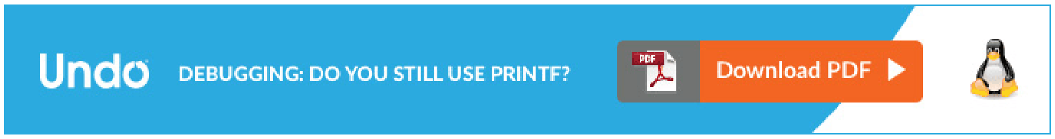
std::map, how to sort by value, then by key



I need to sort a map by value, then by key. I have a map with contents like this...

```
 1  realistically
 8         really
 4         reason
 3     reasonable
 1     reasonably
 1     reassemble
 1    reassembled
 2      recognize
92         record
48        records
 7           recs
```

I need to get the values in order, but the kicker is that the keys need to be in alphabetical order after the values are in order. What is the best way to go about this?

c++ algorithm sorting map key

asked Nov 7 '13 at 17:01
Trevor Hutto
503 ● 2 ● 5 ● 16

do you use a std::map to store the data ? – Raxvan Nov 7 '13 at 17:03

Put the std::pair<int, std::string>s into a list, and sort it. – Wilbert Nov 7 '13 at 17:03

@Raxvan yes, its stored in a map. – Trevor Hutto Nov 7 '13 at 17:04

@Wilbert I know that would put the values in order, but then would it keep the keys in alphabetical order after sorting the values? – Trevor Hutto Nov 7 '13 at 17:05

sort on pairs sorts on first, then second item. See here – Wilbert Nov 7 '13 at 17:11

4 Answers

`std::map` will sort its elements by `keys`. It doesn't care about the `values` when sorting.

You can use `std::vector<std::pair<K,V>>` then sort it using `std::sort` followed by `std::stable_sort`:

```
std::vector<std::pair<K,V>> items;

//fill items

//sort by value using std::sort
std::sort(items.begin(), items.end(), value_comparer);

//sort by key using std::stable_sort
std::stable_sort(items.begin(), items.end(), key_comparer);
```

The first sort should use `std::sort` since it is `nlog(n)`, and then use `std::stable_sort` which is `n^2log(n)`.

Note that while `std::sort` is chosen for performance reason, `std::stable_sort` is needed for correct ordering, as you want the order-by-value to be preserved.

@gsf noted in the comment, you could use *only* `std::sort` if you choose a comparer which compares `values` first, and IF they're equal, sort the `keys`.

```
auto cmp = [](std::pair<K,V> const & a, std::pair<K,V> const & b)
{
     return a.second != b.second?  a.second < b.second : a.first < b.first;
};
std::sort(items.begin(), items.end(), cmp);
```

That should be efficient.

But wait, there is a better approach: store `std::pair<V,K>` instead of `std::pair<K,V>` and then you don't need any comparer at all — the standard comparer for `std::pair` would be enough, as it compares `first` (which is `V`) first then `second` which is `K`:

```
std::vector<std::pair<V,K>> items;
//...
std::sort(items.begin(), items.end());
```

That should work great.

edited Nov 7 '13 at 17:46

answered Nov 7 '13 at 17:03
Nawaz
168k ● 40 ● 355 ● 577

Would the efficiency of the sort be hurt if I add to the vector in alphabetical order like I have it now? – Trevor Hutto Nov 7 '13 at 17:13

Oh I see, it wouldn't because you are sorting on value first? – Trevor Hutto Nov 7 '13 at 17:15

@TrevorHutto: If one (either `key` or `value`) is sorted, then you need only `stable_sort` to sort the other which is not sorted. – Nawaz Nov 7 '13 at 17:16 ✎

@TrevorHutto: Yes. – Nawaz Nov 7 '13 at 17:16

1 I got it working with the last solution in your answer, works like a charm. Thanks. – Trevor Hutto Nov 9 '13 at 1:30

You can use `std::set` instead of `std::map`.

You can store both key and value in `std::pair` and the type of container will look like this: `std::set< std::pair<int, std::string> >`. `std::set` will sort this values both by original keys and values.

answered Nov 7 '13 at 17:34
ks1322
11k ● 6 ● 31 ● 64

`std::map` already sorts the values using a predicate you define or `std::less` if you don't provide one. `std::set` will also store items in order of the of a define comparator. However neither set nor map allow you to have multiple keys. I would suggest defining a `std::map<int,std::set<string>` if you want to accomplish this using your data structure alone. You should also realize that `std::less` for string will sort lexicographically not alphabetically.

answered Nov 7 '13 at 17:07
rerun
16.3k ● 2 ● 25 ● 62

EDIT: The other two answers make a good point. I'm assuming that you want to order them into some other structure, or in order to print them out.

"Best" can mean a number of different things. Do you mean "easiest," "fastest," "most efficient," "least code," "most readable?"

The most obvious approach is to loop through twice. On the first pass, order the values:

```
if(current_value > examined_value)
{
    current_value = examined_value
    (and then swap them, however you like)
}
```

Then on the second pass, alphabetize the words, but only if their values match.

```
if(current_value == examined_value)
{
    (alphabetize the two)
}
```

Strictly speaking, this is a "bubble sort" which is slow because every time you make a swap, you have to start over. One "pass" is finished when you get through the whole list without making any swaps.

There are other sorting algorithms, but the principle would be the same: order by value, then alphabetize.

answered Nov 7 '13 at 17:09

mHurley
319 ● 1 ● 10