Divide by three, multiply by two

Giordano Colli

+TAGS: Sorting, Graphs, Binary Search
+Difficulty: 1400
+Description: A sorting Trick is used into the main proof
+Problem Link: Codeforces Link

1 Problem Definition

Given an array $A =< a_1, a_2, ..., a_n >$:

- $n \in \mathbb{N}^+$
- $a_i \in \mathbb{N}^+ \quad \forall 1 \leq i \leq n$

Rearrange the indices $i_1, i_2, ..., i_n$ such that

$$a_{i_{k+1}} = 2 \cdot a_{i_k} \quad \vee \quad a_{i_{k+1}} = \frac{a_{i_k}}{3}$$

Where a_{i_k} must be divisible by 3 if the second condition holds.
It is guaranteed that such rearrangement exists.

2 Example

Input: $< 4, 8, 6, 3, 12, 9 >$
Output: $< 9, 3, 6, 12, 4, 8 >$
Explanation: starting from 9 :

- $\frac{9}{3} = 3$
- $3 \cdot 2 = 6$
- $6 \cdot 2 = 12$
- $\frac{12}{3} = 4$
- $4 \cdot 2 = 8$

3 Graph Approach

An intuitive representation of the problem is a directed graph $G = (V, E)$ in which:

- $V = \{a_1, ..., a_n\}$
- $(a_u, a_v) \in E \implies a_v = 2 \cdot a_u \vee a_v = \frac{a_u}{3}$

The intuition behind this is to find an **Hamiltonian Path** in the graph G to solve the problem.
The Hamiltonian path found is feasible by construction.
It is known that finding a Hamiltonian path is an NP-hard problem even on graphs with bounded degrees.
Note that the in-degree plus the out-degree of each node is at most 4 by construction.
Observing the graph lets us deduce an interesting property, check the following example:

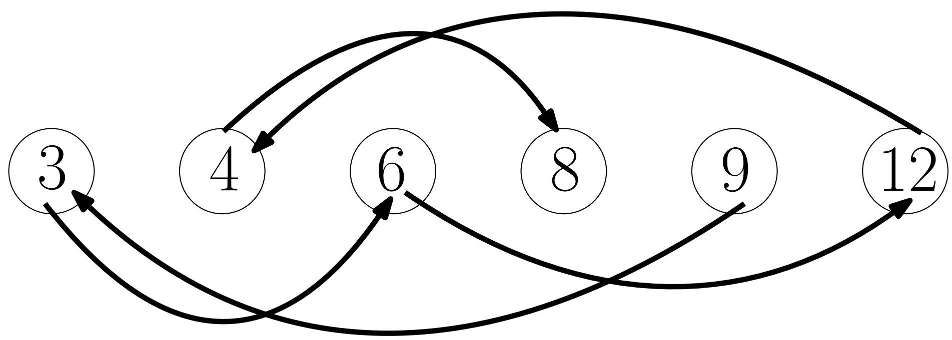

Figure 1: $< 4, 8, 6, 3, 12, 9 >$

Lemma 3.1. *The graph G is acyclic.*

Proof. Consider a Cycle $u, (u, v), v, ..., v_n, (v_n, u)$.
Each edge of the cycle can be considered an operation that:

- doubles the value of u
- divides by 3 the value of u

The order in which the operations are made does not affect the final result.

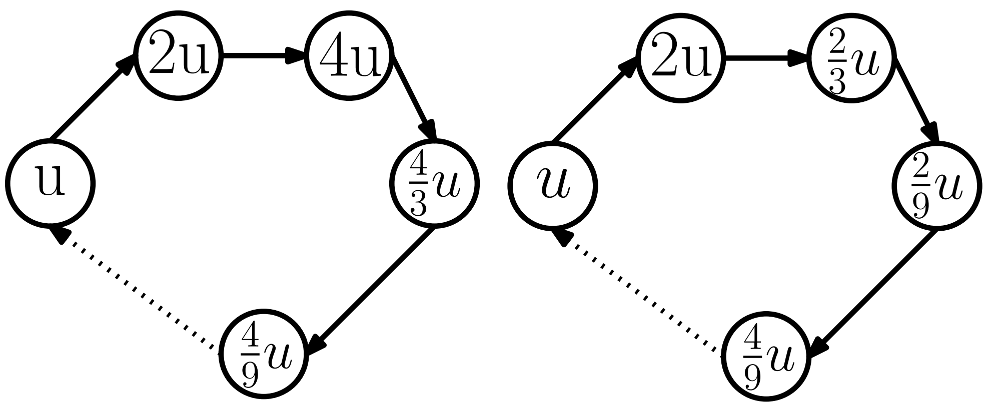

Figure 2: 2 Cycles that are performing the same operations in a different order

Suppose that there are k edges into a cycle, $a \in \mathbb{N}^+$ multiplying by 2 operations and $b \in \mathbb{N}^+$ dividing by 3 operations. Clearly $a + b = k$.
A cycle of size k can exists if the final result of this cycle is u itself, then

$$\frac{2^a}{3^b} u = u$$

$$2^a = 3^b$$

$$a = \log_2 3^b$$

$$a = b \log_2 3$$

Since a, b are integer numbers greater or equal than 1 and $\log_2 3$ is not an integer number, $a \neq b \log_2 3$ no matter the choices of a and b. $\square$

Since G is acyclic it is possible to topologically sort the graph, the problem statement ensures that a Hamiltonian path exists. This implies that each vertex must be alone in each "layer" of the topologically sorted graph. Formally given a topological sort $\sigma : V \to \{1, ..., n\}$, $\sigma(u) \neq \sigma(v) \quad \forall \{u, v\} \in [V]^2$. Otherwise, if there are $u, v \in V : u \neq v, \sigma(u) = \sigma(v)$ a Hamiltonian path can not exist. To show that it is sufficient to note that u must be visited. Since there are no cycles, it is no longer possible to visit v from u because from u it is possible to visit only nodes $\alpha \in V : \sigma(\alpha) > \sigma(u)$.

Algorithm 1: GRAPH-ALGORITHM(A)

```
G ← (V = ∅, E = ∅);
foreach a_i ∈ A do
    V ← V ∪ {a_i} ;
    if 2 · a_i ∈ A then
        E ← E ∪ (a_i, 2 · a_i) ;
    if a_i/3 ∈ A then
        E ← E ∪ (a_i, a_i/3) ;
σ ← Topologically sort G ;
return σ;
```

- Inserting vertices into G costs $O(n)$ time.
- Checking if there are vertices to attach edges costs $O(n)$ time for each edge. Moreover, there are $O(n)$ edges since the maximum degree is 4.
- Topological sort costs $O(n + m) \in O(n)$ time, since $m \in O(n)$.

TIME: $O(n^2)$.
MEMORY: $O(n)$.

The main bottleneck is the construction of the graph.
Checking if there are $2 \cdot a_i, \frac{a_i}{3} \in A$ can be simply done in $O(\log n)$ if A is sorted. Using binary search we can improve the time complexity of the algorithm to $O(n \log n)$.

Algorithm 2: GRAPH-ALGORITHM-BINARY-SEARCH(A)

Sort A in non-decreasing order ;
$G \leftarrow (V = \emptyset, E = \emptyset)$;
foreach $a_i \in A$ **do**
  $V \leftarrow V \cup \{a_i\}$;
  if $2 \cdot a_i \in A$ *using binary search* **then**
    $E \leftarrow E \cup (a_i, 2 \cdot a_i)$;
  if $\frac{a_i}{3} \in A$ *using binary search* **then**
    $E \leftarrow E \cup (a_i, \frac{a_i}{3})$;
$\sigma \leftarrow$ **Topologically sort** G ;
return σ;

- Sorting A costs $O(n \log n)$ time.
- Inserting vertices into G costs $O(n)$ time .
- Checking if there are vertices to attach edges costs $O(\log n)$ time for each edge. There are $O(n)$ edges since the maximum degree is 4. This implies that the total time is $O(n \log n)$
- Topological sort costs $O(n + m) \in O(n)$ time, since $m \in O(n)$.

TIME: $O(n \log n)$.
MEMORY: $O(n)$.

4 Lower Bound

A trivial lower bound of this problem can be found by inspecting instances.
Consider an instance that has only power of 2 elements.
There is a single feasible permutation of elements: $< 2^1, 2^2, 2^3, ..., 2^n >$. Since there exists a map from $2^i -> i$ that is the $log_2(2^i)$ the algorithm sorts numbers from 1 to n, in other words, there is a linear reduction from sorting numbers $\in \{1, ..., n\}$ to this problem. Sorting numbers $\in \{1, ..., n\}$ using comparisons has a lower bound of $\Omega(n \log n)$. **Note** that is possible to precalculate all the logarithms in $O(n)$ time!

5 Sorting Approach

This type of reasoning can be used if it is easy to see that the problem is about sorting elements
Since there must be a total order relation, try to find it.
Think to the solution and call it elements $B =< b_1, ..., b_n >$, focus on b_i and b_{i+1}, there are 2 cases:

- $b_{i+1} = 2 \cdot b_i$
- $b_{i+1} = \frac{b_i}{3}$

These 2 elements differ at most by a factor of 3. If an element is divisible by 3, say k times, and another is divisible by 3 $k+1$ times, the first element can not be before the second.
Call $deg_3(b_i) = \max\{y \in \mathbb{N} : 3^y | b_i\}$ the maximum number of times that 3 divides a number b_i.

Proposition 5.1. $deg_3(b_i) > deg_3(b_j) \iff i < j \quad 1 \leq i < j \leq n$ //in the optimal solution B

Proof. There are 2 cases:

- $b_j = \frac{b_i}{3} \iff deg_3(b_i) = deg_3(b_j) + 1 > deg_3(b_j)$
- $b_j = 2 \cdot b_i \iff deg_3(b_i) = deg_3(b_j)$

□

If $deg_3(b_i) = deg_3(b_j)$ clearly b_j must be equal to $2 \cdot b_i$ since they differs of a factor less than 3.
In other words, is not possible to increment $deg_3(b_i)$ by multiplying b_i by 2.
Think to the decomposition of $b_i = 3^j \cdot q$, where $3 \nmid q$, $2 \cdot b_i = 3^j \cdot q \cdot 2$ where $3 \nmid 2 \cdot q$.
The algorithm is based on the fact that if $deg_3(a_i) > deg_3(a_j)$, a_i must precede a_j in the feasible solution.
If there are $a_i, a_j : deg_3(a_i) = deg_3(a_j)$ must be that $a_i = 2 \cdot a_j \vee a_j = 2 \cdot a_i$, then sort all the elements with the same deg_3 with respect to their size in non-decreasing order.

Algorithm 3: SORTING-ALGORITHM(A)

$B \leftarrow \emptyset$;
foreach $a_i \in A$ **do**
    Calculate $deg_3(a_i)$;
    $B \leftarrow B \quad \cup < deg_3(a_i), a_i >$
Sort **lexicographically** B.;
return B;

- Calculating $deg_3(a_i)$ takes $O(\log(a_i))$ time, since only values like $3^j \leq a_i$ are tested.
- Sorting lexicographically takes $O(n \log n)$ time.

TIME: $O(\max\{n \log(\max\{a_i\}), n \log n\})$
MEMORY: $O(n)$
This algorithm is pseudo-polynomial, but if values of a_i are bounded the time complexity is the same as the GRAPH-ALGORITHM-BINARY-SEARCH(A).

TRICK: Calculating $deg_3(a_i)$ takes $O(n \max\{\log(\max\{a_i\}))$ time. Due to the monotonicity of $deg_3(a_i)$, we can calculate this value faster.
Think of how is defined $deg_3(a_i) = \max\{y \in \mathbb{N} : 3^y | a_i\}$, this is sufficient to check that

$$3^{deg_3(a_i)} | a_i \implies 3^{deg_3(a_i)-1} | a_i$$

Do a binary search on this value to perform a search that runs in $O((\log j = \log\log 3^j) \cdot \log j) \in O((\log j)^2) \in O((\log\log a_i)^2) \quad \forall a_i \in A$ time.
The first $O(\log j)$ factor is given by the **binary search** on the factor j.
The second $O(\log j)$ factor is given by the **calculation of** 3^j each time that j is fixed.
Note that you can not precalculate all the possible 3^j values because $max\{a_i \in A\}$ is **unbounded**.