One algorithm from The Book: A tribute to Ira Pohl

Alexander Stepanov
A9.com

The highest compliment [Erdős] could pay to a colleague's work was to say, “That's straight from The Book.”

Encyclopedia Britannica
CS needs its Book

The Book contains algorithms that are:

- Beautiful
- Optimal
- Useful
A Sorting Problem and Its Complexity

Ira Pohl
University of California*

*Santa Cruz, CA 95060.
Finding both $\text{min}$ and $\text{max}$

• To find minimum (or maximum) of $n$ elements we need $n – 1$ comparisons
• Don’t we need $2n – 2$ (or 3?) comparisons to find both?
• Ira showed that we need at most $\lceil \frac{3}{2} n \rceil - 2$ comparisons
• And he showed that his algorithm is optimal
maybe min or maybe max

not max

not min

not min and not max
Strict Weak Ordering

- Weak trichotomy
  \[ x \preceq y \lor y \preceq x \lor x \sim y \]

- Transitivity
  \[ (x \preceq y \land y \preceq z) \implies x \preceq z \]

- Irreflexivity, or strictness
  \[ \neg(x \preceq x) \]
template <StrictWeakOrdering R>
struct min_max
{
    R r;

    template <Regular T>  // T == Domain<R>
    const T& min(const T& x,
                  const T& y) const {
        if (r(y, x)) return y;
        else return x;
    }
}
Weak Commutativity

- Is min commutative?
- Not for StrictWeakOrdering
- Weak Commutativity!
  \[ a \circ b \sim b \circ a \]
- Set with min defined is
  - semigroup
  - (weak Abelian) semigroup
- Weak theories
  - equivalence axioms (instead of equational)
template <Regular T>  // T == Domain<R>
const T& max(const T& x,
const T& y) const {
    if (r(y, x)) return x;
    else return y;
}

// the idiot who designed STL wrote:
template <Regular T>  // T == Domain<R>
const T& max(const T& x,
             const T& y) const {
    if (r(x, y)) return y;
    else return x;
}

// why is it wrong?
template <Regular T> // T == Domain<R>
pair<T, T> construct(const T& x,
                     const T& y) const {
    if (r(y, x)) return {y, x};
    else return {x, y};
}
template <Regular T>  // T == Domain<R>
pair<T, T>
    combine(const pair<T, T>& x,  
             const pair<T, T>& y) const {
        return { min(x.first, y.first),  
                 max(x.second, y.second) };
    }
};
Iterators

• Input
• Forward
• Bidirectional
• RandomAccess
template <StrictWeakOrdering R>
struct compare_dereference
{

    R r;

    template <InputIterator I>
    // Domain<R> == ValueType<I>
    bool operator() (const I& i,
                       const I& j) const {
        return r(*i, *j);
    }
};
template <ForwardIterator I,
          StrictWeakOrdering R>
pair<I, I>
min_max_element_even_length(I first,
                            I last,
                            R r) {
    // assert(distance(first, last) % 2 == 0)
    min_max<compare_dereference<R>> op{r};
    if (first == last) return {last, last};
I prev = first;
pair<I, I> result =
    op.construct(prev, ++first);
while (++first != last) {
    prev = first;
    result = op.combine(
        result,
        op.construct(prev, ++first));
}
return result;
template <ForwardIterator I,
         StrictWeakOrdering R>
pair<I, I>
min_max_element(I first, I last, R r) {
    min_max<compare_dereference<R>> op{r};
    I prev = first;
    if (first == last || ++first == last)
        return {prev, prev};
pair<I, I> result =
    op.construct(prev, first);
while (++first != last) {
    prev = first;
    if (++first == last)
        return op.combine(result,
            {prev, prev});
    result = op.combine(
        result,
        op.construct(prev, first));
}
return result;
Type Functions

template <InputIterator I>
using ValueType = typename std::iterator_traits<I>::value_type;
template <InputIterator I, StrictWeakOrdering R>
   pair<ValueType<I>, ValueType<I>>
min_max_value_nonempty(I first,
         I last,
       R r) {

typedef ValueType<I> T;
min_max<R> op{r};
T val = *first;
if (++first == last) return {val, val};
pair<T, T> result =
    op.construct(val, *first);
while (++first != last) {
    val = *first;
    if (++first == last)
        return op.combine(result,
                          {val, val});
    result = op.combine(
        result,
        op.construct(val, *first));
}
return result;
template <InputIterator I,
         StrictWeakOrdering R>
pair<ValueType<I>, ValueType<I>>
min_max_value(I first, I last, R r) {
   typedef ValueType<I> T;
   if (first == last)
      return {supremum(r), infimum(r)}
   return min_max_value_nonempty(first,
                                  last,
                                  r);
}
• I have been teaching this algorithm every 2 – 3 years for the last 30 years
• When I teach it, I implement it anew
• Writing the code and teaching it gives me joy every time
THANK YOU, IRA!
Getting rid of an extra compare

// Add to min_max:

template <Regular T>  // T == Domain<R
pair<T, T> combine(const pair<T, T>& x,

    const T& val) const {

    if (r(val, x.first)) return { val, x.second};
    if (r(val, x.second)) return x;
    return {x.first, val};
}
Getting rid of an extra compare (2)

// In min_max_element and
// min_max_value_nonempty, replace:
if (++first == last)
    return op.combine(result, {val, val});

// with
if (++first == last)
    return op.combine(result, val);