Codeforces Round #488

Problema B: *Knights of a Polygonal Table*

Prof. Edson Alves – UnB/FGA

Problema

Unlike Knights of a Round Table, Knights of a Polygonal Table deprived of nobility and happy to kill each other. But each knight has some power and a knight can kill another knight if and only if his power is greater than the power of victim. However, even such a knight will torment his conscience, so he can kill no more than k other knights. Also, each knight has some number of coins. After a kill, a knight can pick up all victim's coins.

Now each knight ponders: how many coins he can have if only he kills other knights?

You should answer this question for each knight.

Entrada e saída

Input

The first line contains two integers n and k ($1 \le n \le 10^5, 0 \le k \le \min(n-1, 10)$) – the number of knights and the number k from the statement.

The second line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le 10^9$) – powers of the knights. All p_i are distinct.

The third line contains n integers $c_1, c_2, \ldots, c_n$ ($0 \le c_i \le 10^9$) – the number of coins each knight has.

Output

Print n integers – the maximum number of coins each knight can have it only he kills other knights.

Exemplo de entradas e saídas

Sample Input	Sample Output
4 2 4 5 9 7 1 2 11 33	1 3 46 36
5 1 1 2 3 4 5 1 2 3 4 5	1 3 5 7 9
1 0 2 3	3

Solução com complexidade $O(N \log N)$

- Uma abordagem quadrática, avaliando todos os demais cavaleiros para cada cavaleiro i leva ao TLE, uma vez que $n \leq 10^5$
- Assim, é preciso ordernar os cavaleiros para evitar o processamento desnecessário e reaproveitar ao máximo o que já foi computado para o próximo cavaleiro
- Primeiramente, os cavaleiros devem ser ordenados em ordem crescente por sua força
- É preciso guardar o índice de cada cavaleiro em relação à entrada, para que a saída fique na ordem correta
- Com esta ordenação, o cavaleiro i será capaz de derrotar todos os cavaleiros cujo índice j é menor do que i
- Para computar o ganho do cavaleiro, é preciso manter o registro das k maiores moedas disponíveis até então
- Uma fila com prioridades pode ser utilizada para alcançar tal fim

Solução AC com complexidade $O(N \log N)$

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

using namespace std;

struct Knight
{
    int p, c, idx;

    bool operator<(const Knight& k) const
    {
        return p < k.p;
    }
};

vector<long long> solve(vector<Knight>& ks, size_t K)
{
    vector<long long> ans(ks.size());
    priority_queue<int> coins;
    long long sum = 0;
```

Solução AC com complexidade $O(N \log N)$

```
21      sort(ks.begin(), ks.end());
22
23      for (auto& knight : ks)
24      {
25          ans[knight.idx] = (knight.c + sum);
26          coins.push(-knight.c);
27          sum += knight.c;
28
29          if (coins.size() > K)
30          {
31              auto coin = coins.top();
32              coins.pop();
33
34              sum += coin;
35          }
36      }
37
38      return ans;
39  }
```

Solução AC com complexidade $O(N \log N)$

```
41 int main()
42 {
43     ios::sync_with_stdio(false);
44
45     int n, k;
46     cin >> n >> k;
47
48     vector<int> ps(n), cs(n);
49
50     for (int i = 0; i < n; ++i)
51         cin >> ps[i];
52
53     for (int i = 0; i < n; ++i)
54         cin >> cs[i];
55
56     vector<Knight> ks(n);
57
58     for (int i = 0; i < n; ++i)
59         ks[i] = Knight { ps[i], cs[i], i };
```

Solução AC com complexidade $O(N \log N)$

```
61      auto ans = solve(ks, k);
62
63      for (int i = 0; i < n; ++i)
64          cout << ans[i] << (i + 1 == n ? "\n" : " ");
65
66      return 0;
67  }
```