

Codeforces Round #488

Problema B: *Knights of a Polygonal Table*

Prof. Edson Alves – UnB/FCTE

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.

Input

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

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

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

Output

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

Exemplo de entradas e saídas

Sample Input

```
4 2
4 5 9 7
1 2 11 33
```

```
5 1
1 2 3 4 5
1 2 3 4 5
```

```
1 0
2
3
```

Sample Output

```
1 3 46 36
```

```
1 3 5 7 9
```

```
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 ordenar 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 $O(N \log N)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct Knight
6 {
7     int p, c, idx;
8
9     bool operator<(const Knight& k) const
10    {
11        return p < k.p;
12    }
13 };
14
15 vector<long long> solve(vector<Knight>& ks, size_t K)
16 {
17     vector<long long> ans(ks.size());
18     priority_queue<int> coins;
19     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 }
```