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.

2 The LaTeX Ėẋṗėŕiėńċė

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 1, Figure 2).

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

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

Figure 1: This command does not like to be used on a non-constant string.

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.

```
@@ -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 2: 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 3). Using that allows us to escape to the shell - which is either bash or the windows cmd.exe.

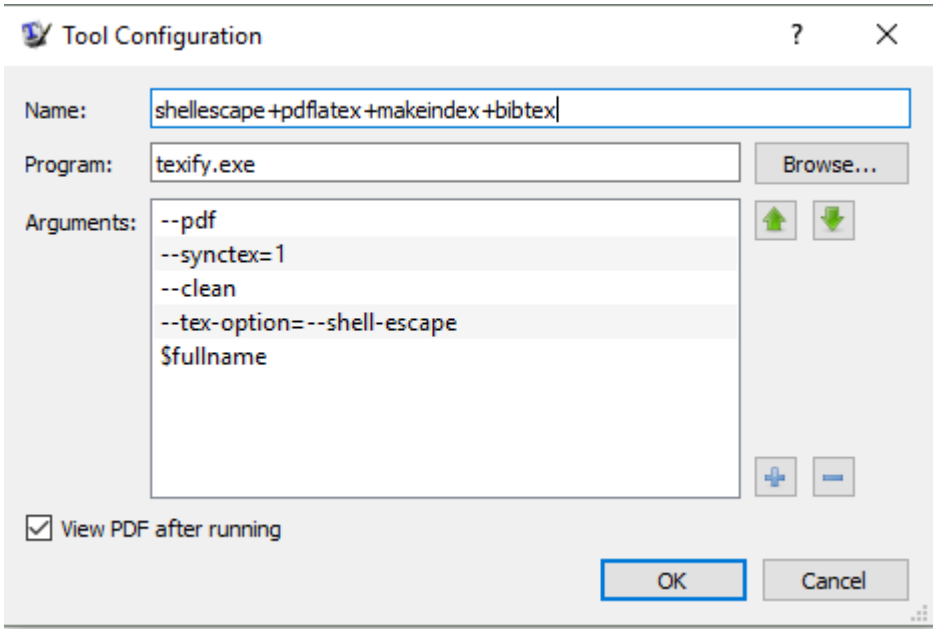

Figure 3: TeXworks settings for shellescape

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. 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);""}
```

3

And as you should be able to see, it works!

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

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

22

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 PyAuCalc.

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

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

5 ...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.

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

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
000001000001220022
hello world 3 3 300
Image Width: 3 Image Height: 2
I want to loop 6 times for the first layer.
The input file contains 18characters.

7 Execution Two

> `\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`. If you care about the following PoC loop, just look at the TeX source.

1, 2, 3,
outerloopcount: 3

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.