Maratona SBC de Programação 2025 – Subregional

Problema M: Muralhas reforçadas

Prof. Edson Alves – UnB/FCTE

Problema

Devido à sua localização privilegiada, com um relevo bem favorável, geralmente apenas muralhas frontais eram necessárias para a proteção de cidades medievais em Minas Gerais. Ainda hoje, é possível encontrar vestígios dessas construções ao admirar o belo horizonte da região.

Cada uma dessas muralhas era composta por um número de segmentos consecutivos, cada um com uma altura inicial medida em unidades, correspondendo ao número de blocos usados para construí-lo.

De tempos em tempos, os construtores reforçavam as muralhas escolhendo um segmento e empilhando blocos extra em um padrão de escada: o segmento escolhido recebia K blocos adicionais, o anterior $K-1$, e assim sucessivamente até que apenas 1 bloco fosse adicionado ou que não houvesse mais segmentos à esquerda.

A solidez da defesa era determinada pela altura do segmento mais baixo da muralha.

Dada a descrição inicial de uma muralha, seu objetivo é determinar qual é a maior altura mínima possível após a aplicação de um único reforço.

Entrada e saída

Entrada

A primeira linha contém dois inteiros N $(1 \leq N \leq 10^5)$, o número de segmentos da muralha, e K $(1 \leq K \leq N)$, o número de blocos adicionados ao segmento escolhido.

A segunda linha contém N inteiros $x_1, x_2, \ldots, x_N$ $(1 \leq x_i \leq 10^9)$, representando as alturas iniciais dos segmentos.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro: a maior altura mínima possível da muralha após um único reforço.

Exemplos de entradas e saídas

Sample Input	Sample Output
5 5 5 4 3 2 1	6
6 1 3 3 1 3 3 3	2
5 5 3 4 7 8 7	7

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

- Uma possível solução consiste em aplicar o reforço a partir de cada um dos elementos a_i e avaliar, em cada aplicação, o menor elemento m_i
- A resposta do problema será o mínimo entre os elementos $m_1, m_2, \ldots, m_N$
- Porém, a aplicação direta dos reforços, seguida de uma identificação *naive* do elemento mínimo, leva a uma solução com complexidade $O(N^2)$ e ao veredito TLE
- É preciso, portanto, realizar duas tarefas de forma eficiente para reduz a complexidade:
  1. aplicar o reforço a partir do elemento a_i
  2. identificar o menor elemento m_i da muralha após a aplicação
- Em relação à tarefa 2, ela pode ser dividida em 3 subtarefas:
  (a) Identificação do mínimo r_i dentre os elementos a_j, com $i \in [1, i-K]$
  (b) Identificação do mínimo s_i dentre os elementos $1 + a_{i-K+1}, 2 + a_{i-K+2}, \ldots, K + a_i$
  (c) Identificação do mínimo t_i dentre os elementos a_k, com $k \in [i+1, N]$
- Nessas condições, $m_i = \min(r_i, s_i, t_i)$

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

- Se $[1, i-K]$ ou $[i+1, N]$ forem intervalos degenerados, teremos $r_i = \infty$ ou $t_i = \infty$, respectivamente
- Nos demais casos, os valores de s_i e t_i podem ser determinado em $O(1)$, desde que sejam pré-computados, em $O(N)$, dois vetores p e s, onde $p(i)$ e $s(i)$ correspondem aos mínimos dos prefixos $a[1..i]$ e dos sufixos $a[i..N]$, respectivamente
- Para determinar o mínimo do cenário (b), que envolve a aplicação do reforço
- Isto pode ser feito com o auxílio de um *Venice Set*, uma estrutura de dados que permite identificar o menor elemento armazenado, remover este elemento e adicionar igualmente um valor x em todos os elementos armazenados em $O(\log M)$, onde M é o número de elementos armazenados

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

- Para manter, no máximo, K elementos no conjunto de Veneza v, é preciso armazenar pares (y, i), onde y é o valor a ser armazenado e i o índice que esse valor ocupava no vetor a, onde $v = a_i + K$
- Se forem mantidos dois ponteiros L, R para a identificação da janela móvel de tamanho K onde o reforço será aplicado, os mínimos s_i em $[L, R]$ podem podem ser identificados da seguinte maneira:
  1. diminua os valores já armazenados em v em uma unidade. Esta ação aproveita boa parte do reforço anterior pois teremos, em v, ao menos os valores

$$a_{i-K+1}, 1 + a_{i-K+2}, \ldots, (K-1) + a_{i-1}$$

  2. o reforço em a_i fica completo ao adicionar $y_i = a_i + K$ em v
  3. os elementos excedentes em v devem ser removidos: basta remover os mínimos de v enquanto o segundo elemento do par for menor do que L
  4. realizado este ajuste, o menor elemento de v pode ser obtido em $O(\log M)$

Solução AC

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

using namespace std;

template<typename T>
struct VeniceSet
{
    multiset<pair<T, int>> s;
    T delta = 0;

    void add(T x, int i) { s.emplace(x + delta, i); }
    void update_all(T x) { delta -= x; }
    void pop() { s.erase(s.begin()); }

    auto min() const
    {
        auto [m, i] = *s.begin();
        return make_pair(m - delta, i);
    }
```

Solução AC

```
21      bool empty() const { return s.empty(); }
22      int size() const { return static_cast<int>(s.size()); }
23  };
24
25  auto solve(int N, int K, const vector<int>& as)
26  {
27      // Mínimos do prefixos de as
28      vector<int> p(N);
29      p[0] = as[0];
30
31      for (int i = 1; i < N; ++i)
32          p[i] = min(p[i - 1], as[i]);
33
34      // Mínimos do sufixos de as
35      vector<int> s(N);
36      s[N - 1] = as[N - 1];
37
38      for (int i = N - 2; i >= 0; --i)
39          s[i] = min(s[i + 1], as[i]);
```

Solução AC

```
41      VeniceSet<int> vs;
42      int ans = 0;
43
44      for (auto R = 0, L = R - K + 1; R < N; ++L, ++R) {
45          vs.update_all(-1);
46          vs.add(as[R] + K, R);
47
48          while (not vs.empty() and vs.min().second < L)
49              vs.pop();
50
51          auto [m, _] = vs.min();
52
53          if (L > 0)
54              m = min(m, p[L - 1]);
55
56          if (R < N - 1)
57              m = min(m, s[R + 1]);
58
59          ans = max(ans, m);
60      }
```

Solução AC

```
62      return ans;
63  }
64
65  int main()
66  {
67      ios::sync_with_stdio(false);
68
69      int N, K;
70      cin >> N >> K;
71
72      vector<int> as(N);
73
74      for (auto& a : as)
75          cin >> a;
76
77      cout << solve(N, K, as) << '\n';
78
79      return 0;
80  }
```