Applying a permutation to a vector, part 3

devblogs.microsoft.com/oldnewthing/20170104-00

January 4, 2017

Raymond Chen

We spent the last two days looking at the `apply_permutation` function and arguing pros and cons of various implementation choices. Today's we're going to look at generalization.

One of the things you are taught in mathematics is that after you've proved something, you should try to strengthen the conclusion and weaken the hypotheses. Can we apply that principle here?

I don't see much that can be done to strengthen the conclusion, but I see a way to weak the hypotheses: The inputs don't actually have to be vectors. Anything that supports random access will do. So let's use a random access iterator.

And the indices don't have to be integers. Anything that can be used to index the random access iterator will do. So let's not require to to be an integer; we'll take whatever it is.

```
template<typename Iter1, typename Iter2>
void
apply_permutation(
    Iter1 first,
    Iter1 last,
    Iter2 indices)
{
 using T = std::iterator_traits<Iter1>::value_type;
 using Diff = std::iterator_traits<Iter2>::value_type;
 Diff length = last - first;
 for (Diff i = 0; i < length; i++) {
  Diff current = i;
  if (i != indices[current]) {
   T t{std::move(first[i])};
   while (i != indices[current]) {
    Diff next = indices[current];
    first[current] = std::move(first[next]);
    indices[current] = current;
    current = next;
   }
   first[current] = std::move(t);
   indices[current] = current;
  }
 }
}
```

Note that we used `std::iterator_traits` to determine the appropriate types for the indices and the underlying type. This is significant when the iterator returns a proxy type (such as the infamous `vector<bool>`).

Another observation is that the `indices` don't have to be in the range [0, *N* – 1]; as long as we can map the values into that range. But we don't need to generalize that, because that can already be generalized in another way: By creating a custom iterator whose `*` operator returns a proxy object that does the conversion.

Okay, I think I've run out of things to write about this `apply_permutation` function. But we'll use it later.

Exercise: Write an `apply_inverse_permutation` which applies the inverse of the specified permutation: Instead of each element of the `indices` telling you where the item comes from, it specifies where the item *goes to*. In other words, if `v` is a copy of the original vector and `v2` is a copy of the result vector, then `v2[indices[i]] = v[i]`.

Raymond Chen

Follow