How to choose between map and unordered_map?

Suppose I wanted to map data with a string as the key. What container should I have chosen, `map` or `unordered_map` ? `unordered_map` takes up more memory so let's suppose memory isn't an issue, and the concern is speed.

`unordered_map` should generally give average complexity of O(1) with the worst case of O(n). In what cases would it get to O(n)? When does a `map` get more time efficient than `unordered_map` ? Does it happen when n is small?

Assuming I would use STL `unordered_map` with the default haser Vs. map. string is the key.

If I'm going to iterate over the elements rather than access an individual element each time, should I prefer `map` ?

c++ dictionary data-structures stl unordered-map

edited Jul 31 at 5:07 Krishna_Oza 396 2 18

asked Dec 10 '12 at 10:58 StackHeapCollision 445 1 3 15

2 Do you need to items in the mapping to be sorted? – Joachim Pileborg Dec 10 '12 at 11:02

Which implementation of `unordered_map` uses more memory? – Peter Wood Dec 10 '12 at 11:07

You always have memory overhead in a hash map, although it is typically negligible. – ypnos Dec 10 '12 at 11:09

Yes items would be stored and assuming I would use STL unordered_map with the default haser Vs. map. – StackHeapCollision Dec 10 '12 at 11:21

It's a minor point but as you mention iteration, it's worth pointing out that if you iterate while inserting elements, you should favor map over unordered_map. – JMcF Aug 29 '13 at 23:51

4 Answers

In practice, if memory is no issue, unordered_map is always faster if you want single element access.

The worst case is theoretical and bound to a single hash accounting for all of the elements. This is not of practical relevance. The unordered_map gets slower as soon as you have at least log N elements belonging to the same hash. This is also not of practical relevance. In some special scenarios you could use a specific hashing algorithm that ensures a more uniform distribution. For ordinary strings that don't share a specific pattern, the generic hash functions coming with unordered_map are just as good.

If you want to traverse the map (using iterators) in a sorted fashion, you cannot use unordered_map.

answered Dec 10 '12 at 11:07 ypnos 25.9k 6 56 94

```
                       | map             | unordered_map
---------------------------------------------------------
element ordering       | strict weak     | n/a
                       |                 |
common implementation  | balanced tree   | hash table
                       | or red-black tree|
                       |                 |
search time            | log(n)          | O(1) if there are no hash collisions
                       |                 | Up to O(n) if there are hash collisions
                       |                 | O(n) when hash is the same for any key
                       |                 |
Insertion time         | log(n)+rebalance | Same as search
                       |                 |
Deletion time          | log(n)+rebalance | Same as search
                       |                 |
needs comparators      | only operator <  | only operator ==
                       |                  |
needs hash function    | no               | yes
                       |                  |
common use case        | when good hash is| In most other cases.
                       | not possible or  |
                       | too slow. Or when|
                       | order is required|
```

edited Dec 10 '12 at 11:26 answered Dec 10 '12 at 11:16
user1773602

This is exactly what I want! Thanks! – daizuozhuo Nov 12 '13 at 6:42

Comment about common implementation: A red–black tree is a *kind* of balanced tree (or more specifically, a kind of self-balancing binary search tree). – HelloGoodbye Oct 18 at 14:44

> In what cases would it get to O(n)?

if you have such a **bad** hash function which produces the same hash value for all input stirngs (i.e. produce collisions)...

> What container should I have chosen, map or unordered_map?

It is always the questions of requirements and kind/amount of data do you have.

> When does a map get more time efficient than unordered_map?

It is just different structures. You'd better to make a chiose to use one of them depending on your typical use cases (takeing in account what kind of data do you have and its amount)

> Does it hppaen when n is small?

In case of small data amount everything depends on particular STL implementation... So sometimes even a plain vector/array could be faster than associative containers...

answered Dec 10 '12 at 11:07
zaufi
2,767 6 17

> What container should I have chosen, map or unordered_map? unordered_map takes up more memory so let's suppose memory isn't an issue, and the concern is speed.

Profile and then decide. `unordered_map` is generally faster, but it varies per case.

> In what cases would it get to O(n)?

When the hashing isn't good and a bunch of elements are being assigned to the same bins.

> When does a map get more time efficient than unordered_map? Does it happaen when n is small?

Probably not, but profile it if you really care. Having a container with a small size be the bottleneck of your program seems extremely unlikely. Anyway, a simple `vector` with linear search may be faster for such cases.

The most important thing when deciding is the requirements of ordering and lack of iterator invalidation. If you need either, you pretty much have to use `map` . Otherwise, `unordered_map` .

answered Dec 10 '12 at 11:09
Pubby
31.7k 3 70 130