Thoughts on finding the essential elements of a set

devblogs.microsoft.com/oldnewthing/20240826-00

August 26, 2024

Raymond Chen

Suppose you have a set of *n* items, two of which are essential, and the rest are superfluous. You can pass any subset of these items to an oracle, and the oracle will tell you whether the set contains all of the essential elements. The objective is to identify those essential elements.

You might run into this problem if the elements are package dependencies, and you want to figure out which ones are actually necessary for your project to build, and which ones are just cargo cult.

If the problem had been formulated with just one essential element, then it would be a simply binary search: Divide the set into equal-sized subsets and ask the oracle which subset contains the essential element. Recurse on that subset, and you can find the essential element in $O(\log n)$ steps.

But what if there are *two* essential elements? You could try the same thing and divide in half, but if the oracle says “Neither half contains both of the essential elements,” then you’re in a bit of a pickle because you don’t know which pieces of the two halves need to be combined.

One option is to try to peel off the essential elements one at a time. For example, an inefficient algorithm would be to remove one element and ask the oracle of the remaining elements include all the essential elements. If it says yes, then you can recurse with the smaller set. if it says no, then you know that the element you removed is one of the essential elements, and you can now use the “find one essential element” algorithm on the rest. (Just remember to add the essential element you already found to each query you pass to the oracle.)

Now that we have an inefficient algorithm, we can try to make it more efficient: Instead of removing one element at a time, you can use a binary search to find the “highest-numbered essential element”: At the start, you know that the (zero-based) index of the highest-numbered essential element is somewhere between 1 and $n - 1$,[1] inclusive. At each step, find the midpoint between the low and high boundaries of the range and ask the oracle whether all the elements up to that midpoint element include all the essential elements. If so, then you can move the upper boundary of the range down to the midpoint; if not, then you can move the lower boundary of the range up to the midpoint. In this way, you can do a binary search on “the highest-numbered essential elements.”

And then once you’ve found one essential element, you can use a regular binary search to find the other one.

We can generalize this to the case where there are m essential elements: Start with a known range of $(m - 1)$ to $(n - 1)$, and use binary search to find the highest-numbered essential element. Once you’ve done that, you’ve reduced the problem to finding the $m - 1$ essential elements below the highest-numbered essential element, and so on, for a total complexity of $O(m \log n)$.

[1] You know that it cannot be zero, because there are two essential elements, and the earliest you can get both of them is if one of them is at index 0 and the other is at index 1.