map vs. hash_map in C++

I have a question with `hash_map` and `map` in C++. I understand that `map` is in STL, but `hash_map` is not a standard. What's the difference between the two?

c++ map hashmap

edited Sep 29 '13 at 14:21
Lance Roberts
12.7k 21 73 111

asked Feb 3 '10 at 2:12
skydoor
6,770 17 86 146

4 Answers

They are implemented in very different ways.

`hash_map` (`unordered_map` in TR1 and Boost; use those instead) use a hash table where the key is hashed to a slot in the table and the value is stored in a list tied to that key.

`map` is implemented as a balanced binary search tree (usually a red/black tree).

An `unordered_map` should give slightly better performance for accessing known elements of the collection, but a `map` will have additional useful characteristics (e.g. it is stored in sorted order, which allows traversal from start to finish). `unordered_map` will be faster on insert and delete than a `map` .

edited Dec 18 '12 at 13:41

darcyq
1,253 1 12 25

answered Feb 3 '10 at 2:18

Joe
20.4k 4 53 87

5 I don't fully agree with you regarding the performance. The performance is influenced by a number of parameters and I would scold any programmer using an unordered_map for a mere 10 entries because "It's faster". Worry about interface / functionality first, performance later. – Matthieu M. Feb 3 '10 at 14:05

16 Well, yes, it helps if you understand your problem. Up to certain orders of magnitude it is probably a wash performance-wise, but it is important to understand the performance characteristics of both containers as they do deviate in different ways as data volumes get larger. – Joe Feb 3 '10 at 15:07

6 Interestingly, I just swapped a std::map with a boost::unordered_map in an application in which I do a lot of random lookups, but also iterate over all the keys in the map. I saved a large amount of time in lookup, but gained it back via the iterations, so I switched back to map and am looking for other ways to improve application performance. – Erik Garrison Sep 6 '10 at 21:36

@ErikGarrison If you use random access and iteration a lot more than you insert and delete elements, you could have your objects in both a tree and a hash_map (by storing a pointer, or better yet a shared_ptr, to the same objects in both in case you were using actual instances). You will then have O(1) time access time through the hash_map and O(n) iteration time through the map. Of course, you have to remember to add and remove the pointers from both every time. You could easily write a custom container class that (probably template it as well) that would encapsulate this behavior for you. – sprite Nov 23 '14 at 22:37

@ErikGarrison Of course, if you try this method, you would be paying with a minor additional space. However, since you'd be using pointers, that shouldn't be too much. If you really want to, you can go overboard, and write your own implementation of an AVL and use the node pointer as your data type in the hash_map, this will give you O(1) time access to a node in the tree from which you'll be able to iterate linearly to wherever you need. Of course this would involve quite a bit of coding and I'm not sure it would pay off unless you need to iterate a lot from and to random access locations. – sprite Nov 23 '14 at 22:44

`hash_map` was a common extension provided by many library implementations. That is exactly why it was renamed to `unordered_map` when it was added to the C++ standard as part of TR1. map is generally implemented with a balanced binary tree like a red-black tree (implementations vary of course). `hash_map` and `unordered_map` are generally implemented with hash tables. Thus the order is not maintained. `unordered_map` insert/delete/query will be O(1) (constant time) where

map will be O(log n) where n is the number of items in the data structure. So `unordered_map` is faster, and if you don't care about the order of the items should be preferred over `map`. Sometimes you want to maintain order (ordered by the key) and for that `map` would be the choice.

edited Dec 18 '12 at 13:42

darcyq
1,253 1 12 25

answered Feb 3 '10 at 2:24

janglin
429 3 4

9 I would point out that hashmap has a worst case access of O(N) when collisions are likely (bad hash fcn, loading factor too high, etc) – KitsuneYMG Feb 3 '10 at 6:25

A good hashmap has an expected cost of O(1), it is not guaranteed to be so. Bad hashmaps might have an expected cost that's not O(1). – Clearer Oct 5 '14 at 21:48

Some of the key differences are in the complexity requirements.

A map requires O(log(N)) time for inserts and finds.

An unordered_map requires an 'average' time of O(1) for inserts and finds but is allowed to have a worst case time of O(N).

So, usually, unordered_map will be faster, but depending on the keys and the hash function you store, can become much worse.

answered Feb 3 '10 at 5:39

R Samuel Klatchko
49.9k 7 79 143

The C++ spec doesn't say exactly what algorithm you must use for the STL containers. It does, however, put certain constraints on their performance, which rules out the use of hash tables for `map` and other associative containers. (They're most commonly implemented with red/black trees.) These constraints require better worst-case performance for these containers than hash tables can deliver.

Many people really do want hash tables, however, so hash-based STL associative containers have been a common extension for years. Consequently, they added `unordered_map` and such to later versions of the C++ standard.

edited Feb 3 '10 at 5:28

answered Feb 3 '10 at 2:17

Warren Young
19.6k 4 49 73

It was actually added in TR1 (std::tr1::unordered_map), not C++0x – Terry Mahaffey Feb 3 '10 at 5:22

Edited to fix it. – Warren Young Feb 3 '10 at 5:29

I thought that the reason `map` is generally a balanced btree was due to using `operator<()` as the means of determining location. – KitsuneYMG Feb 3 '10 at 6:26

@kts: Do any STL implementations actually use a B-tree? – bk1e Feb 3 '10 at 7:14

Technically all binary search trees are b-trees (a 1-2 tree). That being said, I don't know of any STL that uses anything other than red/black – KitsuneYMG Feb 3 '10 at 14:56