
Subject: Substring search algorithm

Posted by [chickenk](#) on Sun, 26 Feb 2012 09:55:46 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hello,

I have a very bad knowledge of algorithms in general (I'm not a math guy...) but I found this article and I thought it could be interesting sharing it, results seem to be good (I know, I know, benchmarks have no value when they are made by the algorithm creator... anyway...) :
http://volnitsky.com/project/str_search/index.html

I don't know if there are any specific substring searching algorithm in U++, but that may be interesting to compare and see who can perform better. The reference implementation is in C++, and there are fallbacks to `std::search()` in some places, which could be replaced by fallbacks to U++ own search implementation. I am really curious about the results from such a combination...

Hope there's something interesting to take from this algo.

Cheers

Lionel

Subject: Re: Substring search algorithm

Posted by [mirek](#) on Sun, 26 Feb 2012 12:14:06 GMT

[View Forum Message](#) <> [Reply to Message](#)

Quite interesting idea. Have to admit took me hour+ to understand the algorithm; perhaps part of initial misunderstanding was that if I understand it well, the substring length is limited to 64KB (or `H_size`) (code suggest something like `offset_t` maximum).

Also, even the the original code suggests that it only works fine for searched strings > 20kb, because initialization costs are pretty high - that is a bit high for `Upp::String`.

OTOH, it might be interesting to try this with `VectorMap` instead of that hash thing in the code. Very likely, it would be quite faster and init costs would be much smaller.

Mirek

Subject: Re: Substring search algorithm

Posted by [mr_ped](#) on Mon, 27 Feb 2012 09:55:12 GMT

[View Forum Message](#) <> [Reply to Message](#)

"Boyer-Moore-Horspool" and family of those are suitable for U++ implementation (I did it years ago in Pascal at uni), but I wonder what's the real speed up benefit in real world app, because nowadays the CPU + L1 cache operates light years faster than L2+ cache/RAM, so while BMH will save you some compares, it has to fetch the full text anyway, and I would expect a properly

implemented naive byte compare will easily do it's work inside the "time window" of memory fetch of next data.

But I may try to write BMH for U++ and toy around with that for a while to see some real numbers.
