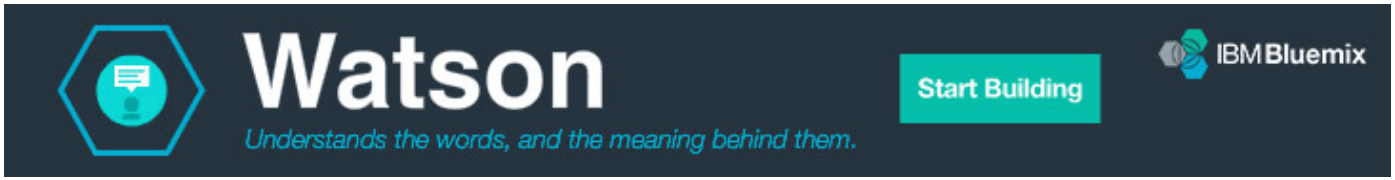
sign up log in tour help stack overflow careers

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free. Take the 2-minute tour ×

hash_map and map which is faster? less than 10000 items



vs2005 support ::stdext::hash_map ::std::map.

however it seems ::stdext::hash_map's insert and remove OP is slower then ::std::map in my test. (less then 10000 items)

Interesting....

Can anyone offered a comparision article about them?

c++ stl map hashmap

edited Jul 14 '09 at 14:21
JohnFx
26k 14 73 129

asked Jul 14 '09 at 9:57
user25749
1,525 11 40 71

VS2005's hash_map is horribly inefficient with std::string's... what are you hashing? – PiNoYBoY82 Jul 14 '09 at 21:36

I put shared_ptr<XXX> in it – user25749 Jul 15 '09 at 6:42

6 Answers

It is not just about insertion and removal. You must consider that memory is allocated differently

in a hash_map vs map and you every time have to calculate the hash of the value being searched.

I think this Dr.Dobbs article will answer your question best:

C++ STL Hash Containers and Performance

edited Nov 22 '13 at 16:18

chrisaycock
14.9k 2 27 61

answered Jul 14 '09 at 10:24

ovanes
1,944 15 29

That is exactly what I needed, many thanks! :) – user25749 Jul 14 '09 at 10:28

Normally you look to the complexities of the various operations, and that's a good guide: amortized O(1) insert, O(1) lookup, delete for a hashmap as against O(log N) insert, lookup, delete for a tree-based map.

However, there are certain situations where the complexities are misleading because the constant terms involved are extreme. For example, suppose that your 10k items are keyed off strings. Suppose further that those strings are each 100k characters long. Suppose that different strings typically differ near the beginning of the string (for example if they're essentially random, pairs will differ in the first byte with probability 255/256).

Then to do a lookup the hashmap has to hash a 100k string. This is O(1) in the size of the collection, but might take quite a long time since it's probably O(M) in the length of the string. A balanced tree has to do log N <= 14 comparisons, but each one only needs to look at a few bytes. This might not take very long at all.

In terms of memory access, with a 64 byte cache line size, the hashmap loads over 1500 sequential lines, and does 100k byte operations, whereas the tree loads 15 random lines (actually probably 30 due to the indirection through the string) and does 14 * (some small number) byte operations. You can see that the former might well be slower than the latter. Or it might be faster: how good are your architecture's FSB bandwidth, stall time, and speculative read caching?

If the lookup finds a match, then of course in addition to this both structures need to perform a single full-length string comparison. Also the hashmap might do additional failed comparisons if there happens to be a collision in the bucket.

So assuming that failed comparisons are so fast as to be negligible, while successful comparisons and hashing ops are slow, the tree might be roughly 1.5-2 times as fast as the hash. If those assumptions don't hold, then it won't be.

An extreme example, of course, but it's pretty easy to see that on your data, a particular O(log N) operation might be considerably faster than a particular O(1) operation. You are of course right to want to test, but if your test data is not representative of the real world, then your test results may not be representative either. Comparisons of data structures based on complexity refer to behaviour in the limit as N approaches infinity. But N doesn't approach infinity. It's 10000.

edited Jun 19 '12 at 9:05

answered Jul 14 '09 at 11:12

Steve Jessop
184k 17 252 503

It depends upon your usage and your hash collisions. One is a binary tree and the other is a hashtable.

Ideally the hash map will have O(1) insertion and lookup, and the map O(ln n), but it presumes non-clashing hashes.

answered Jul 14 '09 at 10:02

polyglot
1,223 1 12 22

---

hash_map uses a hash table, something that offers almost constant time O(1) operations assuming a good hash function.

map uses a BST, it offers O(lg(n)) operations, for 10000 elements that's 13 which is very acceptable

I'd say stay with map, it's safer.

answered Jul 14 '09 at 14:26


Diaa Sami
2,094 14 18

Thanks ! I decided to use map instead of hash_map – user25749 Jul 15 '09 at 6:44

Hash tables are supposed to be faster than binary trees (i.e. std::map) for lookup. Nobody has ever suggested that they are faster for insert and delete.

answered Jul 14 '09 at 10:06
anon

I think hashmap is designed O(1) time complex when reading ,insert. maybe delete is another thing – user25749 Jul 14 '09 at 10:10

If the hashmap resolves collisions by using a linked list; the insert, search and delete performance should all be (roughly) the same. – Jasper Bekkers Jul 14 '09 at 10:14

2 You've got this wrong - deletion and insertion times are a function of the time it takes to look up the insertion point, O(1) and O(log n) respectively. It *is* claimed that deletion and insertion are of the same order. – polyglot Jul 14 '09 at 14:59

A hash map will create a hash of the string/key for indexing. Though while proving the complexity it is mentioned as O(1), hash_map does collision detection for every insert as a hash of a string can produce the same index as the hash of another string. A hash map hence has complexity for managing these collisions & you konw these collisions are based on the input data.

However, if you are going to perform lot of look-ups on the structure, opt for hash_map.

answered Jul 14 '09 at 10:09

Narendra N
774 1 5 10