Daily Coding Problem #198

Problem

This problem was asked by Google.

Given a set of distinct positive integers, find the largest subset such that every pair of elements in the subset (i, j) satisfies either i % j = 0 or j % i = 0.

For example, given the set [3, 5, 10, 20, 21], you should return [5, 10, 20]. Given [1, 3, 6, 24], return [1, 3, 6, 24].

Solution

The brute force solution would generate all subsets of numbers and, for each one, compare all pairs of numbers to check divisibility.

Since there are 2^N subsets of any set, and looking at all pairs of each subset is $O(N^2)$, this would take $O(2^N * N^2)$. We must find a better solution.

Note that, for any number a and b, if a | b, then every element that divides a will also divide b. So if we have a sorted list, knowing how many divisors each element has before k will also tell us how many divisors the kth element has- just one more than that of its greatest divisor. Therefore, we can use dynamic programming to find the largest subset that includes a given number by looking at the sizes of previously computed subsets.

To make this more concrete, suppose we are using the list `[5, 10]`.

Now we look at the second element. Since 5 | 10, and 5 had one divisor, `num_divisors[1]` = `num_divisors[0]` + 1 = 2.

Finally, for each element in the solution subset, we store the index where we can find the next highest element in the subset. In other words, if a < b < c, then `prev_divisor_index[c]` would be the index of b, and `prev_divisor_index[b]` would be the index of a.

Let's see how this looks in code:

```
def largest_divisible_subset(nums):
    if not nums:
        return []

    nums.sort()

    # Keep track of the number of divisors of each element, and where to find
    # its last divisor.
    num_divisors = [1 for _ in range(len(nums))]
    prev_divisor_index = [-1 for _ in range(len(nums))]

    # Also track the index of the last element in the best subset solution so far.
    max_index = 0

    # For each element, check if a previous element divides it. If so, and if adding
    # the element will result in a larger subset, update its number of divisors
    # and where to find its last divisor.
    for i in range(len(nums)):
        for j in range(i):
            if (nums[i] % nums[j] == 0) and (num_divisors[i] < num_divisors[j] + 1):
                num_divisors[i] = num_divisors[j] + 1
                prev_divisor_index[i] = j

        if num_divisors[max_index] < num_divisors[i]:
            max_index = i

    # Finally, go back through the chain of divisors and get all the subset elements.
    result = []
    i = max_index
    while i >= 0:

        result.append(nums[i])
        i = prev_divisor_index[i]

    return result
```

Since we are looping through the list twice and storing lists of size N, this has time complexity O(N^2) and space complexity O(N).