Last byte in Huffman compression

Asked 11 years, 10 months ago Modified 3 years, 1 month ago

Viewed 4k times

8

I am wondering about what is the best way to handle the last byte in Huffman Copression. I have some nice code in C++, that can compress text files very well, but currently I must write to my coded file also number of coded chars (well, it equal to input file size), because of no idea how to handle last byte better.

For example, last char to compress is 'a', which code is 011 and I am just starting new byte to write, so the last byte will look like: 011 + some 5 bits of trash, I am making them zeros for example at the end. And when I am encoding this coded file, it may happen that code 00000 (or with less zeros) is code for some char, so I will have some trash char at the end of my encoded file.

As I wrote in first paragraph, I am avoiding this by saving numbers of chars of input file in coded file, and while encoding, I am reading the coded file to reach that number (not to EndOfFile, to don't get to those example 5 zeros). It's not really efficient, size of coded file is increased for long number.

How can I handle this in better way?

algorithm compression ascii eof huffman-code

Share

Improve this question

Follow

edited Oct 27, 2021 at 22:37

Silicomancer

9,096 ● 12 ● 74 ● 140

asked Feb 2, 2013 at 23:31

Horuss

300 ● 3 ● 19

2 Answers

Sorted by: Highest score (default)

8

Your approach (write the number of encoded bytes the to the file) is a perfectly reasonable approach. If you want to try a different avenue, you could consider inventing a new "pseudo-EOF" character that marks the end of the input (I'll denote it as □). Whenever you want to compress a string *s*, you instead compress the string *s*□. This means that when you build up your encoding tree, you would include one copy of the □ character so that you have a unique encoding for □. Then, when you write out the string to the file, you would write out the bits characters of the string as normal, then write out the bit pattern for □. If there are leftover bits, you can just leave them set arbitrarily.

The advantage to this approach is that as you decode the file, if at any point you find the □ character, you can immediately stop decoding bits because you know that

you have hit the end of the file. This does not require you to store the number of bytes that were written out anywhere - the encoding implicitly marks its own endpoint.

The disadvantage to this setup is that it might increase the length of the bit patterns used by certain characters, since you will need to assign a bit pattern to □ in addition to all the other characters.

I teach an introductory programming course and we use Huffman encoding as one of our assignments. We have students use the above approach, since it's a bit easier than having to write out the number of bits or bytes before the file contents. For more details, you could take a look at this handout or these lecture slides from the course.

Hope this helps!

Share Improve this answer

Follow

answered Feb 2, 2013 at 23:39

templatetypedef

372k ● 111 ● 941 ● 1.1k

There really isn't a disadvantage, since you can prove that this won't take more bits than encoding the length.
– Mark Adler Feb 3, 2013 at 2:37

1 @MarkAdler- Do you have a reference for that? I have often wondered this, but I've never seen a formal proof.
– templatetypedef Feb 3, 2013 at 7:37

I don't remember off-hand where I saw that. I'll look for it.
– Mark Adler Feb 3, 2013 at 18:46

Thanks for answer. But idea with "sacrificing" one char for pseudo-EOF means, that when it will appear in text it ruin everything. I know that many ASCII chars are not used at all, but I would really love to have sure, that I can code/compress every ASCII char, what then? And about my approach: when text file will be bigger than sizeof long (well, it's possible maybe ;-D) that will crash – Horuss Feb 3, 2013 at 21:32

@Kuba- You wouldn't do this by sacrificing a character for pseudo-EOF. Instead, you would pick a non-char value (say, INT_MAX) and pretend that it is a character, since this value can't appear in any string. I think the easiest way to do this is to do all of your encoding on values of type `int` rather than type `char`, since you can always encode a `char` as an `int` and can then encode extra values like the pseudo-EOF as an `int` as well. Does that make sense?
– templatetypedef Feb 3, 2013 at 21:34

3

I know this is an old question, but still, there's an alternate, so it might help someone.

When you're writing your compressed file to output, you probably have some integer keeping track of where you are in the current byte (for bit shifting).

```
char c, p;
p = '\0';
int curr = 7;
while (infile.get(c))
{
    std::string trav = GetTraversal(c);
    for (int i = 0; i < trav.size(); i++)
    {
        if (trav[i] == '1')
            p += (1 << curr);
```

```
            if (--curr < 0)
            {
                outfile.put(p);
                p = '\0';
                curr = 7;
            }
        }
}
if (curr < 7)
    outfile.put(p);
```

At the end of this block, `(curr+1)%8` equals the number of trash bits in the last data byte. You can then store it at the end as a single extra byte, and just keep it in mind when you're decompressing.

Share Improve this answer

Follow

answered Dec 16, 2014 at 20:38

Regan Carver
31 • 1