Brunhilda's Birthday (Spoiler)

Let $d(n)$ denote the number of calls Wotan needs to end the game when n children are left and let $M = \{k_1, \ldots, k_m\}$ be the set of primes Wotan can choose from.

1 Dynamic Programming

For 20 points it is enough to evaluate the obvious formula

$$d(n) = 1 + \min_{k \in M} d\big((n - (n \bmod k)\big) \tag{1}$$

using dynamic programming. Then all queries can be answered by simple lookup.

To handle the case $d(n) = \infty$ it suffices to check whether

$$n \geq \operatorname{lcm}(k_1, \ldots, k_m) = \prod_{k \in M} k$$

(since if n is not divisible by k after calling k less children will be over, but if n is at least the product p of all numbers Wotan can call, after any call at least p children will remain) or to simply set $d(n) = \infty$ for $n \neq 0$ before evaluating the above formula. Runtime is $\Theta(mn + Q)$ in both cases; an implementation can be found in file `brunhilda_trivial.cpp`.

2 Greedy Approach

Let us denote the *predecessor*, i.e. the number of children that are left after Wotan made a perfect call, of n as $\pi(n)$. If there are multiple solutions, let $\pi(n)$ be the minimum of this numbers. The main point of the solution is the following

Proposition 1. *Wotan can call the numbers greedily, i.e.* $\pi(n) = \min_{k \in M}(n - (n \bmod k))$.

This fact is—once stated—quite obvious, but it can be established rigorously using the following

Lemma 2. *π and d are both monotonically increasing in n.*

Proof. We show this for any interval $[0..N]$ by induction on N. Without loss of generality let us assume that $d(N) < \infty$. The case $N = 0$ is trivial. Otherwise we have $\pi(N) \leq N - 1$ as stated above and thus $\pi(N) = \min_{k \in M}\big(N - (N \bmod k)\big)$. Since $(N-1) \bmod k \geq (N \bmod k) - 1$ for any N, k we have $\pi(N) \geq \pi(N-1)$. Thus

$$d(N) = d(\pi(N)) + 1 \geq d(\pi(N-1)) + 1 \geq d(N-1)$$

because of the monoticity of d in $[0..\pi(N)] \subseteq [0..N-1]$. □

To show that this suffices for subtask 2 we need to establish an upper bound for $d(n)$. For simplicity let $k_{\max}$ denote $\max M$.

Lemma 3. *Let $n = n'k_{\max}$ and $d(n) < \infty$. Then $d(n) \leq 2n'$.*

Proof. We use induction on n'. Again there is nothing to show for $n' = 0$. For $n' \geq 1$ we have $\pi(n) < n$ by assumption and thus

$$\pi(\pi(n)) \leq \pi(n-1) \leq (n-1)-\big((n-1) \bmod k_{\max}\big) = (n-1)-(k_{\max}-1) = n-k_{\max} = (n'-1)k_{\max}$$

by monotocity of π. Using the monoticity of d we get thus $d(n) = d(\pi(\pi(n))) + 2 \leq d((n'-1)k_{\max}) + 2 = 2(n'-1) + 2 = 2n'$. $\square$

Once again using monoticity we get

Corollary 4. *If $d(n) < \infty$, then $d(n) \leq \left\lceil \frac{2n}{k_{\max}} \right\rceil$. Especially we have $d(n) = O(n/m)$.*

Using the prime number theorem stating that the n^{th} prime is asymptotically as big as $n \ln n$ one can decrease this bound further to $O\big(n/(m \log m)\big)$, however, this is not required directly for the solution.

Since we can calculate $\pi(n)$ primitively in $O(m)$ we can answer one query in $O(m + n)$ time. This suffices for subtasks 1 and 2 and is implemented in `brunhilda_singlequery.cpp`.

3 DP over inverse function

Let $d^{(-1)}$ denote the *inverse function* of d, i.e.

$$d^{(-1)}(k) = \max\{n : d(n) \leq k\}. \tag{2}$$

Since $\pi(k + k_{\max}) > k$ and thus $d(k + k_{\max}) > d(k)$, we have

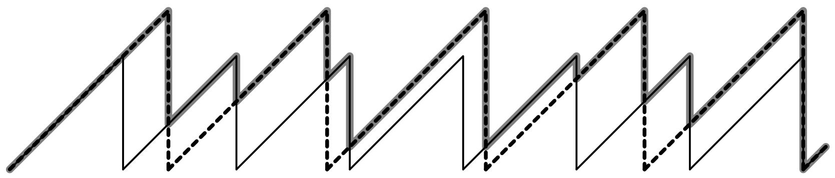
$$d^{(-1)}(k + 1) \leq d^{(-1)}(k) + k_{\max}. \tag{3}$$

Thus having calculated $d^{(-1)}(x)$ for $x \in [1..k]$ one can calculate $d^{(-1)}(k + 1)$ using binary search in the interval $[d^{(-1)}(k), d^{(-1)}(k) + k_{\max}]$. It further suffices to check for given x in this interval whether $\pi(x) \leq d^{(-1)}(k)$ (instead of really calculating $d(x)$, which would be too slow). So one can calculate this function for every needed value of k in time $O(n + m)$ (here the additional log-factor mentioned before comes in handy).

With this function one can answer any query in logarithmic time using binary search or simply fill an array $d[1..n]$ of all values in time $O(n)$ and then answer queries in $O(1)$. Both suffices to get full score; an implementation of the second technique can be found in `brunhilda_alternative.cpp`.

4 Model solution: Fast evaluation of the predecessor function

Instead of minimizing $\pi(n)$ we can simply maximize the term we subtract (since n is fixed), let's call it $\mu(n, k) = n \bmod k$. If we plot some those functions for variable n and some $k \in M$ the image consists of a set of straight lines of slope 1 and "breaks".

 (4)

Thus for $\mu^*(n) := \max_{k \in M} \mu(n, k)$ we get the same simple characteristic. If we plot all those functions—both μ and μ^*—and scan through this image from right to left the optimal k can only change when a break occurs. Thus if we evaluate μ^* at all those breaks of all the μ-functions we can simply fill in the rest of them by subtracting one from the next one at the right.

For any break point n, we have that $\mu^*(n-1)$ is $k-1$ for some $k \in M$. Thus to initialize our array $M[1..n]$ of values of μ^* it suffices to set $M[ak-1] = k-1$ for any a and increasings $k \in M$. This needs

$$\sum_{k \in M} \frac{n}{k} = n \sum_{k \in M} \frac{1}{k} \leq n \sum_{i=1}^{m} \frac{1}{k} = nH_m = O(n \log m)$$

steps (messing with analytic number theory one can reduce this bound further to $O(n \log \log m)$) but with really good constant factor.

Afterwards one can calculate $\pi(n)$ on the fly in constant time and thus use the simple DP approach from the beginning to get full score. This is implemented in file `brunhilda.cpp`.

An implementation using a priority queue to evaluate μ^* using the same ideas above is expected to get something around full score, too, depending on the data structure used (**set**/**priority_queue**/segment tree). A program featuring the STL **priority_queue**, which also gets full score, can be found in file `brunhilda_pq.cpp`.