Vector or Map or Hash map for C++?

I have a large number of records, say around 4,000,000, that I want to address them repeatedly and put information in a class that is linked to that record. I am not sure what kind of data structure should I use? Should I use the vectors, maps, or hash maps. I don't need to insert a record, but I need to read a table which contain sets of these records numbers (or names), and then grab some of the data which are linked to that record and do some processes on them. Is the finding on map fast enough to not go for hashmaps for this example? The records have a class as its structure and I have not done anything before with using the map or hashmap that has a class as its value (if it is possible). Thanks in advance guys.

Edited:

I do not need to have all the records on the memory at the same time for now> I need to first give it a structure first and then grab the data from some of the records. The total number of records is around 20 million, and I want to read each of these raw records and then if its basic info doesn't exist in my new map or vector that I want to create and put the rest of data in there as a vector. Because I have 20 million records, I think it would be very excruciating that for every record go through 4 million record to find if the basic info of that record exist or not. I have around 4 million type of packages and each of these packages could have more than one type of service (roughly around 5 (20/4) per package). I want to read each of these records and then if the package ID does not exist into the vector or whatever I want to use and push the basic info into the vector and then for the services that are related to that package be saved in a vector inside the package class.

c++ map vector hashmap

edited Sep 5 '12 at 1:20

asked Sep 5 '12 at 0:59

POD
156 1 12

Here's a good link (one of many): stackoverflow.com/questions/5139859/…. for whatever it's worth, it

sounds like "map" might suit your needs best. IMHO... – paulsm4 Sep 5 '12 at 1:02

There aren't enough details here to answer for sure - any of these structures might be appropriate. How are you looking up records in this list? Do you need them all to be loaded in memory at once? Can you elaborate about the use case? – vercellop Sep 5 '12 at 1:02

I'm not sure I understood what you want to do. It might help if you could enumerate briefly all the operations that you require... If you want fast retrieval with minimal overhead on a fixed-size dataset, go for std::array. If the size is variable, go for std::vector. If you need the stuff ordered in some way, go for std::map (or std::multimap if the ID can appear multiple times). If it doesn't need to be sorted, you may also use std::set, depending on the structure of the data. – Mihai Todor Sep 5 '12 at 1:05

what you are describing sounds like the job of a relational database. Otherwise, considering that you want fast lookups (for "linking" records to each other), you will probably want to use a hashtable, i.e. `unordered_map` . – Preet Kukreti Sep 5 '12 at 1:09

No, I do not need to have all the records on the memory at the same time for now> I need to give a structure first and then grab the data from some of the records. The total number of records is around 20 million, and I want to read each of these raw records and then if its basic info doesn't exist in my new map or vector that I want to create and put the rest of data in there as a vector. Because I have 20 million records, I think it would be very excruciating that for every record go through 4 million record to find if the basic info of that record exist or not. – POD Sep 5 '12 at 1:11

1 Answer

These three data structures have each a different purpose.

A `vector` is basically a dynamic array, which is good for indexed values.

A `map` is a sorted data-structure with O(log(n)) retrieval and insertion time (implemented using a balanced binary tree, usually Red-Black). This is best if you can't find an efficient hash method.

A `hash_map` uses hashes to retrieve object. If you have a well defined hash function with a low collision rate, you will get constant retrieval and insertion time *on average*. `hash_map` s are usually faster than `map` but not always. It is highly dependent on the hash function.

For your example, I think that it is best to use a `hash_map` where the key would be the record number (assuming record numbers are unique).

If these record numbers are dense (meaning there are little or no gaps between the indexes , say: 1,2,4,5,8,9,10...), you can use a `vector` . If your records are coming from a database with an autoincrement primary key and not many deletions, this should be usually the case.

edited Sep 5 '12 at 1:28

Mr Fooz
28.6k 3 44 74

answered Sep 5 '12 at 1:16

Samy Arous
5,369 5 13

How can I find out the time used for these types? For example I tried to make 1 million simple record of map. When I used the find() function, it was instance. But they say it of order O(log(n)). That does not make sense to me.I tried to search for the third to the last record, but the record came up instantly. If that is of order O(log(n)), so shouldn't it take log(1,000,000)? – POD Sep 5 '12 at 1:38

log(1,000,000) = 20 which is an instant for any computer. Complexity is a vast subject, but basically, when we say an operation order is O(log(n)), this means that it takes twice as much time to treat x*x objects as it would take to treat x objects. So for example, if the algorithm needs 10 sec to go through 1000 items, it would only take 20 sec to go through 1000*1000 = 1000 000 items and only 40sec to go through 1 000 000 000 000 items. This is extremely fast. – Samy Arous Sep 5 '12 at 1:51

Thank you. Took off a burden from me. – POD Sep 5 '12 at 2:32