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:

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++:

```
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 = 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 < nums.size(); ++j)
            {
                // 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];
        }

        return ret;
    }
};
```

written by roy14 original link here

Solution 2

Update: The question has now been updated and is correct.

I have a major problem with this question it says `Given a set of distinct positive integers, find the largest subset such that **every pair** (Si, Sj) of elements in this subset satisfies: Si % Sj = 0.`

The key word here is every, lets look at one of the examples

```
input is nums = [1,2,3]

then the result is [1,2]
```

but in the question is say that for every pair in our solution we should have Si % Sj = 0. So lets look at every pair we have the first pair (2, 1) and we have 2%1 = 0, so that holds. Now lets look at the other pair (1,2) 1%2 = 1 this is not zero hence [1,2] cannot be the solution.

If we require **every pair** so that **Si % Sj = 0** then your subset will be a set of one item since we require both that **Sj% Si = 0** and **Si % Sj = 0** which forces that **Si = Sj**.

So please fix the requirements of the question to reflect what the author intended.

written by kevin36 original link here

Solution 3

```
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