

Re: SGI hash_map

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.cpp/2005-03/3199.html

From: Ivan Vecerina (INVALID_use_webform_instead_at_vecerina.com)

Date: 03/22/05

Date: Tue, 22 Mar 2005 07:28:57 +0100

"Christian Meier" <chris@gmx.ch> wrote in message
news:d1mo02\$3t7\$1@news.hispeed.ch...

> "Ivan Vecerina" <NOT_VALID_please_use_contact_webform@vecerina.com>
> schrieb

>> *It is essential that the function returns a relatively uniformly
>> distributed*

>> *random value. A minimalistic way to achieve this is to multiply an input
>> value by some large prime number, better is to use one of the many well-
>> studied hash functions you'll find on the web.*

>> *For an intro, see for example:*

>> <http://www.concentric.net/~Ttwang/tech/inthash.htm>

...

> *Because there is no hash<uint64_t> function, I wrote my own. As the
> hash<int> value for an int of the value 5435438 is 5435438 and for 123456
> is*

> *123456, I just return the uint64_t value:*

> *return static_cast<size_t>(ujHash);*

>

> *My values do not have to be multiplied by a prime number because I get
> different values with little difference (not 1000000, 2000000 and
> 3000000).*

> *And before inserting into the map, each hash value is calculated with:*

> *hash_val %= bucket_count();*

> *And the number of buckets is always a prime number in the SGI
> implementation.*

Depending on how the values are distributed, you may or may not have
a uniform distribution. If you care to check, you could probably write
a program to count the number of buckets that contain multiple items.

> *In the meantime I looked up the source code of the SGI library. And there
> is*

> *the function insert_unique which is called by the hash map function*

> *::insert():*

>

> *pair<iterator, bool> insert_unique(const value_type& __obj)*

> *{*

> *resize(_M_num_elements + 1);*

comp.lang.c++. Re: SGI hash_map

> *return insert_unique_noresize(__obj);*
> }
>
> *This means: Each time an element is inserted into the hash map, it will be*
> *checked for resizing depending on _M_num_elements. _M_num_elements is the*
> *number of ALL elements in the map. If I have all elements in the same*
> *bucket, the map will be resized after reaching the number of buckets*
> *although*
> *they are all in the same bucket...*
> *I don't know why this is written like this.*

This ensures that item search is always as efficient as possible (if this doesn't matter to a program, then std::map may be a better candidate). Like for the resizing of std::vector, the number of 'rehashings' in hashmap is amortized constant relative to the number of contained item. So this is normally not a problem. (NB: there are some sophisticated hash table algorithms to dynamically 'redistribute' items, but they only make sense in specific implementations).

> *This implementation is written for a hash codes which are unique.*
Yes, this is what they are supposed to be !

> *Well, this is no problem for numeric data*
> *types of smaller size than std::size_t. But this implementation of the*
> *hash*
> *map would be quite ugly if I wanted to insert large strings for*
> *example.....*

Again, not really a problem because the number of hash code computations is amortized constant (~2) per item inserted.

> *Well, I could answer my question by myself. But I do not really understand*
> *why the SGI people want to have as many buckets as elements in every*
> *case....*

In non-pathological cases (proper hashing) this is what allows hash_map to perform queries at optimal speed – this is the only benefit of hash_map. Searching (linearly) through multiple items in the same bucket can be quite expensive.

Cheers,
Ivan

--

http://ivan.vecarina.com/contact/?subject=NG_POST <- email contact form