Fast String Hashing Algorithm with low collision rates with 32 bit integer [closed]

Asked 16 years, 3 months ago Modified 9 years, 8 months ago

Viewed 107k times

70

Closed. This question is seeking recommendations for software libraries, tutorials, tools, books, or other off-site resources. It does not meet Stack Overflow guidelines. It is not currently accepting answers.

We don't allow questions seeking recommendations for software libraries, tutorials, tools, books, or other off-site resources. You can edit the question so it can be answered with facts and citations.

Closed 9 years ago.

The community reviewed whether to reopen this question 2 years ago and left it closed:

> Original close reason(s) were not resolved

Improve this question

I have lots of unrelated named things that I'd like to do quick searches against. An "aardvark" is always an "aardvark" everywhere, so hashing the string and reusing

the integer would work well to speed up comparisons. The entire set of names is unknown (and changes over time). What is a fast string hashing algorithm that will generate small (32 or 16) bit values and have a low collision rate?

I'd like to see an optimized implementation specific to C/C++.

c++ algorithm string hash

Share

Improve this question

Follow

edited Sep 9, 2012 at 16:20

Coding Mash

3,346 ● 5 ● 25 ● 45

asked Sep 22, 2008 at 10:03

Jason Citron

885 ● 2 ● 9 ● 11

please add keywords: hash algorithm unique low-collision – slashmais Sep 24, 2008 at 10:31

25 The following page has several implementations of general purpose hash functions which are "performant" and have low "collision rates": partow.net/programming/hashfunctions/index.html – Matthieu N. Oct 31, 2010 at 23:08

13 Answers

Sorted by: Highest score (default)

One of the FNV variants should meet your requirements. They're fast, and produce fairly evenly distributed outputs.

30

Share Improve this answer

Follow

answered Sep 22, 2008 at 10:08

Nick Johnson

101k ● 17 ● 130 ● 198

If you're going to use FNV, stick to FNV-1a, since it has acceptable results on the avalanche test (see home.comcast.net/~bretm/hash/6.html). Or just use MurmurHash2, which is better in both speed and distribution (murmurhash.googlepages.com). – Steven Sudit Jul 10, 2009 at 3:38

7 @Steven: MurmurHash hash has only been analzyed by its author. I've used it in a few different scenarios and the newer version of FNV seems to do a better job. – Matthieu N. Jan 25, 2011 at 22:03

@sonicoder: While I'm not going to oversell MurmurHash, plain FNV is downright terrible and FNV-1a is only passable. As it happens, MurmurHash has been extensively analyzed and found useful. It's still not a cryptographic hash and there are going to be collisions no matter what, but it's still a huge improvement over any type of FNV. – Steven Sudit Jan 26, 2011 at 20:50

8 @Steven Sudit: As I said, it was analyzed "only" by its author and no one else. hence the results of the "analysis" aren't really objective. – Matthieu N. Jan 30, 2011 at 12:07

6 @sonicoder: I'll speak more bluntly: no, you're mistaken. It was analyzed by a number of third parties, including academic ones. Visit Wikipedia for links. What's more important is that, not only did it do well in general, but the specific flaws that were found were addressed through the creation of MurmurHash3. – Steven Sudit Feb 3, 2011 at 7:25

20

There's also a nice article at eternallyconfuzzled.com.

Jenkins' One-at-a-Time hash for strings should look something like this:

```
#include <stdint.h>

uint32_t hash_string(const char * s)
{
    uint32_t hash = 0;

    for(; *s; ++s)
    {
        hash += *s;
        hash += (hash << 10);
        hash ^= (hash >> 6);
    }

    hash += (hash << 3);
    hash ^= (hash >> 11);
    hash += (hash << 15);

    return hash;
}
```

Share Improve this answer

Follow

answered Dec 16, 2008 at 22:25

Christoph

169k ● 36 ● 185 ● 241

17

For a fixed string-set use gperf.

If your string-set changes you have to pick one hash function. That topic has been discussed before:

[What's the best hashing algorithm to use on a stl string when using hash_map?]

Share Improve this answer

Follow

edited May 23, 2017 at 12:02

Community Bot

1 • 1

answered Sep 22, 2008 at 10:13

Nils Pipenbrinck

86.2k • 33 • 155 • 223

A perfect hash is a very elegant solution, when available.
– Steven Sudit Jan 27, 2011 at 18:23

9

Another solution that could be even better depending on your use-case is **interned strings**. This is how symbols work e.g. in Lisp.

An interned string is a string object whose value is the address of the actual string bytes. So you create an interned string object by checking in a global table: if the string is in there, you initialize the interned string to the address of that string. If not, you insert it, and then initialize your interned string.

This means that two interned strings built from the same string will have the same value, which is an address. So if N is the number of interned strings in your system, the characteristics are:

- Slow construction (needs lookup and possibly memory allocation)
- Requires global data and synchronization in the case of concurrent threads
- Compare is O(1), because you're comparing addresses, not actual string bytes (this means sorting works well, but it won't be an alphabetic sort).

Cheers,

Carl

Share Improve this answer Follow

edited Sep 22, 2008 at 13:08

answered Sep 22, 2008 at 11:02

Carl Seleborg

13.3k ● 11 ● 59 ● 70

6

It is never late for a good subject and I am sure people would be interested on my findings.

I needed a hash function and after reading this post and doing a bit of research on the links given here, I came up

with this variation of Daniel J Bernstein's algorithm, which I used to do an interesting test:

```
unsigned long djb_hashl(const char *clave)
{
    unsigned long c,i,h;

    for(i=h=0;clave[i];i++)
    {
        c = toupper(clave[i]);
        h = ((h << 5) + h) ^ c;
    }
    return h;
}
```

This variation hashes strings ignoring the case, which suits my need of hashing users login credentials. 'clave' is 'key' in Spanish. I am sorry for the spanish but its my mother tongue and the program is written on it.

Well, I wrote a program that will generate usernames from 'test_aaaa' to 'test_zzzz', and -to make the strings longer- I added to them a random domain in this list: 'cloud-nueve.com', 'yahoo.com', 'gmail.com' and 'hotmail.com'. Therefore each of them would look like:

```
test_aaaa@cloud-nueve.com, test_aaab@yahoo.com,
test_aaac@gmail.com, test_aaad@hotmail.com and so
```

```
on.
```

Here is the output of the test -'Colision entre XXX y XXX' means 'Collision of XXX and XXX'. 'palabras' means 'words' and 'Total' is the same in both languages-.

```
    Buscando Colisiones...
    Colision entre 'test_phiz@hotmail.com' y
'test_juxg@cloud-nueve.com' (1DB903B7)
    Colision entre 'test_rfhh@hotmail.com' y
'test_fpgo@yahoo.com' (2F5BC088)
    Colision entre 'test_wxuj@hotmail.com' y
'test_pugy@cloud-nueve.com' (51FD09CC)
    Colision entre 'test_sctb@gmail.com' y
'test_iohw@cloud-nueve.com' (52F5480E)
    Colision entre 'test_wpgu@cloud-nueve.com' y
'test_seik@yahoo.com' (74FF72E2)
    Colision entre 'test_rfll@hotmail.com' y
'test_btgo@yahoo.com' (7FD70008)
    Colision entre 'test_wcho@cloud-nueve.com' y
'test_scfz@gmail.com' (9BD351C4)
    Colision entre 'test_swky@cloud-nueve.com' y
'test_fqpn@gmail.com' (A86953E1)
    Colision entre 'test_rftd@hotmail.com' y
'test_jlgo@yahoo.com' (BA6B0718)
    Colision entre 'test_rfpp@hotmail.com' y
'test_nxgo@yahoo.com' (D0523F88)
    Colision entre 'test_zlgo@yahoo.com' y
'test_rfdd@hotmail.com' (DEE08108)
    Total de Colisiones: 11
    Total de Palabras  : 456976
```

That is not bad, 11 collisions out of 456,976 (off course using the full 32 bit as table lenght).

Running the program using 5 chars, that is from 'test_aaaaa' to 'test_zzzzz', actually runs out of memory building the table. Below is the output. 'No hay memoria para insertar XXXX (insertadas XXX)' means 'There is not memory left to insert XXX (XXX inserted)'. Basically malloc() failed at that point.

```
    No hay memoria para insertar 'test_epjcv'
(insertadas 2097701).

    Buscando Colisiones...

    ...451 'colision' strings...

    Total de Colisiones: 451
    Total de Palabras  : 2097701
```

Which means just 451 collisions on 2,097,701 strings. Note that in none of the occasions, there were more than 2 collisions per code. Which I confirm it is a great hash for me, as what I need is to convert the login ID to a 40 bit unique id for indexing. So I use this to convert the login credentials to a 32 bit hash and use the extra 8 bits to handle up to 255 collisions per code, which lookign at the test results would be almost impossible to generate.

Hope this is useful to someone.

EDIT:

Like the test box is AIX, I run it using LDR_CNTRL=MAXDATA=0x20000000 to give it more

memory and it run longer, the results are here:

Buscando Colisiones... Total de Colisiones: 2908 Total de Palabras : 5366384

That is 2908 after 5,366,384 tries!!

VERY IMPORTANT: Compiling the program with -maix64 (so unsigned long is 64 bits), the number of collisions is 0 for all cases!!!

Share Improve this answer Follow

edited Sep 29, 2013 at 12:27

answered Sep 26, 2013 at 12:22

Antonio Morales

61 ● 1 ● 2

There seems to be a bug in your code: your `h` needs to be initialized to zero, or you might start with uninitialized memory and get non-reproducible changing hashes which would be kind of bad. – E. K. Aug 5, 2022 at 10:58

3

The Hsieh hash function is pretty good, and has some benchmarks/comparisons, as a general hash function in C. Depending on what you want (it's not completely obvious) you might want to consider something like cdb instead.

Share Improve this answer Follow

answered Sep 24, 2008 at 4:13

James Antill
2,855 • 18 • 16

3

Why don't you just use Boost libraries? Their hashing function is simple to use and most of the stuff in Boost will soon be part of the C++ standard. Some of it already is.

Boost hash is as easy as

```
#include <boost/functional/hash.hpp>

int main()
{
    boost::hash<std::string> string_hash;

    std::size_t h = string_hash("Hash me");
}
```

You can find boost at boost.org

Share Improve this answer

Follow

answered Dec 16, 2008 at 21:11

Bernard Igiri
1,290 • 2 • 18 • 34

5 Both STL and boost tr1 has extremely weak hash function for strings. – obecalp Feb 16, 2009 at 19:19

3

Bob Jenkins has many hash functions available, all of which are fast and have low collision rates.

Share Improve this answer

answered Dec 16, 2008 at 21:30

Follow

user7116
64k ● 17 ● 146 ● 174

2 The hashes are quite solid, and technically interesting, but not necessarily fast. Consider that One-at-a-Time hash processes input byte by byte, where other hashes take 4 or even 8 bytes at a time. The speed differnece is substantial! – Steven Sudit Jul 23, 2009 at 15:23

Bob's hashes are very fast: azillionmonkeys.com/qed/hash.html – user7116 Jul 23, 2009 at 20:20

Have a look at GNU gperf.

2

Share Improve this answer

Follow

answered Sep 22, 2008 at 10:06

Rob Wells
37k ● 13 ● 84 ● 147

Yay for perfect hash generators! – C. K. Young Sep 22, 2008 at 10:08

5 Perfect hashing is NOT appropriate for this application, since the set of names is unknown and changes. Therefore, gperf won't work for this. – TimB Sep 24, 2008 at 4:43

2

You can see what .NET uses on the String.GetHashCode() method using Reflector.

I would hazard a guess that Microsoft spent considerable time optimising this. They have printed in all the MSDN documentation too that it is subject to change all the time. So clearly it is on their "performance tweaking radar" ;-)

Would be pretty trivial to port to C++ too I would have thought.

Share Improve this answer

Follow

answered Dec 16, 2008 at 21:34

nbevans

7,949 ● 3 ● 29 ● 33

2

There is some good discussion in this previous question

And a nice overview of how to pick hash functions, as well as statistics about the distribution of several common ones here

Share Improve this answer

Follow

edited May 23, 2017 at 12:25

Community Bot

1 ● 1

answered Dec 9, 2008 at 21:29

AShelly

35.5k ● 16 ● 95 ● 157

-1

Described here is a simple way of implementing it yourself: http://www.devcodenote.com/2015/04/collision-free-string-hashing.html

A snippet from the post:

if say we have a character set of capital English letters, then the length of the character set is 26 where A could be represented by the number 0, B by the number 1, C by the number 2 and so on till Z by the number 25. Now, whenever we want to map a string of this character set to a unique number , we perform the same conversion as we did in case of the binary format

Share Improve this answer

Follow

answered Apr 17, 2015 at 3:33

Abhishek Jain

4,518 ● 8 ● 35 ● 52

How does that (given a hypertext browser that does display that links contents) map to `(32 or 16) bit values`, given character sets from, say, 24 to 1.111.998 code points? Base conversion is not a useful hash function. – greybeard Apr 17, 2015 at 4:01

-3

CRC-32. There is about a trillion links on google for it.

Share Improve this answer

Follow

answered Sep 22, 2008 at 10:06

1800 INFORMATION

135k ● 30 ● 163 ● 242

8 CRCs are designed for error detection and correction. Their distribution characteristics are typically not very good. – Nick Johnson Sep 22, 2008 at 10:09

1 Arachnid has obviously never tried CRC32 as hashes. They work well. – Nils Pipenbrinck Sep 22, 2008 at 10:11

10 "CRC32 was never intended for hash table use. There is really no good reason to use it for this purpose." cf. home.comcast.net/~bretm/hash/8.html – obecalp Feb 16, 2009 at 21:24