OJ 12879

Golf Bot

Prof. Edson Alves – UnB/FGA

Problema

Do you like golf? I hate it. I hate golf so much that I decided to build the ultimate golf robot, a robot that will never miss a shot. I simply place it over the ball, choose the right direction and distance and, flawlessly, it will strike the ball across the air and into the hole. Golf will never be played again.

Unfortunately, it doesn't work as planned. So, here I am, standing in the green and preparing my first strike when I realize that the distance-selector knob built-in doesn't have all the distance options! Not everything is lost, as I have 2 shots.

Given my current robot, how many holes will I be able to complete in 2 strokes or less? The ball must be always on the right line between the tee and the hole. It isn't allowed to overstep it and come back.

Entrada e saída

Input

The input file contains several test cases, each of them as described below.

The first line has one integer: N, the number of different distances the Golf Bot can shoot. Each of the following N lines has one integer, k_i, the distance marked in position i of the knob.

Next line has one integer: M, the number of holes in this course. Each of the following M lines has one integer, d_j, the distance from Golf Bot to hole j.

Constraints:

- $1 \leq N, M \leq 200000$
- $1 \leq k_i, d_j \leq 200000$

Entrada e saída

Output

For each test case, you should output a single integer, the number of holes Golf Bot will be able to complete. Golf Bot cannot shoot over a hole on purpose and then shoot backwards.

Sample Output Explanation

Golf Bot can shoot 3 different distances (1, 3 and 5) and there are 6 holes in this course at distances 2, 4, 5, 7, 8 and 9. Golf Bot will be able to put the ball in 4 of these:

- The 1st hole, at distance 2, can be reached by striking two times a distance of 1.
- The 2nd hole, at distance 4, can be reached by striking with strength 3 and then strength 1 (or vice-versa).
- The 3rd hole can be reached with just one stroke of strength 5.
- The 5th hole can be reached with two strikes of strengths 3 and 5.

Holes 4 and 6 can never be reached.

Exemplo de entradas e saídas

Sample Input

```
3
1
3
5
6
2
4
5
7
8
9
```

Sample Output

```
4
```

Solução $O(K \log K + M)$

- Para cada distância d_j, é preciso verificar se é possível atingí-la com uma ou duas tacadas
- Com uma tacada basta verificar se d_j é igual a algum dos k_i, o que pode ser feito em $O(1)$ com uma tabela *hash*
- Já para duas tacadas é a verificação pode ser feita em $O(N)$, também usando tabelas *hash*
- Contudo, este algoritmo teria complexidade $O(NM)$, o que resultaria em um veredito TLE, uma vez que $N, M \leq 2 \times 10^5$
- A transformada de Fourier permite resolver este problema com complexidade $O(K \log K + M)$, onde $K = \max\{k_1, k_2, \ldots, k_N\}$

Solução $O(K \log K + M)$

- Seja $p(x) = a_0 + a_1x + a_2x^2 + \ldots$ um polinômio tal que $a_0 = 1$ e $a_r = 1$, se $r > 0$ e existe ao menos um $k_i = r$, ou $a_i = 0$, caso contrário
- Para o exemplo de entrada, $p(x) = 1 + x + x^3 + x^5$
- Uma vez que $1 = x^0$, cada monômio de $p(x)$ representa uma tacada possível (ou mesmo não dar tacada, no caso de x^0) onde o grau é a distância atingida pelo golpe
- As distâncias possíveis de serem atingidas com duas tacadas são dadas pelos monômios não nulos do polinômio $q(x) = p^2(x)$, pois

  $$q(x) = p^2(x) = (a_0 + a_1x + a_2x^2 + \ldots)(a_0 + a_1x + a_2x^2 + \ldots)$$

  e a distributividade vai associar cada par de monômios possível
- Logo, para cada d_j basta verificar se a_j é igual a zero ou não
- O produto $p^2(x)$ pode ser obtido com a FFT em $O(K \log K)$, permitindo checar cada d_j em $O(1)$

Solução $O(K \log K + M)$

```cpp
#include <bits/stdc++.h>

using namespace std;

const double PI { acos(-1.0) }, EPS { 1e-6 };

void fft(vector<complex<double>>& xs, bool invert = false)
{
    int N = (int) xs.size();

    if (N == 1)
        return;

    vector<complex<double>> es(N/2), os(N/2);

    for (int i = 0; i < N/2; ++i)
        es[i] = xs[2*i];

    for (int i = 0; i < N/2; ++i)
        os[i] = xs[2*i + 1];
```

Solução $O(K \log K + M)$

```
    fft(es, invert);
    fft(os, invert);

    auto signal = (invert ? 1 : -1);
    auto theta = 2 * signal * PI / N;
    complex<double> S { 1 }, S1 { cos(theta), sin(theta) };

    for (int i = 0; i < N/2; ++i)
    {
        xs[i] = (es[i] + S * os[i]);
        xs[i] /= (invert ? 2 : 1);

        xs[i + N/2] = (es[i] - S * os[i]);
        xs[i + N/2] /= (invert ? 2 : 1);

        S *= S1;
    }
}
```

Solução $O(K \log K + M)$

```cpp
int solve(const vector<int>& ks, const vector<int>& ds)
{
    auto K = *max_element(ks.begin(), ks.end());
    int size = 1;

    while (size < K + 1)
        size *= 2;

    size *= 2;

    vector<complex<double>> xs(size, 0);

    for (auto k : ks)
        xs[k] = 1;

    xs[0] = 1;
    fft(xs);

    for (int i = 0; i < size; ++i)
        xs[i] *= xs[i];
```

Solução $O(K \log K + M)$

```
    fft(xs, true);

    int ans = 0;

    for (auto d : ds)
        ans += (d < size and fabs(xs[d].real()) > EPS ? 1 : 0);

    return ans;
}

int main()
{
    ios::sync_with_stdio(false);

    int N;

    while (cin >> N)
    {
        vector<int> ks(N);
```

Solução $O(K \log K + M)$

```
        for (int i = 0; i < N; ++i)
            cin >> ks[i];

        int M;
        cin >> M;

        vector<int> ds(M);

        for (int i = 0; i < M; ++i)
            cin >> ds[i];

        auto ans = solve(ks, ds);

        cout << ans << '\n';
    }

    return 0;
}
```