

Re: Is reverse reading possible?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-03/2419.html>

From: Stephen Horne (steve_at_ninereeds.fsnet.co.uk)

Date: 03/15/04

Date: Mon, 15 Mar 2004 04:56:46 +0000

On Sun, 14 Mar 2004 11:10:47 -0800 (PST), Anthony Liu
<antonyliu2002@yahoo.com> wrote:

>The files (nearly 4M each)I am gonna read are a
>mixture of Chinese and English, where each Chinese
>character has 2 bytes and each English (ASCII) has 1
>byte, although the majority of the texts is Chinese.

Four MB isn't much when typical machines have from 256MB to 1GB of RAM. Though maybe this isn't the case in China?

Still, if you are concerned, it is possible to read the file line-by-line backwards. But it will be very difficult to impossible to do it straight off when you have non-uniform character sizes as character encodings aren't designed to allow this – it is not generally possible to find the start-byte of each character scanning backwards from the end-byte, only to find the end-byte scanning forward from the start-byte.

Solution – do an initial pass reading the file, and keep a list of the start positions of each line. The following code is untested and probably wrong, but should show the general idea of how to do this without knowing the encoding method...

```
f = file("r")
starts = []

while 1 :
    starts.append(f.tell ())
    t = f.readline ()
    if t == " " : break
```

At the end of the loop, starts contains one position for the start of each line plus one extra for the end of the file. The for line no. i, the line can be read as follows...

```
f.seek (starts [i])
t = f.readline ()
```

comp.lang.python: Re: Is reverse reading possible?

And given that the purpose is reverse reading, the following generator might be useful...

```
def backward_read (p_File) :
    l_Starts = []
    p_File.seek (0) # probably paranoia

    # First pass, recording line start positions
    # (no end-of-file pos this time)
    while 1 :
        l_Pos = starts.append (p_File.tell ())
        if p_File.readline () == " : break
        l_Starts.append (l_Pos)

    l_Starts.reverse ()

    # Second pass, yielding the lines of text
    for i in l_Starts :
        p_File.seek (i)
        yield p_File.readline ()
```

If you really want to avoid the initial read too, the general approach would have to be...

1. Derive a grammar (basically a regular expression) for a single character using the character encoding on your machine, such that a match would recognise the sequence of bytes for any single character.

The general form might be something like...

```
CHAR : 0x41 # ASCII character 'A'
CHAR : 0x42 # ASCII character 'B'
...
CHAR : 0x80 0x01 # A wide character
CHAR : 0x80 0x02 # Another wide character
...
```

2. From this, derive the grammar for a whole text file – without using common regular expression shortcuts such as '*'. The general form will be something like...

```
FILE :
FILE : CHAR FILE
```

3. Reverse every rule in the grammar.
4. Generate a parser from the grammar.
5. Run the parser, using line end characters to figure out the 'starts' of new lines.

Re: Is reverse reading possible?

comp.lang.python: Re: Is reverse reading possible?

Step 1 is a bit of a killer already – it's a pain having to find out the details of an encoding when you're used to things just working ;-)

Step 4 is a much more serious killer. Traditional LL and LR techniques (let alone regular expressions) simply won't cope as the grammar will be full of shift/reduce conflicts. This is the parser-generators way of saying that it can't determine the end token of a rule (in this case CHAR) by scanning from the start token (remember the grammar is reversed, so the start token of a CHAR is the end byte of a character).

It is possible to handle this though. Tomita parsers are based on LR techniques, but resolves conflicts at run time – they use 'stack duplication' to keep track of different possible interpretations arising due to LR conflicts.

Step 5 is then at least attemptable. As long as the degree of ambiguity (ie the number of stack duplications) stays within reasonable bounds, it should be possible to read the file backwards. This will use a lot of memory, though, even with quite modest files. And it is always possible that in a worst-case runs none of the ambiguities won't get resolved until the whole file is read (and maybe not even then, if the file isn't in the expected encoding).

This isn't intended as a practical how-to-do-it, obviously, but rather as an explanation of why you don't want to have to do it!

Personally, I'd just read the text lines into a list and iterate the list backwards.

--

Steve Horne
steve at ninereeds dot fsnet dot co dot uk