Sorting by indices, part 1

devblogs.microsoft.com/oldnewthing/20170105-00

January 5, 2017

Raymond Chen

Okay, now we're going to start using the `apply_permutation` function that we beat to death for first part of this week.

Suppose you are sorting a collection of objects with the property that copying and moving them is expensive. (Okay, so in practice, moving is not expensive, so let's say that the object is not movable at all.) You want to minimize the number of copies.

The typical solution for this is to perform an indirect sort: Instead of moving expensive things around, use an inexpensive thing (like as an integer) to represent the expensive item, and sort the inexpensive things. Once you know where everything ends up, you can move the expensive things just once.

```
template<typename Iter, typename Compare>
void sort_minimize_copies(Iter first, Iter last, Compare cmp)
{
 using Diff = std::iterator_traits<Iter1>::difference_type;
 Diff length = last - first;
 std::vector<Diff> indices(length);
 std::iota(indices.begin(), indices.end(), static_cast<Diff>(0));
 std::sort(indices.begin(), indices.end(),
    [&](Diff a, Diff b) { return cmp(first[a], first[b]); });
 apply_permutation(first, last, indices.begin());
}

template<typename Iter>
void sort_minimize_copies(Iter first, Iter last)
{
    return sort_minimize_copies(first, last, std::less<>());
}
```

We use `std::iterator_traits` to tell us what to use to represent indices, then we create a vector of those indices. (The difference type is required to be an integral type, so we didn't have to play any funny games like `first - first` to get the null index. We could just write `0` .)

We then sort the indices by using the indices to reference the original collection. (We also provide an overload that sorts by `<`.) This performs an indirect sort, where we are sorting the original collection, but doing so by mainpulating indices rather than the actual objects.

Once we have the indices we need, we can use the `apply_permutation` function to rearrange the original items according to the indices.

We'll wrap up next time with another kind of sorting.

Raymond Chen

Follow