (unique!) binary representation of n: if you append a 0 to it, it becomes $2n$, if you append a 1, it becomes $2n + 1$.

Given $\mathbf{M}$, to compute $\mathbf{M}^n$, all we have to do is start with $\mathbf{I}$, read the binary representation of n from most to least significant: if at some point we have $\mathbf{M}^j$, $\mathbf{M}^j \cdot \mathbf{M}^j = \mathbf{M}^{2j}$, $\mathbf{M}^{2j} \cdot \mathbf{M} = \mathbf{M}^{2j+1}$

In Algorithm 3, we use a data structure called a **stack**. The constant time operations it supports are

- pushing an element onto the top of the stack
- popping an element from the top of the stack (i.e. deleting the topmost element)
- reading the topmost element
- checking whether the stack is empty

As you can see, it's Last In, First Out.

The division while loop gets the bits of n from least to most significant, the iterated square while loop uses the bits from most to least significant. A stack fits the bill.

Algorithm 3: Iterated squaring approach efficient

Data: LRS $(u_n)_{n=0}^{\infty}$, n
Result: u_n

```
quotient ← n
operationStack ← {}
M ← companion((u_n)_{n=0}^∞)
x ← [u_0 ... u_{k-1}]^T
A ← I_{k×k}
while quotient ≠ 0 do
    push(operationStack, quotient%2)
    quotient ← quotient/2
end
while operationStack ≠ {} do
    A ← A · A
    if top(operationStack) = 1 then
        A ← A · M
    end
    pop(operationStack)
end
y ← Ax
return y[0]
```

3.2 Problems associated with LRS

Surprisingly, the following rather simple *decision problem*[5], referred to as the Skolem problem, has been open for the last eight decades or so.

Definition 3.2 (Skolem problem)**.** *For the arbitrary LRS $(u_n)_{n=0}^{\infty}$ whose description is given as input, does there exist an integer $n \geq 0$ such that $u_n = 0$?*

This has a bunch of equivalent formulations, and by being open, we mean that nobody really knows of an algorithm that can decide it (the algorithm should terminate and give the correct answer for *all* possible inputs), or whether there is such an algorithm at all.

We can also consider another related problem for LRS over fields where the $>$ operator is defined:

Definition 3.3 (Positivity problem)**.** *For the arbitrary LRS $(u_n)_{n=0}^{\infty}$ whose description is given as input, is it the case that for all integers $n \geq 0$, $u_n \geq 0$?*

Deciding even special cases of these problems (i.e. restricting what kind of LRS can be fed as input) requires sophisticated maths, like Kronecker's theorem on Diophantine approximations, [2, Chap. 7, Sec. 1.3, Prop. 7], and profound properties of "algebraic" numbers [3] and then some more mathematical arsenal [1, 4].

If you found this short write up interesting, here is a survey talk about the problem, given not so long ago.

[5] Given an arbitrary input, answer a particular question about it with Yes or No

4 Programming

In this section, we describe what we observe while running code (read the comments before `int main` for how to use) that tries to implement Algorithm 3 of Section 3.1. So we go

```
# g++ iterate_buggy.cpp -o executable
```

and then

```
# ./executable <n> <k>
```

and boom, there's a segfault (i.e. the program is trying to access a memory address that isn't allocated to it). We don't really need to go scanning through the code, inserting print statements. We're going to use `gdb`. So we go,

```
# g++ -g iterate_buggy.cpp -o executable
```

and then

```
# gdb -tui --args executable <n> <k>.
```

Hit enter till the code shows up on the interface. At this point, take a screenshot.

Now, enter

```
(gdb) r
```

in the prompt, and the program runs, and segfaults. You can track the offending line exactly with

```
(gdb) bt
```

which is short for backtrace, and in this context, has nothing to do with online semesters. This command prints out the program stack, and you can see the sequence of function calls that led to this point. Take another screenshot of this output. In this example, the mistake is pretty trivial, and you can fix it.

You now have two screenshots, put them in your PDF side by side, like this: (we will put two unrelated images in the original as placeholders)

Figure 1: Aristotle: The first formal logician

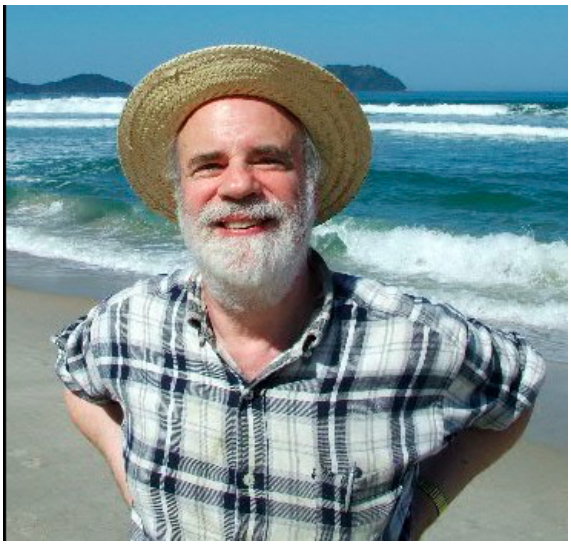

Figure 2: Saul Kripke: we've come a long way since then.

Aristotle image source
Saul Kripke image source

For correctness, you can conveniently check with Fibonacci, as the answers are easily available online. It turns out, that after your small fix, the code exits normally, but gives the wrong answers.

What you can do is navigate the program, line by line. Set breakpoints where you want to pause, it can be a line number, or the name of a function, like

```
(gdb) b 42
```

or

```
(gdb) b main
```

If you have paused, and want to fast forward upto some line, say 42, you can issue

```
(gdb) until 42
```

Or if you want to continue till the next breakpoint, simply

```
(gdb) c
```

To execute the next line, without stepping into the functions, you go

```
(gdb) n
```

If you want to step into the function call present in the next line, you go

```
(gdb) s
```

instead. You can print pretty much any expression (even entire vectors or STL containers) with

```
(gdb) print <expression>
```

Instead of breakpoints, you can set watchpoints with variables: pause whenever a variable is modified

```
(gdb) watch <variable>
```

You can remove breakpoints and watchpoints:

```
(gdb) info breakpoints
(gdb) info watchpoints
```

Note the number in the `Num` column, and then issue

```
(gdb) delete <number>
```

Armed with this knowledge, and whatever else you learn from `man gdb`, can you find out why the code gives wrong answers?

Once you're sure you've fixed everything, consider the last three digits of your roll numbers. Add the numbers (< 1000) they form. Let this sum be m. Consider the smallest multiple of m that exceeds 300. Let this be n. Use the code to compute the n^{th} Fibonacci term, set a breakpoint on line 68, and complete the following:

```
(gdb) print working
copy paste verbatim the exact output gdb gives
(gdb) print operations
copy paste verbatim the exact output gdb gives
```

Make this table (one significant digit is ok), we will partially make it for 719.

operations	**working**
{1, 1, 1, 1, 0, 0, 1, 1, 0, 1}	{1, 0, 0, 1}
{1, 1, 1, 1, 0, 0, 1, 1, 0}	{0, 1, 1, 1}
{1, 1, 1, 1, 0, 0, 1, 1}	{1, 1, 1, 2}
...	...
{}	{5e+149, 8e+149, 8e+149, 1e+150}

Table 1: Printing everything! Note the alignment

For this section, you only have to put the two screenshots, the two gdb outputs asked for, and the table in your PDF.

5 Hints and tips

Typesetting the algorithms can appear daunting, and is not covered in the tutorial, so we've given you boilerplate code for that. Note that this also helps you along with some symbols, indicates the usage of `\mathsf`, and encourages the use of macros.

Listing 1: [LaTeX]TeX
Algorithm Boilerplate

```
\documentclass{article}
%...
\usepackage[linesnumbered,ruled,vlined]{algorithm2e}
%...
\newcommand{\function}{\mathsf{function}}
%...
\begin{document}
%...
\begin{algorithm}[ht]
\caption{...}
\SetAlgoLined
\DontPrintSemicolon
\KwData{...}
\KwResult{...}
Statement \;
%Let magic happen here
$a \gets b \cdot \function(c)\%d$ \;
\end{algorithm}
%...
\end{document}
```

Listing 2: [LaTeX]TeX
Packages imported in the original

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[a4paper,hmargin=1.2in,bottom=1.5in]{geometry}
\usepackage[parfill]{parskip}
\usepackage{hyperref}
\usepackage{fancyhdr}
\usepackage{enumitem}
\usepackage{amsmath}
\usepackage{amsthm}
\usepackage{amssymb}
\usepackage[linesnumbered,ruled,vlined]{algorithm2e}
\usepackage{listings}
\usepackage{xcolor}
\usepackage{floatrow}
\usepackage{graphicx}
\bibliographystyle{plainurl}
\bibliography{biblio}
```

To earn bonus credit, your PDF should include this listing, and a reference to it, like so: Listing 1. The lesson on Advanced Formatting in the tutorial may help you go about defining such a custom environment and a counter with the help of `\lstnewenvironment`

Also, don't worry if you can't place the algorithms exactly where they are with respect to the text in the original. The entire section tells a story, and if you place the algorithms intuitively, you'll get more or less the same result, and that's perfectly fine by us.

Just make sure to start every section on a new page.

Bell and Gerhold, and Renegar wrote journal articles, your bibliography entry should have the author, title, journal, volume and year (page numbers optional).

Bourbaki wrote a book, the fields in your entry will be author, year, title and publisher.

Mignotte wrote a conference article, include author, year, title, booktitle.

The source project has the following files:

- aristotle.png
- biblio.bib
- kripke.png
- main.tex
- typeset.sh

The original typeset.sh (you can create your own but **don't suppress output**)

Listing 3: Bash
typeset.sh

```
#!/bin/bash
pdflatex main
bibtex main
pdflatex main
pdflatex main
```

References

[1] J. P. Bell and S. Gerhold. On the positivity set of a linear recurrence. *Israel Jour. Math*, 57, 2007.

[2] N. Bourbaki. *Elements of Mathematics: General Topology (Part 2).* Addison-Wesley, 1966.

[3] M. Mignotte. Some useful bounds. In *Computer Algebra*, 1982.

[4] James Renegar. On the computational complexity and geometry of the first-order theory of the reals, part i: Introduction. preliminaries. the geometry of semi-algebraic sets. the decision problem for the existential theory of the reals. *J. Symb. Comput.*, 13:255–300, 1992.