Árvores

Heaps Binárias na STL: Problemas Resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

Timus 1306 – Sequence Median

Problema

Given a sequence of N nonnegative integers. Let's define the median of such sequence. If N is odd the median is the element with stands in the middle of the sequence after it is sorted. One may notice that in this case the median has position $(N+1)/2$ in sorted sequence if sequence elements are numbered starting with 1. If N is even then the median is the semi-sum of the two "middle" elements of sorted sequence. I.e. semi-sum of the elements in positions $N/2$ and $(N/2)+1$ of sorted sequence. But original sequence might be unsorted.

Your task is to write program to find the median of given sequence.

Entrada e saída

Input

The first line of input contains the only integer number N – the length of the sequence. Sequence itself follows in subsequent lines, one number in a line. The length of the sequence lies in the range from 1 to 250000. Each element of the sequence is a positive integer not greater than $2^{31} - 1$ inclusive.

Output

You should print the value of the median with exactly one digit after decimal point.

Exemplo de entradas e saídas

Sample Input

```
4
3
6
4
5
```

Sample Output

```
4.5
```

Soluções $O(N \log N)$ e MLE

- Este problema é conceitualmente simples, mas traz uma dificuldade adicional: o limite de memória é curto para o tamanho máximo da entrada (apenas 1 MB)
- A solução mais simples seria ordenar os elementos da sequência em um vetor e computar a média a partir dos índices indicados no problema, mas é necesssário mais de um 1 MB para armazenar todos os elementos
- Assim, soluções baseadas em `sort()` e `nth_element()` levam ao MLE
- Uma alternativa seria usar uma `priority_queue()` e armazenar apenas $N/2 + 1$ elementos, descartando os demais sempre que o tamanho da *heap* exceder este limite
- Esta solução ainda leva ao MLE, por conta do `vector` subjacente
- A solução portanto, é utilizar esta abordagem, porém utilizando uma implementação customizada da *max heap*, onde os elementos são armazenados em um vetor estático

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

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

using namespace std;

const int MAX { 125002 };

class Heap {
private:
    size_t N;
    unsigned int xs[MAX];

    size_t parent(int i) { return i/2; }
    size_t left(int i) { return 2*i; }
    size_t right(int i) { return 2*i + 1; }

public:
    Heap() : N(0) {}

    size_t size() const { return N; }

```

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

```cpp
    void insert(int x)
    {
        ++N;

        xs[N] = x;

        int i = N, p = parent(i);

        while (p and xs[p] < xs[i])
        {
            std::swap(xs[p], xs[i]);
            i = p;
            p = parent(i);
        }
    }

    unsigned int extract_max()
    {
        auto x = xs[1];
        std::swap(xs[1], xs[N]);
        --N;
```

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

```
        int i = 1, n = left(i) > N ? 0 : left(i);

        while (n)
        {
            auto r = right(i) > N ? 0 : right(i);

            if (r and xs[r] > xs[n])
                n = r;

            if (xs[i] < xs[n])
            {
                std::swap(xs[i], xs[n]);
                i = n;
                n = left(i) > N ? 0 : left(i);
            } else
                n = 0;
        }

        return x;
    }
};
```

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

```

int main()
{
    unsigned int N;
    scanf("%u", &N);

    Heap pq;

    for (size_t i = 0; i < N; ++i)
    {
        unsigned int x;
        scanf("%u", &x);

        pq.insert(x);

        if (pq.size() > N/2 + 1)
            pq.extract_max();
    }

    double ans = pq.extract_max();

```

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

```
    if (N % 2 == 0)
    {
        ans += pq.extract_max();
        ans /= 2.0;
    }

    printf("%.1f\n", ans);

    return 0;
}
```

Codeforces Round #488 – Problem B: Knights of a Polygonal Table

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

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

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

Output

Print n integers – the maximum number of coins each knight can have it 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 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;

    sort(ks.begin(), ks.end());
```

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

```

    for (auto& knight : ks)
    {
        ans[knight.idx] = (knight.c + sum);
        coins.push(-knight.c);
        sum += knight.c;

        if (coins.size() > K)
        {
            auto coin = coins.top();
            coins.pop();

            sum += coin;
        }
    }

    return ans;
}

```

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

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

    int n, k;
    cin >> n >> k;

    vector<int> ps(n), cs(n);

    for (int i = 0; i < n; ++i)
        cin >> ps[i];

    for (int i = 0; i < n; ++i)
        cin >> cs[i];

    vector<Knight> ks(n);

    for (int i = 0; i < n; ++i)
        ks[i] = Knight { ps[i], cs[i], i };

```

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

```
    auto ans = solve(ks, k);

    for (int i = 0; i < n; ++i)
        cout << ans[i] << (i + 1 == n ? "\n" : " ");

    return 0;
}
```

Codechef – Restaurant Rating

Problema

Chef has opened up a new restaurant. Like every other restaurant, critics critique this place. The Chef wants to gather as much positive publicity as he can. Also, he is very aware of the fact that people generally do not tend to go through all the reviews. So he picks out the positive reviews and posts it on the website of the restaurant. A review is represented by an integer which is the overall rating of the restaurant as calculated by that particular review. A review is considered to be positive if it is among the top one-third of the total reviews when they are sorted by their rating. For example, suppose the ratings given by 8 different reviews are as follows:

2 8 3 1 6 4 5 7

Then the top one-third reviews will be 8 and 7. Note that we considered one-third to be 8/3=2 top reviews according to integer division. (see Notes)

Problema

So here is what the Chef wants from you: Given the reviews (ratings) by different critics, the Chef needs you to tell him what is the minimum rating that his website will be displaying. For example in the above case, the minimum rating that will be displayed is 7. Also, different critics keep reviewing the restaurant continuosly. So the new reviews keep coming in. The Chef wants your website ratings to be up-to-date. So you should keep updating the ratings there. At any point of time, the Chef might want to know the minimum rating being displayed. You'll have to answer that query. An interesting thing to note here is that a review might be in the website for some time and get knocked out later because of new better reviews and vice-versa.

Notes: To be precise, the number of reviews displayed website will be `floor(n/3)`, where n denotes the current number of all reviews.

Entrada e saída

Input

First line of the input file consists of a single integer N, the number of operations to follow. The next N lines contain one operation each on a single line. An operation can be of 2 types:

`1 x` : Add a review with rating 'x' to the exisiting list of reviews (x is an integer)

`2` : Report the current minimum rating on the website

Output

For every test case, output a single integer each for every operation of type `2` mentioned above. If no review qualifies as a positive review, print "`No reviews yet`".

Constraints

$1 \leq N \leq 250000$

$1 \leq x \leq 1000000000$

Exemplo de entradas e saídas

Sample Input

```
10
1 1
1 7
2
1 9
1 21
1 8
1 5
2
1 9
2
```

Sample Output

```
No reviews yet
9
9
```

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

- Este é um problema dinâmico, onde é preciso manter o registro de um terço dos melhores *ratings* obtidos até o momento
- Para tal serão utilizadas duas *heap* binárias: `lo`, uma *max heap* que manterá os piores *ratings* (2/3 do total); e `hi`, uma *min heap* que manterá os melhores *ratings* (1/3 do total)
- Para manter o invariante destas *heaps*, é preciso, ao processar comandos do tipo 1, inserir os elementos em `lo` e atualizar o total R de *ratings* já inseridos
- Se o tamanho de `hi` for inferior a $\lfloor R/3 \rfloor$, é preciso passar o topo de `lo` para `hi`

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

- Um último cuidado é necessário: uma inserção pode fazer com que `lo` tenha elementos maiores do que `hi`, violando o invariante
- Assim, se `hi` não estiver vazia, enquanto o topo x de `lo` for maior do que o topo y de `hi`, eles elementos devem passar de uma *heap* para a outra
- Como cada elemento será inserido, no máximo, 2 vezes em `lo`, a complexidade desta solução é $O(R \log R)$, onde R é o número de *ratings* obtidos

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

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

using namespace std;

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

    int N;
    cin >> N;

    priority_queue<int> lo;
    priority_queue<int, vector<int>, greater<int>> hi;

    size_t total = 0;

    while (N--)
    {
        int cmd, x;
        cin >> cmd;

```

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

```
        switch (cmd) {
        case 1:
            cin >> x;

            lo.push(x);
            ++total;

            if (hi.size() < total / 3)
            {
                hi.push(lo.top());
                lo.pop();
            }

            while (not hi.empty() and lo.top() > hi.top())
            {
                auto x = lo.top();
                auto y = hi.top();

                lo.pop();
                hi.pop();

```

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

```
                hi.push(x);
                lo.push(y);
            }

            break;

        default:
            if (hi.empty())
                cout << "No reviews yet\n";
            else
                cout << hi.top() << '\n';
        }
    }

    return 0;
}
```

Referências

1. Timus 1306 – Sequence Median
2. Codeforces Round #488 – Problem B: Knights of a Polygonal Table
3. Codechef – Restaurant Rating