Largest Divisible Subset

Given a set of **distinct** positive integers, find the largest subset such that every pair (S_i, S_j) of elements in this subset satisfies: $S_i \% S_j = 0$.

If there are multiple solutions, return any subset is fine.

Example 1:

```
nums: [1,2,3]

Result: [1,2] (of course, [1,3] will also be ok)
```

Example 2:

```
nums: [1,2,4,8]

Result: [1,2,4,8]
```

Credits:

Special thanks to @Stomach_ache for adding this problem and creating all test cases.

Solution 1

The key concept here is:
Given a set of integers that satisfies the property that each pair of integers inside the set are mutually divisible, for a new integer S, S can be placed into the set as long as it can divide the smallest number of the set or is divisible by the largest number of the set.

For example, let's say we have a set P = { 4, 8, 16 }, P satisfies the divisible condition. Now consider a new number 2, it can divide the smallest number 4, so it can be placed into the set; similarly, 32 can be divided by 16, the biggest number in P, it can also placed into P.

Next, let's define:

EDIT: For clarification, the following definitions try to enlarge candidate solutions by appending a larger element at the end of each potential set, while my implementation below is prefixing a smaller element at the front of a set. Conceptually they are equivalent but by adding smaller elements at the front saves the trouble for keeping the correct increasing order for the final answer. Please refer to comments in code for more details.

For an increasingly sorted array of integers a[1 .. n]

T[n] = the length of the largest divisible subset whose largest number is a[n]

T[n+1] = max{ 1 + T[i] if a[n+1] mod a[i] == 0 else 1 }

Now, deducting T[n] becomes straight forward with a DP trick. For the final result we will need to maintain a backtrace array for the answer.

Implementation in C++:

```cpp
class Solution {
public:
    vector<int> largestDivisibleSubset(vector<int>& nums) {
        sort(nums.begin(), nums.end());

        vector<int> T(nums.size(), 0);
        vector<int> parent(nums.size(), 0);

        int m = 0;
        int mi = 0;

        // for(int i = 0; i < nums.size(); ++i) // if extending by larger element
s
        for(int i = nums.size() - 1; i >= 0; --i) // iterate from end to start si
nce it's easier to track the answer index
        {
            // for(int j = i; j >=0; --j) // if extending by larger elements
            for(int j = i; j < nums.size(); ++j)
            {
                // if(nums[i] % nums[j] == 0 && T[i] < 1 + T[j]) // if extending
by larger elements
                // check every a[j] that is larger than a[i]
                if(nums[j] % nums[i] == 0 && T[i] < 1 + T[j])
                {
                    // if a[j] mod a[i] == 0, it means T[j] can form a larger sub
set by putting a[i] into T[j]
                    T[i] = 1 + T[j];
                    parent[i] = j;

                    if(T[i] > m)
                    {
                        m = T[i];
                        mi = i;
                    }
                }
            }
        }

        vector<int> ret;

        for(int i = 0; i < m; ++i)
        {
            ret.push_back(nums[mi]);
            mi = parent[mi];
        }

        // sort(ret.begin(), ret.end()); // if we go by extending larger ends, th
e largest "answer" element will come first since the candidate element we observe
will become larger and larger as i increases in the outermost "for" loop above.
       // alternatively, we can sort nums in decreasing order obviously.

        return ret;
    }
};
```

written by roy14 original link here

Solution 2

```
def largestDivisibleSubset(self, nums):
    S = {-1: set()}
    for x in sorted(nums):
        S[x] = max((S[d] for d in S if x % d == 0), key=len) | {x}
    return list(max(S.values(), key=len))
```

My `S[x]` is the largest subset with `x` as the largest element, i.e., the subset of all divisors of `x` in the input. With `S[-1] = emptyset` as useful base case. Since divisibility is transitive, a multiple `x` of some divisor `d` is also a multiple of all elements in `S[d]`, so it's not necessary to explicitly test divisibility of `x` by all elements in `S[d]`. Testing `x % d` suffices.

While storing entire subsets isn't super efficient, it's also not that bad. To extend a subset, the new element must be divisible by all elements in it, meaning it must be at least twice as large as the largest element in it. So with the 31-bit integers we have here, the largest possible set has size 31 (containing all powers of 2).

written by StefanPochmann original link here

Solution 3

Use DP to track max Set and pre index.

```
public class Solution {
    public List<Integer> largestDivisibleSubset(int[] nums) {
        int n = nums.length;
        int[] count = new int[n];
        int[] pre = new int[n];
        Arrays.sort(nums);
        int max = 0, index = -1;
        for (int i = 0; i < n; i++) {
            count[i] = 1;
            pre[i] = -1;
            for (int j = i - 1; j >= 0; j--) {
                if (nums[i] % nums[j] == 0) {
                    if (1 + count[j] > count[i]) {
                        count[i] = count[j] + 1;
                        pre[i] = j;
                    }
                }
            }
            if (count[i] > max) {
                max = count[i];
                index = i;
            }
        }
        List<Integer> res = new ArrayList<>();
        while (index != -1) {
            res.add(nums[index]);
            index = pre[index];
        }
        return res;
    }
}
```

written by jiangbowei2010 original link here