need help on how to encode words using huffman code

Asked 16 years ago Modified 9 years, 5 months ago Viewed 5k times

3

how do you encode words using the huffman code such as NEED

algorithm encoding compression huffman-code

Share

Improve this question

Follow

edited Dec 7, 2008 at 6:36

Schwern

164k ● 27 ● 218 ● 362

asked Dec 7, 2008 at 6:23

denise

4 Answers

Sorted by: Highest score (default)

5

Huffman encoding basically uses variable-length bit strings to represent tokens (generally characters with a couple of exceptions). The more common a token is, the shorter it's bit-length is and this is (usually) dynamic as the stream is processed.

There are usually two special tokens, ESCAPE and END-STREAM.

Encoding maintains a dictionary which is basically a lookup of the bit sequences to get a token. Initially it contains only the two special tokens.

The initial bit sequences for ESCAPE and END_STREAM could be 0 and 1 (which is which doesn't really matter at the start). When the encoder receives a character not in the dictionary, it outputs ESCAPE and the full length token, then it adds it and assigns new bit sequences, based on frequency of all the tokens.

So your 'N' may result in the output stream:

```
0 xxxxxxxx
| +- token code for N
+--- ESCAPE
```

and the new dictionary:

```
ESCAPE:00
END-STREAM:01
N:1
```

Then your 'E' may result in the output stream:

```
0 xxxxxxxx 0 yyyyyyyy
             +- token code for E
```

and the new dictionary:

```
ESCAPE:00
END-STREAM:01
N:10
E:11
```

Your next E will not result in a ESCAPE output since it's already in the dictionary so you just output its code (11). It will change the dictionary since E now has a higher count. This won't matter in our situation since all characters are two binary digits but, in a more complicated example, the bit length of the 'E' token would shorten.

When the D arrives, the output stream becomes:

```
0 xxxxxxxx 0 yyyyyyyy 11 0 zzzzzzzz
                       |    +- token code for D
                       +------ second E
```

and the new dictionary:

```
ESCAPE:00
END-STREAM:011
N:010
E:11
D:10
```

So you can see that, as more characters come in, the bit length of the common ones reduce and that of the uncommon one increase, giving you your compression. N (or D) in this case gets a 3-digit code, while E sticks with a 2-digit code because there's more of them.

The beauty is that you don't need to store the dictionary with the file since the ESCAPE sections build it dynamically for de-compression as well.

In addition, because there's NEVER an END-STREAM token until the end, it's bit length keeps getting bigger. Similar for ESCAPE, while there's still lots of new character coming in, its bit length stays short but, when no new characters are arriving, it suffers the same fate as END-STREAM.

The best case for an (8-bit ASCII) input stream is a file containing nothing but millions of the same character. This costs 9 bits for the first character, then 1 bit for each additional character then 2 bits for the end of stream. That fast approaches a 1-for-8 compression ratio (97.5%).

The worst case is exactly one of each character which costs 9 bits per character plus the end of stream - this actually expands the stream by 12.5%.

edited Dec 7, 2008 at 7:19

answered Dec 7, 2008 at 7:06

paxdiablo
880k • 241 • 1.6k • 2k

This is an example of dynamic huffman coding. Which is not the most well-known variant. – Joris Schellekens Aug 31, 2017 at 11:22

1

Take a look at Huffman Coding with F#, a blog post which presents a Huffman coder/decoder written in F#. It's short and clear.

Share Improve this answer

Follow

answered Dec 7, 2008 at 6:56

namin

38.1k ● 9 ● 61 ● 74

1

I think you mean Huffman Coding. It's an algorithm to compress data without loss. Put simply, you replace the longest and most repetitious contiguous bits of data to the smallest possible representation (which is how most compression works). For example, an HTML page might assign the very common `<DIV` to the binary number `01` reducing the 32 bits of every `<DIV` to just 2 bits.

That's the basic idea. The other trick is how to pick the numbers so you don't need to use a fixed size or a separator. This is done using a Huffman Tree. Read the Wikipedia article for more.

Share Improve this answer

Follow

edited Dec 19, 2009 at 22:33

answered Dec 7, 2008 at 6:33

Schwern

164k ● 27 ● 218 ● 362

It will convert the <DIV token to a smaller set only if you set the alphabit to 4 character long tokens – monksy Dec 19, 2009 at 21:06

0

Take a look at my Huffman in C# project:
https://github.com/kad0xf/HuffmanSharp

Share Improve this answer

Follow

answered Jul 22, 2015 at 19:38

K.A.D.

3,718 ● 3 ● 35 ● 35