1 Introdução

Esse problema é o problema C do *AtCoder Begginer Contest 125*, da plataforma de programação competitiva AtCoder. Disponível em: `https://atcoder.jp/contests/abc125/tasks/abc125_c`.

2 Solução: código em C++

```cpp
#include <bits/stdc++.h>
using namespace std;

/*
left[i] = gcd(a[0], ..., a[i-1])
right[i] = gcd(a[i + 1], ..., a[n - 1])
*/

int gcd(int a, int b){
    int m, M;
    while (a != 0 && b != 0){
        m = min(a,b);
        M = max(a,b);
            b = M % m;
            a = m;
    }
        return max(a, b);
}

void solve(){
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++){
        cin >> a[i];
    }
    int left[n], right[n];
    left[0] = 0; right[n - 1] = 0;
    for (int i = 1; i < n; i++){
        left[i] = gcd(left[i - 1], a[i - 1]);
    }
    for (int i = n - 2; i >= 0; i--){
        right[i] = gcd(right[i + 1], a[i + 1]);
    }

    int ans = 0;
    for (int i = 0; i < n; i++){
        ans = max(ans, gcd(left[i], right[i]));
    }
    cout << ans;
}
```

3 Demonstração

3.1 Lemas importantes sobre *gcd*

Teorema 3.1. *A função* $\gcd : \mathbb{N} \to \mathbb{N}$ *é associativa:*

$$\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c)$$

Ademais, ela é simétrica em relação a seus argumentos: $\gcd(a, b) = \gcd(b, a)$.

Prova. Pelo Teorema Fundamental da Aritmética, temos

$$a = \prod_{1 \le i \le n} p_i^{\alpha_i}$$
$$b = \prod_{1 \le i \le n} p_i^{\beta_i}$$
$$c = \prod_{1 \le i \le n} p_i^{\gamma_i}$$

em que $p_1 < \ldots, < p_n$ são os n-primeiros primos e um dos três expoentes $\alpha_n, \beta_n, \gamma_n$ é diferente de 0. Assim o lema segue da **associatividade da função min**:

$$\gcd(a, \gcd(b, c)) = \prod_{1 \le i \le n} p_i^{\min(\alpha_i, \min(\beta_i, \gamma_i))} = \prod_{1 \le i \le n} p_i^{\min(\min(\alpha_i, \beta_i), \gamma_i)} = \gcd(\gcd(a, b), c)$$

A associatividade da função min, por sua vez, pode ser provada analisando caso a caso dentre as possíveis ordenação de três números, e portanto sua demonstração é omitida aqui. □

Assim, podemos definir, para todo $k \ge 2$, a função $\gcd_k : \mathbb{N}^k \to \mathbb{N}$ definida recursivamente por

Definição 3.1.

$$\gcd_{k+1}(a_1, \ldots, a_{k+1}) = \gcd(\gcd_k(a_1, \ldots, a_k), a_{k+1})$$

a qual denotaremos apenas por gcd, ficando a quantidade de parâmetros implícita.

Como corolário, vale a seguinte generalização da associatividade:

Corolário 3.1.1. *Para todos* $1 \le i < n$*:*

$$\gcd_n(a_1, \ldots, a_n) = \gcd(\gcd_i(a_1, \ldots, a_i),\ \gcd_{n-i}(a_{i+1}, \ldots, a_n))$$

em que definimos $\gcd_1(a) = a$.

Prova. Provamos por indução sobre $n - i$. O caso $n - i = 1$ segue da própria **definição** de $\gcd_n$. Além disso,

$$\begin{aligned}
\gcd(\gcd_i(a_1, \ldots, a_i), \gcd_{n-i}(a_{i+1}, \ldots, a_{n-i})) &\overset{\text{Definição 3.1}}{=} \gcd(\gcd(\gcd_{i-1}(a_1, \ldots, a_{i-1}), a_i),\ \gcd_{n-i}(a_{i+1}, \ldots, a_n)) \\
&\overset{\text{Teorema 3.1}}{=} \gcd(\gcd_{i-1}(a_1, \ldots, a_{i-1}),\ \gcd(a_i, \gcd_{n-i}(a_{i+1}, \ldots, a_n))) \\
&= \gcd(\gcd_{i-1}(a_1, \ldots, a_{i-1}),\ \gcd(\gcd_{n-i}(a_{i+1}, \ldots, a_n),\ a_i)) \\
&\overset{\text{Definição 3.1}}{=} \gcd(\gcd_{i-1}(a_1, \ldots, a_{i-1}),\ \gcd_{n-i+1}(a_{i+1}, \ldots, a_n, a_i))
\end{aligned}$$

porém, como a ordem dos argumentos de $\gcd_k$ não importa, para qualquer k natural, então a expressão acima é simplesmente $\gcd(\gcd_{i-1}(a_1, \ldots, a_{i-1}),\ \gcd_{n-i+1}(a_i, \ldots, a_n))$. Ou seja, usando a hipótese de indução

$$\gcd_n(a_1, \ldots, a_n) = \gcd(\gcd_i(a_1, \ldots, a_i),\ \gcd_{n-i}(a_{i+1}, \ldots, a_n)) = \gcd(\gcd_{i-1}(a_1, \ldots, a_{i-1}),\ \gcd_{n-i+1}(a_i, \ldots, a_n))$$

provando o corolário por indução. □

Lema 3.1. *Para todos* $a, b \in \mathbb{N}$*, se* $b \ne 0$ *e* $a \equiv r \mod b$ *então*

$$\gcd(a, b) = \gcd(r, b)$$

Teorema 3.2. *O seguinte algoritmo para calcular* gcd *de dois números*

```
int gcd(int a, int b){
    while (a != 0 and b != 0){
        int m = min(a,b);
        int M = max(a, b);
        a = M % m;
        b = m;
    }
    return max(a, b);
}
```

termina com o resultado correto em $O(\log \min(a, b))$.

Prova: ***Complexidade.*** Seja a_i, b_i os valores das variáveis após a i-ésima iteração, e $a_0 = a, b_0 = b$ seus valores iniciais.

A partir da primeira iteração temos como invariante de loop que $a_i \leq b_i$, uma vez que $a_i \leftarrow M \mod m \leq m$ e $b_i \leftarrow m$. Assim, podemos supor sem perda de generalidade que $a \leq b$ inicialmente (do contrário, basta 1 iteração para valer tal desigualdade).

Após uma iteração temos $a_{i+1} = b_i \bmod a_i$ e $b_{i+1} = a_i$, e logo, após duas iterações temos que

$$a_{i+2} = a_i \bmod (b_i \bmod a_i) \tag{1}$$

$$b_{i+2} = b_i \bmod a_i \tag{2}$$

Seja $b_i = q_i a_i + r_i$. Então temos dois casos:

Caso 1: Se $r_i \leq \frac{a_i}{2}$ então

$$a_{i+2} = a_i \mod r_i < r_i \leq \frac{a_i}{2}$$

Caso 2: Se $r_i > \frac{a_i}{2}$, então

$$a_{i+2} = a_i \mod r_i = a_i - r_i < a_i - \frac{a_i}{2} = \frac{a_i}{2}$$

De qualquer forma temos $a_{i+2} \leq 2^{-1} a_i$. Logo, por indução

$$a_{2i} \leq 2^{-i} a_0$$

Como a_{2i} é um inteiro não-negativo, temos que $i > \log_2(a_0)$ implica que $a_{2i} < 1$, e portanto $a_{2i} = 0$, o que faz que o algoritmo termine. Como supomos sem perda de generalidade $a_0 \leq b_0$, isso mostra que o algoritmo termina em no máximo $2(\log_2(a_0) + 1) = 2(\log_2(\min(a, b)) + 1)$ iterações, isto é, possui complexidade $O(\log \min(a, b))$. □

Prova: ***Corretude.*** Precisamos mostrar apenas a seguinte invariante de loop: após i-ésima iteração

$$\gcd(a_i, b_i) = \gcd(a, b)$$

O caso inicial $i = 0$ segue da definição dos termos iniciais da sequência, pois $a_0 := a$ e $b_0 := b$. Suponha, então, que existe $i > 0$ tal que a invariante de loop segue válida para todo $0 \leq j \leq i$. Conforme mostramos na parte anterior, e pela condição do loop while, temos $0 < a_i \leq b_i$, e portanto

$$\gcd(a_{i+1}, b_{i+1}) = \gcd(b_i \bmod a_i, a_i) = \gcd(b_i, a_i) = \gcd(a_i, b_i)$$

pelo Lema 3.1. Pela parte anterior da demonstração, após a iteração final i_0 temos $a_{i_0} = 0$ ou $b_{i_0} = 0$, e portanto

$$\gcd(a, b) \overset{\text{invariante}}{=} \gcd(a_{i_0}, b_{i_0}) = \max(a_{i_0}, b_{i_0})$$

provando a corretude do algoritmo. □

3.2 Demonstrando o algoritmo

Teorema 3.3. *O algoritmo da Seção 2 fornece a resposta correta em $O(n \log M)$, onde $M = \max(a[0], \ldots, a[n-1])$.*

Prova. Temos que a resposta que deve ser fornecida, dado um vetor $a[]$ de n inteiros positivos, é

$$S = \max_{b \in \mathbb{Z}_{\geq 0}} \max_{1 \leq i \leq n} \gcd(a[0], \ldots, a[i-1], b, a[i+1], \ldots, a[n-1])$$

Porém, temos que para todo $b \geq 0$ inteiro:

$$\begin{aligned}
\gcd(a[0], \ldots, a[i-1], b, a[i+1], \ldots, a[n-1]) &= \gcd(a[0], \ldots, a[i-1], a[i+1], \ldots, a[n-1], b) \\
&\overset{\text{Definição 3.1}}{=} \gcd(\gcd(a[0], \ldots, a[i-1], a[i+1], \ldots, a[n-1]),\ b) \\
&\leq \gcd(a[0], \ldots, a[i-1], a[i+1], \ldots, a[n-1]) \\
&\overset{\text{Corolário 3.1.1}}{=} \gcd(\gcd(a[0], \ldots, a[i-1]),\ \gcd(a[i+1], \ldots, a[n-1]))
\end{aligned}$$

com **caso de igualdade quando, por exemplo**, $b = \gcd(a[0], \ldots, a[i-1], a[i+1], \ldots, a[n-1])$.

Assim, podemos definir os seguintes vetores $L[n]$, $R[n]$:

$$\begin{aligned}
L[0] &= 0 \\
L[i+1] &= \gcd(L[i], a[i])
\end{aligned}$$

$$\begin{aligned}
R[n-1] &= 0 \\
R[i-1] &= \gcd(R[i], a[i])
\end{aligned}$$

e pela associatividade (Teorema 3.1) teremos

$$\begin{aligned} L[i] &= \gcd(a[0], \ldots, a[i-1]) \\ R[i] &= \gcd(a[n-1], \ldots, a[n-i]) \end{aligned}$$

Ademais, novamente pelo Teorema 3.1, podemos calcular ambos os vetor em $O(n) \cdot O(\gcd) = O(n \log M)$ onde $M = \max(a[0], \ldots, a[n-1])$.

Por fim, teremos que

$$\begin{aligned} S &= \max_{0<i<n-1} \gcd(\gcd(a[0], \ldots, a[i-1]),\ \gcd(a[i+1], \ldots, a[n-1])) \\ &= \max_{0<i<n-1} \gcd(L[i], R[n-i]) \end{aligned}$$

o que novamente pode ser calculado em $O(n \log M)$.

Portanto, o código da Seção 2, que é exatamente a implementação do algoritmo descrito, fornece uma solução correta em $O(n \log M)$. $\square$