Advent of Code

Day Eight

LucidBrot

August 2020

1 About

The task at adventofcode 2019 day 8 is fairly straightforward itself. It can be summarized as

> Read the input line of N numeric characters into *layers* of size $width * height$ (which are known) to find the layer that contains the lowest number of zeros. Then return the number of `'1'` digits multiplied by the number of `'2'` digits within that layer.

However, we're doing this in LaTeX, which is typeset in spongebob-case for a reason.

> "English words like 'technology' stem from a Greek root beginning with the letters TEX...; and this same Greek word means art as well as technology. "

Figure 1: – Donald E. Knuth [15]

If you would like to see another, less verbose, solution that is done in LaTeX, SanseroGames has published a cleaner approach using arrays [19].

Contents

2 The LaTeX Ėẋṗėṙiėṅċė

First of all, we're doing something that it was not meant to be used for – so that means we never get the search results we want. Searching about arrays in LaTeXfor example gives you an explanation about how to typeset matrices. Very useful, but not what I wanted. Thankfully, the pgfplots sourceforge page contains a pdf with *Notes On Programming in TEX*.

Secondly, there don't seem to be any variables. Just *counters*, *counts* which are the TeXversion. and *ifdefs* and most importantly *macros*. But I did not read up on the internals of TeXand LaTeX, so I have no clue about the exact way that macros are evaluated. Sometimes you can define a command that works perfectly well for a constant argument, but if you dare use it on the result of another command, you're being had from multiple directions. Because that result has not already been evaluated (expanded) and is passed as-is into the other command. My version of pdfLaTex does not feature the primitive `\expanded` yet. Using `\expandafter` feels very clunky. Luckily there's a hack around that to be found here. And sometimes the problem was actually the `xstring` package which also breaks the hack.

> The macros of this package are not purely expandable, i.e. they cannot be put in the argument of an \edef. Nestling macros is not possible neither.
>
> For this reason, all the macros returning a result (i.e. all excepted the tests) have an optional argument in last position. The syntax is `[ name ]`, where name is the name of the control sequence that will receive the result of the macro: the assignment is made with an \edef which make the result of the macro name purely expandable. Of course, if an optional argument is present, the macro does not display anything.[1]

After eliminating some problems of this sort by storing the result in a new command by virtue of the optional argument, the same problem still appeared because some commands just don't work due to the same issue, even if they are making use of the optional argument to return that in turn (See Figure 2, Figure 3).

(Btw, I have used `\autoref` above, for the second figure reference, instead of `\ref` and that is pretty cool.)

Finally, the performance of the `xstring` package is whack. It takes more than two minutes to figure out the length of a 15'000 character string. The bash command `wc -c inputfile.txt` does that in less than a second.

```
\def\getchar[#1]#2{%
\StrMid{#1}{#2}{\numexpr #2 + 0\relax}[\mychar]%
\mychar}
```

Figure 2: This command does not like to be used on a non-constant string.

```
@@ -143,8 +143,8 @@ \section{Introduction}
143 143           % assign current char
144 144           \def\currentchar{\getchar[\fileline]{\digitctr}}
145 145       Char Char Binks: \currentchar\\
146     -     %   % check if zero
147     -     %   \IfEq {0}{\currentchar}{
    146     +     % check if zero
    147     +     \IfEq {0}{\currentchar}{
148 148             \advance \currentlayerzerocount 1
149 149             Advanced currentlayerzerocount to \the\currentlayerzerocount
150 150
@@ -157,9 +157,9 @@ \section{Introduction}
157 157               \digictr={\the\numexpr \layersize * \currentlayer + \layersize}
158 158           Layer \the\currentlayer has more zeros than the current best layer (\the\bestlayer ) so we skip ahead to character at index \digictr to start the next layer.
159 159               \fi
160     -     %       }{%else
    160     +         }{%else
161 161                 The current char \currentchar~does not equal 0. It is \meaning\currentchar whereas 0 is \meaning0.
162     -     %   }%fi
    162     +     }%fi
163 163           \ifnum \digitctr<\inteval{\layersize * \currentlayer + \layersize}
164 164           \repeat
165 165           % if there were very little zeros, we can update the best layer
```

Figure 3: The difference between wrong code that compiles (red) and seemingly correct code that produces a compiler error (green).

3 StrLen

Since `xstring`'s `StrLen` is so slow, how about creating a faster one? We'll just have to run tex with the `--shell-escape` flag.[2] (See Figure 4). Using that allows us to escape to the shell - which is either bash or the windows cmd.exe.

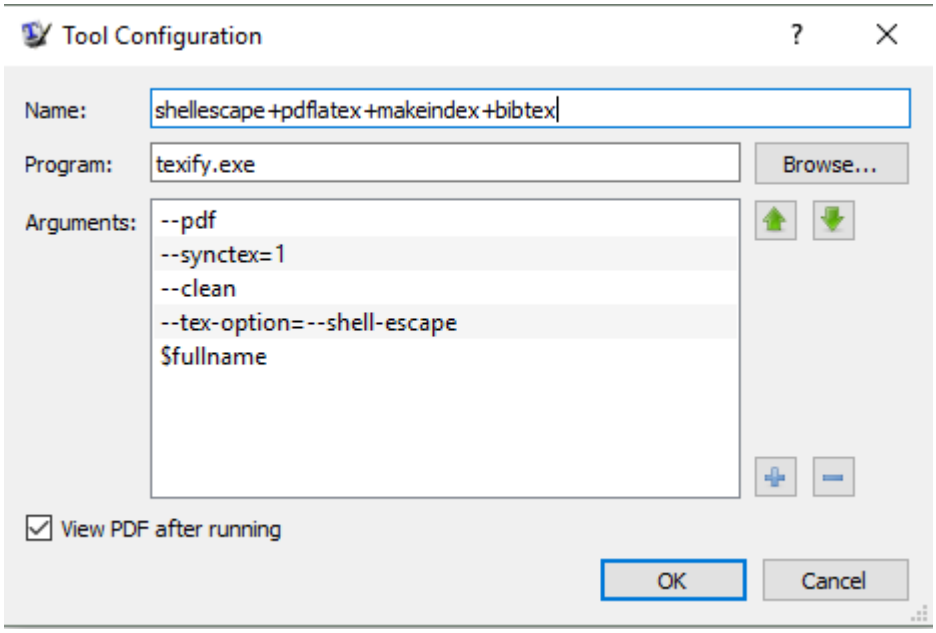

Figure 4: TeXworks settings for shellescape

> Somehow, pdflatex threads kept running despite me stopping them in TeXworks or even closing TeXworks. For further debugging, the rest of this section is skipped for now, hence. Currently, the section is being run.
> But as it turned out, skipping this section about shellex helped nothing at all regarding this issue... It happened with the real input, but not with the `\debug` input.

A simple `\input{|"echo test"}` already works! test!

But I cannot figure out how to correctly call `wc -c inputfile.tex` because the shell spawns in the wrong path (`AppData/Local/Temp/mik53499/_src`). And also, for some reason the following code does not even create a file `outfile.blubb` anywhere on my machine.

```
\input{|"echo a > outfile.blubb"}
```

The problem seems to be, according to the logs, that the pipe closes before the left side is finished writing to stdout. But that is actually happening due to the `echo` earlier. When I leave it out, that's not logged.

Maybe using python is easier? You'd have to read the source to get this one though.[3]

```
\begin{pycode}{abc}
print(1+12)
\end{pycode}
```

But no, that also results in a problem with writing to a file... So let me try something I understand!

```
\input{|"python -c "print(1+2);""}
```

And as you should be able to see, it works! It does output some `SyntaxError` in the Console output, but it seems like I can ignore that?!

So for computing stringlength, a quick python call should speed things up.

```
\ input{|"python -c "print(len('mystringofunkownlength'));""}
```

It is notable that spaces within that string get lost before they are passed to python. Thankfully we don't need this here. But this reminds my of PyAu-Calc. The loss of spacing also explains why some commands don't work. E.g. importing a python module without spaces is not trivial.

We can make LaTeX paste a string into that as well...

```
\def\mystr{hello}
\def\mystrtwo{\mystr}
\input{|"python -c "print(len('\mystrtwo'));""}
```

...but for some reason it fails when used with our included inputstring. There's a site with an example on how to include text from a file, but it does not work at all for me[4]. I guess I'll just hardcode it inside this file here instead... For that, however, the line length limit becomes an issue. So I've declared 74 commands and combined them into one. Aand turns out that also didn't help. The string just ends after some two hundred characters.

So to quote a professor of mine, J. Hromkovic,

> *"**Strategy: We Give Up!** What can we do to still be able to state something impressive?"*

Well, I can just hardcode the damn string length. That's not really worse than hardcoding the input.

During Part 2, I'm revisiting this section again, though. because `StrMid` is just so very slow! Now I could of course hardcode a command that returns the correct character based on the argument, but isn't that a bit too cheaty? Maybe. But my program technically works, it's just too slow. If you want one that does not cheat, check out Sansero's repo.

Hardcoded string char access: blubb 2 blubb

(0)[1], (1)[2], (2)[2], (3)[2],
The output should have been: 1, 2, 2,2

While working on this, I have found an understandable explanation for `\expandafter` [10]. But maybe that's not my problem. Maybe my problem is once again that I'm defining things in a loop.[11, 12] Maybe those approaches would be a good idea: Create distinct macros in each loop iteration instead of overwriting the same one each time (and failing at overwriting it).

But nevermind! It turns out I did everything well, except that I started my test output at a different index than I thought... the last few hours were spent on a ghost bug.

4 Get Head Performance

Again, `xstring` is extremely slow with big strings. So I split everything into layers of 100 chars. But that's still noticeably slower at the end of the layer than at the start of the layer... even though it's only supposed to be a quick character access. So I'm transforming everything into head accesses at index zero.

5 Scoping

Nested loops require scoping around the inner loop. Which in turn means we need to use the `\global` keyword to assign to variables from outside the inner scope. And that in turn makes it really weird to use `StrGobbleLeft` from the `xstring` package when I'm trying to remove a character and store the result back in the same string.

The solution is probably a rewrite that uses only one loop plus an if condition that checks the modulus of the loop counter and acts appropriately whenever a layer is finished.

6 Execution: Getting our Feet Wet

We had 34 Strawberries for this year's harvest. Probably not enough. So we are sad now and solve https://adventofcode.com/2019/day/8.

h e

1

6

122222222021222212220220022122202222202201222222222222202202212222222021000022222222222211222222222

hello world 3 3 300

Image Width: 25 Image Height: 6

I want to loop 150 times for the first layer.

The input file contains 14999characters.

```
\newcounter{outerloopcounter}
\setcounter{outerloopcounter}{0}
\loop
        \addtocounter{outerloopcounter}{1}
           \theouterloopcounter ,
           \ifnum \value{outerloopcounter}<3
\repeat
outerloopcount: \theouterloopcounter
```

Figure 5: Simple Loop using TeX Counters

7 Execution: The Water Is Cold

> \global is a TeX command that declares the following definition or assignment to be global, meaning that if TeX is currently inside a group, the definition or assignment will still remain valid when the group is over. Commands that can follow \global include \def, \edef, \let, \count, \countdef, [...][5]

An alternative to using globals would be to use tikz loops with the `remember` option [6]. Another option I see is using counters instead of counts - they are the latex version of the tex counter. For counters, you can apparently not use \the \ctr, you have to use\thectr (See Figure 5).

Now let us do the same thing again, but with nested loops. See Figure 6. That requires curly braces around the inner loop. Otherwise, only the first outer loop iteration is run.

For the actual implementation, we assume that we are given `inputstringlength`, `imgwidth`, and `imgheight`. Computing those from the input string is feasible but takes long.

I'm using an `edef` to overwrite the `workingline` variable every loop.

Computation starting with 100 Layers of size 25x6.

Normally, I would do a first loop for finding the best Layer and a second block for extracting the relevant information. However, given that the loops are so slow with all those string operations, I'll just spend some brain overhead and do this in one loop, even if it is a bit harder to keep the overview.

```
\newcounter{outerloopcounter}
\setcounter{outerloopcounter}{0}
\newcounter{innerloopcounter}
\loop
        \setcounter{innerloopcounter}{0}
        \addtocounter{outerloopcounter}{1}

        {\loop
        \addtocounter{innerloopcounter}{1}
           (\theouterloopcounter , \theinnerloopcounter )
        \ifnum \value{innerloopcounter}<2
           \repeat }

           \ifnum \value{outerloopcounter}<3
\repeat
outerloopcount: \theouterloopcounter
```

Figure 6: Nested Loops using LaTeX Counters

outerloopcount: 100
Best Layer: 7 with 6 zeros.
It has 12 Ones and 132 Twos.
This is a total of $n_{\text{ones}} \cdot n_{\text{twos}} = 1584$.

8 Day Eight Part Two: Because this task wasn't that hard

> Each image layer consists of numbers that represent pixel information. One digit, one pixel.
>
> $$0 \rightleftarrows \text{black}$$
> $$1 \rightleftarrows \text{white}$$
> $$2 \rightleftarrows \text{transparent}$$
>
> You're given multiple layers. For each pixel coordinate pair (x, y) find the *first* nontransparent value.

Not running part two (first try) as per flags because compilation errors remain. See a later section for my second try.

Let me just perform some magic - Consider Figure 8. There's only one problem with that: It assumes that the input strings are constant. Since I input the value of a counter, mine are not, and the magical string that ends up being collected is just a repetition of the last value it had. Adding `\immediate` and `\edef` was not helpful.[7, 8]

Now of course, I don't actually need that collection - I can place my computation wherever I like. It would still have been cool though - partly because I have weird spacing when I output my computation character by character, and partly because this is a nice trick to have in my spellbook at the ready. Luckily, the author of that trick replied to my inquiry by adding an `\edef` and some `\expandafter` (Figure 7).

After playing around with creating my own command for accessing my input string quickly by index, I've come upon the problem that while it worked perfectly for printing the results to the PDF, it caused syntax errors when I tried to reuse the results. E.g. in another `\edef` or `\typeout`. I ended up asking about this on stackexchange and was explained that because `\edef` evaluates its contents, if its argument contains another `\edef` then it will evaluate to itself and the argument of the inner edef will be an undefined command. [14]

@lucidbrot

```
\def\myappendxstring#1{
  \edef\tmp{#1}\expandafter\g@addto@macro%
    \expandafter\mystring\expandafter{\tmp} }
```

Then,

```
\stepcounter{mycounter} %
  \myappendxstring{Counter value is \themycounter. }
```

- Steven B. Segletes

Figure 7: S.B. Segletes' addition.

```
% Throughout the document, collect the variable using
\myappendstring{hello}
% display the computed variable here
\makeatletter
\@starttoc{xyz}% executes stuff in .xyz file
\makeatother
\thefinalstring
% the following line must be performed at
% the end of the computation - e.g. at the
% end of the document.
\addtocontents{xyz}{\gdef\protect\thefinalstring{\mystring}}
```

Figure 8: Magic to collect a variable, then print it at the start of the document [7].

"If such claims sound too good to be true, keep in mind that they were made by TEX's designer, on a day when TEX happened to be working, so the statements may be biased; but read on anyway."

Figure 9: Donald Knuth displaying the mentality that TeX may sometimes choose not to work. [15]

9 Linux

I've thought I'd continue working on this file in the train, on my laptop. Which uses Ubuntu instead of Windows. A quick

```
sudo apt-get update
sudo apt-get install texlive-full texmaker
```

and a `git clone` later and I was almost set up.

Seems like my latex distribution or environment on Ubuntu behaves differently though. First of all, I had to remove an exclamation mark from the definition of my `mycode` environment for some reason. I hope it's still valid on Windows with MikTeX. And I also had to set up the shell-escape again. It seems like I only have the option of always using it or never, as opposed to the dropdown list with custom options I have on windows. Also, I haven't figured out yet if I'm running the correct tools. On Windows, I'm doing a PdfLaTeX + MakeIndex + BibTex. That does not really exist on my Ubuntu, so I'm just doing PdfLaTeX only. I'll have to learn more about this. But it actually seems like just using PdfLaTeX is enough for generating those citation entries.

Also, the error messages are more, and they are more pedantic/verbose. That could be helpful for debugging, but also annoying. Searching for the first of those error messages in google led me to a valuable explanation about extra closing braces. [13]

Argument of \@newctr has an extra }.<inserted text>\par \repeat

Maybe I should consider all the warnings I'm getting without even running part two first? Maybe also not. They are all just either about design (overfull hbox, fonts) or warn about things that are like I want them to be. E.g. the abbreviation for one of the authors is undefined - but that's totally fine because this is not an abbreviation. It's the authors full name!

10 Night Eight Part Two

The same thing attempted again, after having gotten some help by D. Carlisle [14].

debug for part two: (0)[1], (1)[2], (2)[2], (3)[2],

```
        \newcounter{debugctr}
    \setcounter{debugctr}{-1}
        {\loop
                \stepcounter{debugctr}
                \edef\tmpinput{\the\numexpr \value{debugctr} - 0}
                \def\tmpchar{\expandafter\lucidcharat\tmpinput}
                (\tmpinput)[\tmpchar],
                %\edef\tmpcharr{\tmpchar}
                \show\tmpchar
                \typeout{tmpchar is \tmpchar}
                \ifnum \value{debugctr}<3
                \repeat }
```

Figure 10: Fetching characters in the input string by index.

We already have in Figure 10 the accessor macro for the input string that even works in a loop. Now let's go!

Somehow... I managed to write a command that can be typeout without compiler error if and only if the result is written to the pdf first.

```
  Pre-Loop
Outer Loop at charcounter 1
Inner Loop (layerouncterx = 0) at totalpos 1
tmpchar is the character with meaning macro:->\expandafter
    \lucidcharat \tmpinp
ut
tmpchar has value  \edef 0{0} \relax 1
```

Figure 11: Log File

The log (Figure 11) indicates that my command `\lucidcharat` is not doing what it's supposed to be anyway. It contains the line `\edef\tmpp#1` which should not evaluate `\tmpp` to `0`.

For an explanation of the following magic, see Figure 7 in section 8. It concatenates a string in a loop.

If the magic looks wrong, you might need to run the computation twice. Or

The Image

100100110001100111100110
101001001010010100001001 0
110001000010000111001000 0
101001000010110100001000 0
101001001010010100001001 0
100100110001110111100110 0

Figure 12: The whole image computation happened within this figure. Here it is, formatted as an image.

just look at `day8.xyz`'s content.

Magic: 1:0:0:1:0:0:1:1:0:0:0:0:1:1:0:0:1:1:1:1:1:0:0:1:1:0:0;1:0:1:0:0:1:0:0:1:0:1:0:0:1:0:1:0:0:0:0:1:0:0:1:0;1:1:0:0:0:0:1:0:0

Reminder that the image has 25×6 pixels.

11 Interpretation

Now that we have the binary representation of the image, we still need to interpret it.[18]

Table 1: The image interpreted with zeros black. Hardcoded given the output from part 2.

It would be nice to plot the colon and semicolon-separated string in the same way. But the *pgfplots* package does not feature anything other than newlines as row separators. And those proved to be a real pain to get inside my magic string.

Not possible with pgfplots.

Table 2: The image interpreted with zeros black. Read from the Magic string of the previous section.

12 Todo

Once this is done, I should inform the person who said I should inform them once this is done, to be found at Twitter.

Retry if it still works on windows.

13 Performance

I have tried to measure the time of my code using the `l3benchmark` package [22]. But including this package failed because LaTeX thinks my PC has no clock. PDF Elapsed Time in seconds: 166 seconds

> The value of `\pdfelapsedtime` is the number of seconds elapsed multiplied by 65536. [9]

14 Source

The source code is clingy: It attached itself to this file. [20, 21] Open it in Adobe Reader (on Windows) or Document Viewer (on Ubuntu) and view it under Attachments.
Linking to it does not work unless the file is also a pdf, it seems.

15 Fun Facts

Read [16] for a good warning about using percentage signs at the end of `ifnum` statements. The most intriguing part is displayed in Figure 13.

Another interesting thing about Figure 13 is that it displays very badly unless preceeded by a `\pagebreak`, `\newline`, or `\clearpage` [17]. I'm curious to see if the addition of this paragraph outlining that changes this fact (due to page break positioning). Indeed it turns out that the figure is now displayed correctly without need for a manual page break. Maybe this is because it is now placed "here" instead of on a new page? Nope! Placing it on its own page is okay, this time. I think I won't ever figure this out.

Use `\mbox` around letters that should not be split by line breaks [23]. This is especially useful in the final stage, when proofreading the document's looks.

```
\ifnum1=0%
  \ifnum2=2
    1%
  \else
    2%
  \fi
  a%
\else
  b%
\fi
```

produces a, instead of b, as someone that took a quick glance at the \ifnum1=0 would guess. If, however, you remove the very first %, the output is b.

Figure 13: The dangers of comments.

References

[1] Gonzalo Medina, *Nest StrLen and ifthenelse commands*, https://tex.stackexchange.com/a/15424/102826. Accessed 05.08.2020.

[2] Dimitrios Desyllas, *texify.exe in TeXworks in MSWindows 10: MiKTex encountered internal error when compile with –shell-escape*, https://tex.stackexchange.com/a/437933/1028266. Accessed 05.08.2020.

[3] Uwe Ziegenhagen, *Combining LaTeXwith Python* (09.08.2019), https://tug.org/tug2019/slides/slides-ziegenhagen-python.pdf. Accessed 05.08.2020.

[4] Jason Gross, *Why is everyeof needed to avoid...*, https://tex.stackexchange.com/q/516031/102826. Accessed 07.08.2020.

[5] Burk, *TeX/global*, https://en.wikibooks.org/wiki/TeX/global. Accessed 21.08.2020.

[6] cfr, *Setting a Global Variable in TikZ Loop*, https://tex.stackexchange.com/a/436829/102826. Accessed 21.08.2020.

[7] Steven B. Segletes, *Append strings to a variable and display them at the front of the document*, https://tex.stackexchange.com/a/402174/102826. Accessed 25.08.2020.

[8] Joseph Wright, *What are \immediate & \write18 and how does one use them?*, https://tex.stackexchange.com/a/20446/102826. Accessed 25.08.2020.

[9] Will Robertson, *Are there LaTeX performance profiling tools?*, https://tex.stackexchange.com/a/547/102826. Accessed 25.08.2020.

[10] Stefan v. Bechtolsheim, *A tutorial on \expandafter*, https://www.tug.org/TUGboat/tb09-1/tb20bechtolsheim.pdf. Accessed 28.08.2020.

[11] cgnieder and Sebastian, *Defining and expanding macros In loops*, https://tex.stackexchange.com/a/312177/102826. Accessed 28.08.2020.

[12] David Carlisle, *Defining new commands inside a for loop*, https://tex.stackexchange.com/a/57037/102826. Accessed 28.08.2020.

[13] Unknown, *An extra }?*, http://www.texfaq.org/FAQ-extrabrace. Accessed 28.08.2020.

[14] David Carlisle and LucidBrot, *How to use xstring IfEq inside command?*, https://tex.stackexchange.com/a/561561/102826. Accessed 06.09.2020.

[15] Donald E. Knuth, *The TeXbook*, 1984. http://visualmatheditor.equatheque.net/doc/texbook.pdf.

[16] Phelype Oleynik and Brasil, *How to compare number from \newenvironment argument*, `https://tex.stackexchange.com/a/460014/102826`. Accessed 07.09.2020.

[17] wonderich and egreg, *Comparison between \newpage, \clearpage and \pagebreak, etc.?*, `https://tex.stackexchange.com/a/497776/102826`. Accessed 08.09.2020.

[18] Claudio Fiandrino and Martin Lauridsen, *Drawing a large binary matrix as colored grid in TikZ*, `https://tex.stackexchange.com/a/123728/102826`. Accessed 08.09.2020.

[19] SanseroGames, *Advent of Code - Day 8*, `https://github.com/SanseroGames/adventofcode/tree/master/2019/8`. Accessed 08.09.2020.

[20] vy32, koppor, and diabonas, *Is there some way to embed LaTeX source code in a PDF file?*, `https://tex.stackexchange.com/a/418609/102826`. Accessed 09.09.2020.

[21] David Lichti and Alberto Sartori, *embedall – Embed source files into the generated PDF*, `https://ctan.org/pkg/embedall`. Accessed 09.09.2020.

[22] Latexforti and Oleinik Phelype, *How to measure the compilation time of a document?*, `https://tex.stackexchange.com/a/505840/102826`. Accessed 09.09.2020.

[23] Emanuel Berg and Sveinung, *How can I make a pair of words stick together in line breaks?*, `https://tex.stackexchange.com/a/99691/102826`. Accessed 09.09.2020.