Writing a sort comparison function

devblogs.microsoft.com/oldnewthing/20031023-00

October 23, 2003

Raymond Chen

When you are writing a sort comparison function (say, to be passed to `ListView_SortItems` or *gasp* to be used as an `IComparer`), your comparison function needs to follow these rules:

- **Reflexivity**: `Compare(a, a) = 0`.
- **Anti-Symmetry**: `Compare(a, b)` has the opposite sign of `Compare(b, a)`, where 0 is considered to be its own opposite.
- **Transitivity**: If `Compare(a, b) ≤ 0` and `Compare(b, c) ≤ 0`, then `Compare(a, c) ≤ 0`.

Here are some logical consequences of these rules (all easily proved). The first two are obvious, but the third may be a surprise.

- **Transitivity of equality**: If `Compare(a, b) = 0` and `Compare(b, c) = 0`, then `Compare(a, c) = 0`.
- **Transitivity of inequality**: If `Compare(a, b) < 0` and `Compare(b, c) < 0`, then `Compare(a, c) < 0`.
- **Substitution**: If `Compare(a, b) = 0`, then `Compare(a, c)` has the same sign as `Compare(b, c)`.

Of the original three rules, the first two are hard to get wrong, but the third rule is often hard to get right if you try to be clever in your comparison function.

For one thing, these rules require that you implement a total order. If you merely have a partial order, you must extend your partial order to a total order *in a consistent manner*.

I saw somebody get into trouble when they tried to implement their comparison function on a set of tasks, where some tasks have other tasks as prerequisites. The comparison function implemented the following algorithm:

- If `a` is a prerequisite of `b` (possibly through a chain of intermediate tasks), then `a < b`.

- If `b` is a prerequisite of `a` (again, possibly through a chain of intermediate tasks), then `a > b`.
- Otherwise, `a = b`. "Neither task is a prerequisite of the other, so I don't care what order they are in."

Sounds great. Then you can sort with this comparison function and you get the tasks listed in some order such that all tasks come after their prerequisites.

Except that it doesn't work. Trying to sort with this comparison function results in all the tasks being jumbled together with apparently no regard for which tasks are prerequisites of which. What went wrong?

Consider this dependency diagram:

```
a ----> b
c
```

Task "a" is a prerequisite for "b", and task "c" is unrelated to both of them. If you used the above comparison function, it would declare that "a = c" and "b = c" (since "c" is unrelated to "a" or "b"), which in turn implies by transitivity that "a = b", which contradicts "a < b", since "a" is a prerequisite for "b". If your comparison function is inconsistent, you will get garbled results.

Moral of the story: When you write a comparison function, you really have to know which items are less than which other items. Don't just declare two items "equal" because you don't know which order they should be in.

Raymond Chen

Follow