<table>
<thead>
<tr>
<th>Character</th>
<th>Number of Solves</th>
<th>Character</th>
<th>Number of Solves</th>
</tr>
</thead>
<tbody>
<tr>
<td>whatever</td>
<td>37</td>
<td>groot</td>
<td>13</td>
</tr>
<tr>
<td>thor</td>
<td>35</td>
<td>hawkeye</td>
<td>22</td>
</tr>
<tr>
<td>assemble</td>
<td>24</td>
<td>blackpanther</td>
<td>6</td>
</tr>
<tr>
<td>scarletwitch</td>
<td>26</td>
<td>thanos</td>
<td>4</td>
</tr>
<tr>
<td>Character</td>
<td>Novice Solves</td>
<td>groot Solves</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>whatever</td>
<td>0</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>thor</td>
<td>1</td>
<td>hawkeye</td>
<td></td>
</tr>
<tr>
<td>assemble</td>
<td>5</td>
<td>blackpanther</td>
<td></td>
</tr>
<tr>
<td>scarletwitch</td>
<td>8</td>
<td>thanos</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>62</td>
<td></td>
</tr>
</tbody>
</table>
## Advanced: Number of Solves

<table>
<thead>
<tr>
<th>Superhero</th>
<th>Solves</th>
<th>Antagonist</th>
<th>Solves</th>
</tr>
</thead>
<tbody>
<tr>
<td>assemble</td>
<td>29</td>
<td>blackwidow</td>
<td>5</td>
</tr>
<tr>
<td>ironman</td>
<td>24</td>
<td>asgardians</td>
<td>0</td>
</tr>
<tr>
<td>vision</td>
<td>9</td>
<td>hulk</td>
<td>0</td>
</tr>
<tr>
<td>captainamerica</td>
<td>4</td>
<td>spiderman</td>
<td>0</td>
</tr>
<tr>
<td>titan</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Solve Time</td>
<td>Team</td>
<td>Time</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>assemble</td>
<td>2</td>
<td>blackwidow</td>
<td>54</td>
</tr>
<tr>
<td>ironman</td>
<td>8</td>
<td>asgardians</td>
<td>N/A</td>
</tr>
<tr>
<td>vision</td>
<td>10</td>
<td>hulk</td>
<td>N/A</td>
</tr>
<tr>
<td>captainamerica</td>
<td>42</td>
<td>spiderman</td>
<td>N/A</td>
</tr>
<tr>
<td>titan</td>
<td>68</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ProCo 2019

Speed Round

Solutions
Novice Solutions
We first look at how many cubes with side length “a” can fit along one direction of cubes with side length “b”

This quantity is \( \frac{b}{a} \)--using integer division

Thus we can fit \( \frac{b}{a} \) cubes in each direction, since a cube is symmetric

We can fit a total of \( \left( \frac{b}{a} \right)^3 \) number of cubes

This solution runs in \( O(1) \) time
Observations:
○ The most frequent substring will be of length 1
○ The least frequent substring will be the entire string (1 substring)

For the least frequent substring, we just count the frequency of every letter and output the max
For the most frequent substring, we just output the entire string
This runs in $O(n)$ time
Note that we can ignore trailing 1’s and leading 0’s. We can treat this as just chopping those digits off the string.
Otherwise we can just reverse the entire string and look at the string again.
We don’t have to physically reverse the string, just look at it forwards or backwards.
Keep track of how many reverses we do.
This runs in $O(n)$ time.
Movement is deterministic and can be easily simulated in $O(n)$ time.
Since there are $n$ instructions and only 4 types of instructions, there are only $3n$ possible single-command changes, and we can just try all of them.
This leads to an $O(n^2)$ solution.
It is possible to update the end position in $O(1)$ time by keeping both the change in position and the change in orientation of each suffix and prefix of instructions. This was not necessary for the problem.
● Create a min heap/priority queue and insert all numbers into it.
● Repeat until min heap is empty:
  ○ extract two minimum elements from the heap
  ○ add their sum to the heap
  ○ Update the total cost
● We have n inserts and the loop is repeated n-1 times, each insert() and getMin() is $O(\log n)$. So the algorithm is $O(n \log n)$.
● We also accept $O(n^2 \log n)$ solutions that use an array/list and sort it after every insert().
Notice the number of operations we can do is less than 9. We can brute force.

Try all permutations of the order of numbers and see which ordering is maximal.

This $O(n! \times n)$ solution runs in time since $n \leq 9$. 
Think about two planets with periods A and B (assume A<B). After D days, planet 1 has travelled D/A rotations and planet 2 has travelled D/B. If they’re “in line” then their difference is an integer number of rotations:

\[ \frac{D}{A} - \frac{D}{B} = \text{an integer } k \rightarrow \frac{D(B-A)}{AB} = k \rightarrow D = \frac{kAB}{B-A} \]

We need all \( n \) planets in line, so planet 1 is in line with 2, 1 is in line with 3, … 1 is in line with \( n \)

We get \( n-1 \) fractions

thanos
Our final answer fraction must be an integer multiple of all these fractions. (A “Lowest Common Multiple” but for fractions)

For two fractions: A/B and C/D → find smallest common denominator G, so A'/G and C'/G, then find the lcm of integers A' and C'

Use the Euclidean Algorithm to find GCF, then LCM(x,y)=xy/GCF(x,y)

Find the lcm of the first two fractions. Then the lcm of this answer and the next fraction etc.

This runs in O(n) time
  ○ Careful: numbers get large
Advanced Solutions
The “relative ordering” of a sequence of numbers $a_1, a_2, ..., a_n$ is a permutation of $1, 2, ..., n$ if where the smallest number is replaced by 1, the second smallest is 2, ..., largest is $n$.

Observation: If you swap two rows, the relative ordering of each row is unchanged. If you swap two columns, all of the relative orderings of rows change in the same way.

- The same is true for the relative ordering of rows

Output Yes if all the rows have the same relative ordering and all the columns have the same relative ordering (as other columns, can be distinct from the rows)

- Note that any relative ordering is sortable. If there are two distinct relative orderings in rows, if you sort one, the other cannot be sorted.

Runs in $O(n^2 \log(n))$
We can notice that the prime factorization of $n^3$ must be the same as the prime factorization $c(m^2)$.

We can greedily choose the smallest numbers to build $n$ and $m$.

Sieve of Eratosthenes to find primes from 1 to $10^6$.

Prime factorize $c$.

To build $n$ and $m$ consider each $p^e$ in $X$:

- If $e=1,2$: $p^e$ should be in $n$ and $m$.
- If $e > 2$: the smallest exponents for $p$ in $n$ and $m$ are:
  - If $p = 3k$: $n$ has $p^k$, $m$ has $p^0$.
  - $p = 3k+1$: $n$ has $p^{k+1}$, $m$ has $p^1$.
  - $p = 3k+2$: $n$ has $p^{k+2}$, $m$ has $p^2$.

Careful: This process might get $n=m$. If so, multiply $n$ by 4 and $m$ by 8.

$O(N \log(N) + Q)$.
● Type 0 operations add exactly one triangle when \((u, v)\) is an existing edge.
● Type 1 operations will only remove triangles.
● So, there can only be up to \(q\) triangles, and each triangle is inserted and removed at most once.
● For every edge \((u,v)\) store the set of vertices that are connected to \(u\) and \(v\), with a set (i.e. all the triangles including \((u,v)\))
● Process all the operations in \(O(q \log q)\) time.
Can assume all heights will be the same (as the max height)

For two towers at \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) find an equation for the minimum height they need to be to see each other:

- Angle between them = \(\arccos(x_1 x_2 + y_1 y_2 + z_1 z_2)\)
- Min Height = \(1/\cos(\text{angle}/2) - 1\)

Solution 1: Kruskal’s Algorithm
- Sort all pairs of towers by increasing tower height needed
- Use a union-find data structure to find the minimum height needed to make everything one connected component

Solution 2: Binary Search
- For a given height \(H\), link all the pairs with min height \(\leq H\)
- See if everything is connected

Runs in \(O(N^2 \log N)\) and \(O(N^2 \log(\text{max height}))\) respectively
Maintain a map mp from box number to expected number of balls

Initially the expected number of balls for the ith box is $a_i$

To update mp after operation $[l, r]$, iterate over the keys in that range and compute the sum of the values

After iterating over a key, remove it from the map

Set mp[l] and mp[r] to sum/2 (if $l = r$ we don’t do anything)

Runtime is $O((n + m)\log n)$

- For each operation, we add at most 2 entries to the map so the total number of additions to the map is at most $n + 2*m$
- We can only remove an entry from a map once, so the total number of removals across all operations is $O(n + m)$
Create graph of 26 nodes (one for each letter)

If we have to change letter $c_1$ to $c_2$ at some point, create directed edge from $c_1$ to $c_2$.

Impossible cases:

○ Some letter needs to be changed to two different letters
  ■ $S = abba$ and $T = abca$

○ Edges (of all 26 nodes) form a set of simple cycles (self-loops count as cycles) unless everything is a self-loop (in which case the strings are identical)
• Answer: number of edges in the graph (excluding self-loops) + number of simple cycles (excluding self-loops)
• Resolve tree components: process changes from root outwards
• Resolve flower components: change a to c, then resolve the tree rooted at a

• Resolve cycle components:
  ○ Let d be a leaf node of some flower that has already been resolved
  ○ Change a to d and resolve the tree rooted at a
  ○ Change d back to a (creates an additional unit of cost for a cycle)
• This solution runs in O(n)
Only need to consider grids where all diagonals are the same character

abcd  abcd
qwer  ->  bcdr
asdf  cdrf
xzcv  drfv

- Hence, all we need is a sequence of $2n - 1$ characters that does not contain any bigram or trigram as a substring.
- If $n = 1$, any character works.
- Otherwise, build a graph!
- Make a node $ab$ if the bigram $ab$ is not in the list of bigrams.
- Make an edge from $ab$ to $bc$ if the trigram $abc$ is not in the list.
- Find a path containing $2n - 2$ nodes in this graph.
- If there is a cycle, you can make a path of any length by repeating it.
- Otherwise, the graph is a directed acyclic graph, so you can find the longest path in it in $O(V + E)$ time, where $V$ is at most $26^2$ and $E$ is at most $26^3$. 
For a fixed partitioning scheme, always pick $x$ to be the most frequent value. So cost = length of array minus sum of largest frequencies.

Define $dp(i, j) =$ sum of frequencies for each subarray of the first $i$ elements into $j$ subarrays AND $a_i$ is the value we are using for the $j$th subarray

- Can use the second condition because there always exists a solution with the right endpoint of every subarray equal to the most frequent in that subarray
At $a_i$, we have two cases:

- **Start a new subarray for the $i$th element:**
  - $dp(i, j) = 1 + \max(dp(i', j - 1))$ for all $i' < i$.

- **Extend some subarray:**
  - $dp(i, j) = 1 + dp(prv[a_i], j)$ where $prv[x]$ is the most recent index in which we saw $a_i$

- Answer = $N - \max(dp(i, j))$ for all $i$ and $j$

- This runs in $O(NK)$