Paradigmas de Resolução de Problemas

Busca Completa – *Two pointers*: Problemas resolvidos

Prof. Edson Alves - UnB/FGA

2020

Sumário

Codeforces Round #171 – Problem B: Books

Problema

When Valera has got some free time, he goes to the library to read some books. Today he's got t free minutes to read. That's why Valera took n books in the library and for each book he estimated the time he is going to need to read it. Let's number the books by integers from 1 to n. Valera needs a_i minutes to read the i-th book.

Valera decided to choose an arbitrary book with number i and read the books one by one, starting from this book. In other words, he will first read book number i, then book number $i + 1$, then book number $i + 2$ and so on. He continues the process until he either runs out of the free time or finishes reading the n-th book. Valera reads each book up to the end, that is, he doesn't start reading the book if he doesn't have enough free time to finish reading it.

Print the maximum number of books Valera can read.

Entrada e saída

Input

The first line contains two integers n and t ($1 \leq n \leq 10^5; 1 \leq t \leq 10^9$) – the number of books and the number of free minutes Valera's got. The second line contains a sequence of n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^4$), where number a i shows the number of minutes that the boy needs to read the i-th book.

Output

Print a single integer – the maximum number of books Valera can read.

Exemplo de entradas e saídas

Sample Input

```
4 5
3 1 2 1

3 3
2 2 3
```

Sample Output

```
3

1
```

Solução com complexidade $O(N)$

- O problema consiste em avaliar, para cada $i \in [1, N]$, o maior número de livros que podem ser lidos começando no i-ésimo livro
- Uma solução de busca completa avaliaria cada i em $O(N)$, tendo portanto complexidade $O(N^2)$
- O uso de dois ponteiros reduz esta complexidade para $O(N)$
- O ponteiro L inicia no primeiro elemento, e deve ser incrementado sequenciamente até o último elemento, representando aqui o i do problema
- Já o ponteiro R também inicia no primeiro elemento, e a cada iteração ele será o maior dentre L e R, pois cada livro será avaliado uma única vez por R

Solução com complexidade $O(N)$

- Deve ser mantida uma variável t, inicialmente igual a T, que mantém o registro do tempo ainda disponível para leitura
- A cada iteração, enquanto R apontar para um elemento do vetor e houver tempo para ler o R-ésimo livro, R deve ser incrementado e t atualizado
- Ao fim de cada iteração o $R - L$ livros do intervalo $[L, R)$ podem ser lidos
- A variável t deve ser atualizada, acrescentado o tempo de leitura do L-ésimo livro, caso tenha sido decrementado de t previamente
- A resposta será o tamanho do maior dentre estes intervalos

Solução AC com complexidade $O(N)$

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

using namespace std;

int solve(int N, int T, const vector<int>& xs)
{
    auto ans = 0, L = 0, R = 0, t = T;

    while (L < N)
    {
        R = max(L, R);

        while (R < N and xs[R] <= t)
            t -= xs[R++];

        ans = max(ans, R - L);

        t = min(T, t + xs[L]);
        ++L;
    }

```

Solução AC com complexidade $O(N)$

```
22      return ans;
23  }
24
25  int main()
26  {
27      ios::sync_with_stdio(false);
28
29      int N, T;
30      cin >> N >> T;
31
32      vector<int> xs(N);
33
34      for (int i = 0; i < N; ++i)
35          cin >> xs[i];
36
37      auto ans = solve(N, T, xs);
38
39      cout << ans << '\n';
40
41      return 0;
42  }
```

AtCoder Beginner Contest 098 – Problem D: Xor Sum 2

Problema

There is an integer sequence A of length N.

Find the number of the pairs of integers l and r $(1 \leq l \leq r \leq N)$ that satisfy the following condition:

- $A_l \ xor \ A_{l+1} \ xor \ \ldots \ xor \ A_r = A_l + A_{l+1} + \ldots + A_r$

Here, xor denotes the bitwise exclusive OR.

Entrada e saída

Constraints

- $1 \leq N \leq 2 \times 10^5$
- $0 \leq A_i < 2^{20}$
- All values in input are integers.

Input

Input is given from Standard Input in the following format:

$$N$$
$$A_1 \ A_2 \ \ldots \ A_N$$

Output

Print the number of the pairs of integers l and r $(1 \leq l \leq r \leq N)$ that satisfy the condition.

Exemplo de entradas e saídas

Sample Input

```
4
2 5 4 6

9
0 0 0 0 0 0 0 0 0

19
885 8 1 128 83 32 256 206 639 16 4
128 689 32 8 64 885 969 1
```

Sample Output

```
5

45

37
```

Solução com complexidade $O(N)$

- Dados dois inteiros x e y, a igualdade $x\ xor\ y = x + y$ só é verdadeira se x e y não tiverem nenhum *bit* em comum
- Assim, um intervalo de índices $[l, r]$ atenderá o critério do problema somente se, para cada par de elementos A_i, A_j, com $i, j \in [l, r]$ e $i \neq j$, vale que $A_i\ and\ A_j = 0$ (isto é, A_i e A_j não tem *bits* em comum)
- Como existem $O(N^2)$ pares de índices (l, r), com $1 \leq l \leq r \leq N$, uma solução que verifique cada um destes pares teria um veredito TLE
- Porém, é possível resolver o problema em $O(N)$, por meio da técnica dos dois ponteiros
- Para isso, inicie o ponteiro L no primeiro elemento do vetor

Solução com complexidade $O(N)$

- Inicie com zero uma variável x, que conterá a disjunção (ou) dos elementos do intervalo $[L, R)$
- Para cada valor de L, inclua A_L em x
- O ponteiro R deve ser o maior dentre L e o próprio R
- Enquanto R apontar para um elemento do vetor e A_R não tiver *bits* em comum com x, inclua A_R em x e incremente R
- Ao final do processo, qualquer intervalo $[L, r]$, com $r \in [L, R)$, será um intervalo válido do problema
- Assim, a resposta deve ser acrescida em $R - L$
- Após a atualização da resposta, remova o elemento A_L de x e incremente L (observe que $R - L > 0$ em qualquer iteração)

Solução com complexidade $O(N)$

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

using namespace std;
using ll = long long;

ll solve(int N, const vector<int>& xs)
{
    ll ans = 0;
    auto L = 0, R = 0, x = 0;

    while (L < N)
    {
        R = max(L, R);

        while (R < N and (x & xs[R]) == 0)
        {
            x |= xs[R];
            ++R;
        }

        ans += (R - L);
```

Solução com complexidade $O(N)$

```
22
23          x &= ~xs[L];
24          ++L;
25      }
26
27      return ans;
28  }
29
30  int main()
31  {
32      ios::sync_with_stdio(false);
33
34      int N;
35      cin >> N;
36
37      vector<int> xs(N);
38
39      for (int i = 0; i < N; ++i)
40          cin >> xs[i];
41
42      auto ans = solve(N, xs);
```

Solução com complexidade $O(N)$

```
43
44      cout << ans << endl;
45
46      return 0;
47  }
```

SPOJ – Ada and Plants

Problema

Ada the Ladybug has grown many plants. She was trying to grow all plants with equal size. Now she is wondering about the biggest difference between heights of two plants which are near each other. Plants are near each other, if there are at most K plants between them.

Entrada e saída

Input

The first line contains T, the number of test-cases. The first line of each test-case will contain $N, K, 1 < N \leq 10^5, 0 \leq K \leq 10^5$ where N indicates number of plants. Next line will contain N integers $0 \leq hi \leq 10^9$ indicating height of i-th plant.

Sum of all N among all test-cases won't exceed 3×10^6.

Output

For each test-case, print exactly one number – the biggest difference of plants near each other (biggest $h_i - h_j$ such that $|i - j| - 1 \leq K$).

Exemplo de entradas e saídas

Sample Input

```
3
5 0
1 2 3 5 6
4 6
1 10 2 9
10 1
1 7 8 9 19 11 21 8 11 0
```

Sample Output

```
2
9
13
```

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

- Segundo o texto do problema, quaisquer duas plantas cujos índices estejam em um intervalo $[L, R)$, com $R - L = K + 2$, estão próximas o suficiente
- A cada intervalo $[L, R)$, a maior diferença ocorrerá entre os elementos com índices neste intervalo e que tenham a maior e a menor altura, respectivamente
- Logo, o problema pode ser resolvido usando a técnica de dois ponteiros com janela móvel (*sliding window*)
- Para obter, de forma eficiente, os valores da maior e da menor altura no intervalo, há duas formas
- Uma delas é manter as alturas dos elementos do intervalo em um `multiset`: os ponteiros `rbegin()` e `begin()` apontarão para os elementos desejados

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

- A inserção e remoção de um elemento no `multiset` é feita em $O(\log K)$
- Cada elemento será inserido ou removido do `multiset` no máximo uma vez, logo a solução terá complexidade $O(N \log K)$
- Observe que o valor de K pode exceder N, de modo que ele deve ser tratado com cuidado
- Também é possível resolver este problema em $O(N)$: basta usar a estratégia de dupla pilha para manter os valores do menor e do maior elemento do intervalo
- A implementação é mais longa do que a que utiliza o `multiset`, porém tem melhor tempo de execução e usa menos memória

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

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

using namespace std;

int solve(int N, int K, const vector<int>& xs)
{
    K = min(N - 1, K + 1);

    int L = 0, R;
    multiset<int> s;

    for (R = 0; R <= K; ++R)
        s.insert(xs[R]);

    int ans = *s.rbegin() - *s.begin();

    while (R < N)
    {
        auto it = s.find(xs[L]);
        s.erase(it);
        s.insert(xs[R]);
```

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

```
22
23          ans = max(ans, *s.rbegin() - *s.begin());
24          ++L;
25          ++R;
26      }
27
28      return ans;
29  }
30
31  int main()
32  {
33      ios::sync_with_stdio(false);
34
35      int T;
36      cin >> T;
37
38      while (T--)
39      {
40          int N, K;
41          cin >> N >> K;
42
```

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

```
43          vector<int> xs(N);
44
45          for (int i = 0; i < N; ++i)
46              cin >> xs[i];
47
48          auto ans = solve(N, K, xs);
49
50          cout << ans << endl;
51      }
52
53      return 0;
54  }
```

Solução AC com complexidade $O(N)$

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

using namespace std;
using ii = pair<int, int>;

const int oo { 1000000010 };

struct StackM {

    function<int(int, int)> op;
    stack<ii> in, out;

    void add(int x) { add(x, in); }

    int top()
    {
        if (in.empty())
            return out.top().second;
        else if (out.empty())
            return in.top().second;

```

Solução AC com complexidade $O(N)$

```
22          return op(in.top().second, out.top().second);
23      }
24
25      void pop()
26      {
27          move();
28          out.pop();
29      }
30
31      void move()
32      {
33          if (out.empty())
34          {
35              while (not in.empty())
36              {
37                  auto x = in.top().first;
38                  in.pop();
39                  add(x, out);
40              }
41          }
42      }
```

Solução AC com complexidade $O(N)$

```
43
44      void add(int x, stack<ii>& s)
45      {
46          int m = s.empty() ? x : op(s.top().second, x);
47          s.push(ii(x, m));
48      }
49  };
50
51  int solve(int N, int K, const vector<int>& xs)
52  {
53      K = min(N - 1, K + 1);
54
55      auto min_of = [](int x, int y) { return min(x, y); };
56      auto max_of = [](int x, int y) { return max(x, y); };
57
58      StackM smin { min_of, {}, {} };
59      StackM smax { max_of, {}, {} };
60
61      int L = 0, R;
62
```

Solução AC com complexidade $O(N)$

```
63      for (R = 0; R <= K; ++R)
64      {
65          smin.add(xs[R]);
66          smax.add(xs[R]);
67      }
68
69      int ans = smax.top() - smin.top();
70
71      while (R < N)
72      {
73          smin.pop();
74          smax.pop();
75
76          smin.add(xs[R]);
77          smax.add(xs[R]);
78
79          ++L;
80          ++R;
81
82          ans = max(ans, smax.top() - smin.top());
83      }
```

Solução AC com complexidade $O(N)$

```
84
85      return ans;
86  }
87
88  int main()
89  {
90      ios::sync_with_stdio(false);
91
92      int T;
93      cin >> T;
94
95      while (T--)
96      {
97          int N, K;
98          cin >> N >> K;
99
100         vector<int> xs(N);
101
102         for (int i = 0; i < N; ++i)
103             cin >> xs[i];
104
```

Solução AC com complexidade $O(N)$

```
105         auto ans = solve(N, K, xs);
106
107         cout << ans << endl;
108     }
109
110     return 0;
111 }
```

Codechef – Naruto and Rectangles

Problema

Given n non-negative integers $a_1, a_2, \ldots, a_n$, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of the line i is at (i, a_i) and $(i, 0)$.

Naruto wants to draw a rectangle using these lines. The base should be the x-axis, while the two sides should be any two lines from the above-given lines. The other lines will magically disappear. Note that he cannot move the lines. The only action done my Naruto is to draw the top line of the rectangle by connecting any two lines.

Now naruto wants to know the maximum possible area of rectangle he can get. Help him find it.

Entrada e saída

Input

- First line will contain T, number of testcases. Then the testcases follow.
- First line of each testcase contains n.
- Second line contains n space separated integers $a_1, a_2, \ldots, a_n$.

Output

For each testcase, output in a single line, the maximum area.

Constraints

- $1 \leq T \leq 10$
- $2 \leq N \leq 10^5$
- $2 \leq a_i \leq 10^3$

Exemplo de entradas e saídas

Sample Input

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

Sample Output

```
49
```

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

- Para cada par (i, j), com $j > i$, o retângulo de maior área que pode ser formado terá base $b = (j - i)$ e altura $h = \min(a_i, a_j)$
- Como há $O(N^2)$ pares deste tipo, uma solução que verifique todos eles tem complexidade $O(N^2)$ e, neste problema, tem veredito TLE
- Observe que, para um i fixo, basta verificar apenas dois índices: o elemento L mais à esquerda de i com $a_L \geq a_i$ e o elemento R mais à direita de i tal que $a_R \geq a_i$
- Estes dois elementos podem ser identificados, de forma eficiente, por meio de ordenação e dois ponteiros
- Primeiramente, monte um vetor de pares (a_i, i) e ordene este vetor em ordem não-crescente

Solução com complexidade $O(N)$

- Inicialize os ponteiros $L = \infty$ e $R = -\infty$, e a resposta com zero
- Para cada par do vetor ordenado, processe o par (L, i), se $L < i$, e o par (i, R), se $R > i$
- Após uma possível atualização da resposta, ajuste os ponteiros: $L = \min(L, i)$, $R = \max(R, i)$
- Observe que, com esta estratégia, a altura a_i do próximo elemento a ser processado será inferior à altura dos elementos já processados, de modo que ela pode ser combinada, de forma ótima, com qualquer um deles
- Por isso basta manter o registro, dentre eles, do que está mais à esquerda (L) e mais à direta (R) do vetor
- Esta solução tem complexidade $O(N \log N)$, por causa da ordenação

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

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

using namespace std;
using ii = pair<int, int>;

const int oo { 2000000010 };

int solve(int N, const vector<int>& xs)
{
    vector<ii> ps(N);

    for (int i = 0; i < N; ++i)
        ps[i] = ii(xs[i], i);

    sort(ps.begin(), ps.end(), greater<ii>());

    auto L = oo, R = -oo, ans = 0;

    for (auto p : ps)
    {
        auto x = p.first, i = p.second;
```

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

```
22
23          if (L < i)
24              ans = max(ans, (i - L)*min(x, xs[L]));
25
26          if (R > i)
27              ans = max(ans, (R - i)*min(x, xs[R]));
28
29          L = min(L, i);
30          R = max(R, i);
31      }
32
33     return ans;
34 }
35
36 int main()
37 {
38      ios::sync_with_stdio(false);
39
40      int T;
41      cin >> T;
42
```

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

```
43      while (T--)
44      {
45          int N;
46          cin >> N;
47
48          vector<int> xs(N);
49
50          for (int i = 0; i < N; ++i)
51              cin >> xs[i];
52
53          auto ans = solve(N, xs);
54
55          cout << ans << '\n';
56      }
57
58      return 0;
59  }
```

Referências

1. Codeforces Round #171 – Problem B: Books
2. AtCoder Beginner Contest 098 - Problem D: Xor Sum 2
3. SPOJ ADAPLANT – Ada and Plants
4. Codechef COW207 – Naruto and Rectangles