Árvore de Fenwick

Aplicações e variantes: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

UVA 12532 – Interval Product

Problema

It's normal to feel worried and tense the day before a programming contest. To relax, you went out for a drink with some friends in a nearby pub. To keep your mind sharp for the next day, you decided to play the following game. To start, your friends will give you a sequence of N integers $X_1, X_2, \ldots, X_N$. Then, there will be K rounds; at each round, your friends will issue a command, which can be:

- a change command, when your friends want to change one of the values in the sequence; or
- a product command, when your friends give you two values I, J and ask you if the product $X_I \times X_{I+1} \times \ldots \times X_{J-1} \times X_J$ is positive, negative or zero.

Problema

Since you are at a pub, it was decided that the penalty for a wrong answer is to drink a pint of beer. You are worried this could affect you negatively at the next day's contest, and you don't want to check if Ballmer's peak theory is correct. Fortunately, your friends gave you the right to use your notebook. Since you trust more your coding skills than your math, you decided to write a program to help you in the game.

Entrada e saída

Input

Each test case is described using several lines. The first line contains two integers N and K, indicating respectively the number of elements in the sequence and the number of rounds of the game ($1 \leq N, K \leq 10^5$). The second line contains N integers X_i that represent the initial values of the sequence ($-100 \leq X_i \leq 100$) for $i = 1, 2, \ldots, N$). Each of the next K lines describes a command and starts with an uppercase letter that is either '`C`' or '`P`'. If the letter is '`C`', the line describes a change command, and the letter is followed by two integers I and V indicating that X_I must receive the value V ($1 \leq I \leq N$ and $-100 \leq V \leq 100$). If the letter is '`P`', the line describes a product command, and the letter is followed by two integers I and J indicating that the product from X_I to X_J, inclusive must be calculated ($1 \leq I \leq J \leq N$). Within each test case there is at least one product command.

Entrada e saída

Output

For each test case output a line with a string representing the result of all the product commands in the test case. The i-th character of the string represents the result of the i-th product command. If the result of the command is positive the character must be '+' (plus); if the result is negative the character must be '-' (minus); if the result is zero the character must be '0' (zero)

Exemplo de entradas e saídas

Sample Input

```
4 6
-2 6 0 -1
C 1 10
P 1 4
C 3 7
P 2 2
C 4 -5
P 1 4
5 9
1 5 -2 4 3
P 1 2
P 1 5
C 4 -5
P 1 5
P 4 5
C 3 0
P 1 5
C 4 -5
C 4 -5
```

Sample Output

```
0+-
+-+-0
```

Solução com complexidade $O(T(N+K)\log(N+K))$

- O problema pode ser resolvido adaptando-se uma árvore de Fenwick para manter o registro dos produtos
- É preciso, entretanto, tratar o zero à parte, uma vez que ele não é invertível na operação de multiplicação
- A cada elemento não-negativo, deve ser registrado apenas seu sinal (1 ou -1)
- Cada zero deve ser tratado em uma árvore à parte
- Assim,

$$RPQ(i,j) = \begin{cases} RPQ(j)/RPQ(i-j), & \text{se } RSQ(i,j) = 0 \\ 0, & \text{caso contrário} \end{cases}$$

onde $RSQ(i,j)$ se refere ao vetor z_k de zeros: $z_i = 1$ se $x_i = 0$; $z_i = 0$, caso contrário.

Solução com complexidade $O(T(N+K)\log(N+K))$

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

using namespace std;

class BITree
{
private:
    int N;
    vector<int> ft, zs;

    int LSB(int n) { return n & -n; }

public:
    BITree(int n) : N(n), ft(N + 1, 1), zs(N + 1, 0) { }

    int RPQ(int i, int j)
    {
        auto p = RPQ(j) / RPQ(i - 1);
        auto z = RSQ(i, j);
        return z ? 0 : p;
    }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
22
23      void update(int i, int v)
24      {
25          // Remove o elemento
26          auto x = RPQ(i, i);
27          x ? multiply(i, x/abs(x)) : add(i, -1);
28
29          // Insere o novo elemento
30          v ? multiply(i, v/abs(v)) : add(i, 1);
31      }
32
33  private:
34      int RPQ(int i)
35      {
36          int prod = 1;
37
38          while (i)
39          {
40              prod *= ft[i];
41              i -= LSB(i);
42          }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
43
44          return prod;
45      }
46
47      int RSQ(int i, int j)
48      {
49          return RSQ(j) - RSQ(i - 1);
50      }
51
52      int RSQ(int i)
53      {
54          int sum = 0;
55
56          while (i)
57          {
58              sum += zs[i];
59              i -= LSB(i);
60          }
61
62          return sum;
63      }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
64
65      void multiply(int i, int v)
66      {
67          while (i <= N)
68          {
69              ft[i] *= v;
70              i += LSB(i);
71          }
72      }
73
74      void add(int i, int v)
75      {
76          while (i <= N)
77          {
78              zs[i] += v;
79              i += LSB(i);
80          }
81      }
82  };
83
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
84 struct Query {
85     char c;
86     int i, j;
87 };
88
89 string solve(BITree& ft, const vector<Query>& qs)
90 {
91     ostringstream os;
92
93     for (const auto& q : qs)
94     {
95         switch (q.c) {
96         case 'C':
97             ft.update(q.i, q.j);
98             break;
99
100        default:
101            auto p = ft.RPQ(q.i, q.j);
102            os << (p ? (p > 0 ? '+' : '-') : '0');
103        }
104    }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
105
106     return os.str();
107 }
108
109 int main()
110 {
111     ios::sync_with_stdio(false);
112     int N, K;
113
114     while (cin >> N >> K)
115     {
116         BITree ft(N);
117
118         for (int i = 1; i <= N; ++i)
119         {
120             int x;
121             cin >> x;
122
123             ft.update(i, x);
124         }
125
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
126         vector<Query> qs;
127
128         while (K--)
129         {
130             string c;
131             int i, j;
132
133             cin >> c >> i >> j;
134
135             qs.push_back({ c[0], i, j });
136         }
137
138         auto ans = solve(ft, qs);
139
140         cout << ans << '\n';
141     }
142
143     return 0;
144 }
```

SPOJ – Inversion Count

Problema

Let $A[0 \ldots n-1]$ be an array of n distinct positive integers. If $i < j$ and $A[i] > A[j]$ then the pair (i, j) is called an inversion of A. Given n and an array A your task is to find the number of inversions of A.

Entrada e saída

Input

The first line contains t, the number of testcases followed by a blank space. Each of the t tests start with a number n ($n \leq 200000$). Then $n+1$ lines follow. In the ith line a number $A[i-1]$ is given ($A[i-1] \leq 10^7$). The $(n+1)$th line is a blank space.

Output

For every test output one line giving the number of inversions of A.

Exemplo de entradas e saídas

Sample Input

```
2

3
3
1
2

5
2
3
8
6
1
```

Sample Output

```
2
5
```

Solução

- Uma árvore de Fenwick pode ser utilizada para manter um histograma dos números já processados
- Assim, se os a_j elementos do vetor de entrada forem processados um a um, do fim para o início, o número de inversões onde j é o segundo elemento do par, corresponde a $RSQ(0, a_j - 1)$, isto é, ao total de números que são estritamente menores que a_j e que já apareceram no vetor
- Se os elementos a_i forem processados do início ao fim, o número de inversões onde i é o primeiro elemento do par correspondem a $RSQ(i + 1, M)$, onde $M = 10^7$ é o maior valor possível para um elemento do vetor
- Esta solução tem complexidade $O(TN \log M)$, onde T é o número de casos de teste

Solução AC com complexidade $O(TN \log M)$

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

using namespace std;
using ll = long long;

const int MAX { 10000010 };

class BITree {
private:
    vector<int> ts;
    size_t N;

public:
    BITree(size_t n) : ts(n + 1, 0), N(n) {}

    int RSQ(int i, int j)
    {
        return RSQ(j) - RSQ(i - 1);
    }

```

Solução AC com complexidade $O(TN \log M)$

```
21 private:
22     int LSB(int n) { return n & (-n); }
23
24     int RSQ(int i)
25     {
26         int sum = 0;
27
28         while (i >= 1)
29         {
30             sum += ts[i];
31             i -= LSB(i);
32         }
33
34         return sum;
35     }
36
37 public:
38     void add(size_t i, const int& x)
39     {
40         if (i == 0)
41             return;
```

Solução AC com complexidade $O(TN \log M)$

```
42
43          while (i <= N)
44          {
45              ts[i] += x;
46              i += LSB(i);
47          }
48      }
49  };
50
51  ll solve(const vector<int>& as, int N)
52  {
53      BITree ft(MAX);
54
55      ll ans = 0;
56
57      for (int i = N; i > 0; --i)
58      {
59          ans += ft.RSQ(0, as[i] - 1);
60          ft.add(as[i], 1);
61      }
62
```

Solução AC com complexidade $O(TN \log M)$

```
63      return ans;
64  }
65
66  int main()
67  {
68      ios::sync_with_stdio(false);
69
70      int T;
71      cin >> T;
72
73      while (T--)
74      {
75          int N;
76          cin >> N;
77
78          vector<int> as(N + 1);
79
80          for (int i = 1; i <= N; ++i)
81              cin >> as[i];
82
83          auto ans = solve(as, N);
```

Solução AC com complexidade $O(TN \log M)$

```
84
85          cout << ans << '\n';
86      }
87
88      return 0;
89  }
```

Codeforces Round #179 – Problem A: Greg and Array

Problema

Greg has an array $a = a_1, a_2, \ldots, a_n$ and m operations. Each operation looks as: $l_i, r_i, d_i, (1 \leq l_i \leq r_i \leq n)$. To apply operation i to the array means to increase all array elements with numbers $l_i, l_i + 1, \ldots, r_i$ by value d_i.

Greg wrote down k queries on a piece of paper. Each query has the following form: $x_i, y_i, (1 \leq x_i \leq y_i \leq m)$. That means that one should apply operations with numbers $x_i, x_i + 1, \ldots, y_i$ to the array.

Now Greg is wondering, what the array a will be after all the queries are executed. Help Greg.

Entrada e saída

Input

The first line contains integers $n, m, k (1 \leq n, m, k \leq 10^5)$. The second line contains n integers: $a_1, a_2, \ldots, a_n (0 \leq a_i \leq 10^5)$ – the initial array.

Next m lines contain operations, the operation number i is written as three integers: $l_i, r_i, d_i, (1 \leq l_i \leq r_i \leq n), (0 \leq d_i \leq 10^5)$.

Next k lines contain the queries, the query number i is written as two integers: $x_i, y_i, (1 \leq x_i \leq y_i \leq m)$.

The numbers in the lines are separated by single spaces.

Output

On a single line print n integers $a_1, a_2, \ldots, a_n$ – the array after executing all the queries. Separate the printed numbers by spaces.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the cin, cout streams of the `%I64d` specifier.

Exemplo de entradas e saídas

Sample Input	Sample Output
3 3 3 1 2 3 1 2 1 1 3 2 2 3 4 1 2 1 3 2 3	9 18 17
1 1 1 1 1 1 1 1 1	2

Solução com complexidade $O(N + (M + K)\log(M + K))$

- A solução tem três partes
- A primeira é acumular o número de vezes que cada operação deverá ser realizada
- Isto pode ser feito com uma árvore de Fenwick com suporte para *range update*
- Em seguida, deve-se acumular o impacto de cada operação no vetor original
- O número de vezes x que a operação i será aplicada pode ser feita com uma *point query* na árvore
- Novamente é necessária uma árvore de Fenwick com suporte para *range update*
- O intervalo $[L, R]$ deve ser atualizado com o valor dx
- Por fim, a cada posição do vetor i deve ser adicionado o valor y obtido pela *point query* do índice i da árvore

Solução com complexidade $O(N + (M + K)\log(M + K))$

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

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

class BITree {
private:
    vector<ll> ts;
    size_t N;

public:
    BITree(size_t n) : ts(n + 1, 0), N(n) {}

    ll value_at(int i)
    {
        return RSQ(i);
    }

    void range_add(size_t i, size_t j, ll x)
    {
```

Solução com complexidade $O(N + (M + K)\log(M + K))$

```
22          add(i, x);
23          add(j + 1, -x);
24      }
25
26  private:
27      int LSB(int n) { return n & (-n); }
28
29      ll RSQ(int i)
30      {
31          ll sum = 0;
32
33          while (i >= 1)
34          {
35              sum += ts[i];
36              i -= LSB(i);
37          }
38
39          return sum;
40      }
41
```

Solução com complexidade $O(N + (M + K)\log(M + K))$

```
42      void add(size_t i, ll x)
43      {
44          while (i <= N)
45          {
46              ts[i] += x;
47              i += LSB(i);
48          }
49      }
50  };
51
52  struct Op
53  {
54      int L, R;
55      ll d;
56  };
57
58  vector<ll> solve(int N, int M, const vector<int>& as,
59      const vector<Op>& ops, const vector<ii>& qs)
60  {
61      BITree op_tree(M);
62
```

Solução com complexidade $O(N + (M + K)\log(M + K))$

```
63      for (const auto& q : qs)
64          op_tree.range_add(q.first, q.second, 1);
65
66      BITree ft(N);
67
68      for (int i = 1; i <= M; ++i)
69      {
70          auto x = op_tree.value_at(i);
71          ft.range_add(ops[i].L, ops[i].R, x * ops[i].d);
72      }
73
74      vector<ll> ans(N + 1);
75
76      for (int i = 1; i <= N; ++i)
77          ans[i] = as[i] + ft.value_at(i);
78
79      return ans;
80  }
81
82  int main()
83  {
```

Solução com complexidade $O(N + (M + K) \log(M + K))$

```
84      ios::sync_with_stdio(false);
85
86      int N, M, K;
87      cin >> N >> M >> K;
88
89      vector<int> as(N + 1);
90
91      for (int i = 1; i <= N; ++i)
92          cin >> as[i];
93
94      vector<Op> ops(M + 1);
95
96      for (int i = 1; i <= M; ++i)
97      {
98          int L, R, d;
99          cin >> L >> R >> d;
100
101         ops[i] = Op { L, R, d };
102     }
103
104     vector<ii> qs(K);
```

Solução com complexidade $O(N + (M + K)\log(M + K))$

```
105
106     for (int i = 0; i < K; ++i)
107         cin >> qs[i].first >> qs[i].second;
108
109     auto ans = solve(N, M, as, ops, qs);
110
111     for (size_t i = 1; i < ans.size(); ++i)
112         cout << ans[i] << (i + 1 == ans.size() ? '\n' : ' ');
113
114     return 0;
115 }
```

POJ 1195 – Mobile Phones

Problema

Suppose that the fourth generation mobile phone base stations in the Tampere area operate as follows. The area is divided into squares. The squares form an $S \times S$ matrix with the rows and columns numbered from 0 to $S - 1$. Each square contains a base station. The number of active mobile phones inside a square can change because a phone is moved from a square to another or a phone is switched on or off. At times, each base station reports the change in the number of active phones to the main base station along with the row and the column of the matrix.

Write a program, which receives these reports and answers queries about the current total number of active mobile phones in any rectangle-shaped area.

Entrada e saída

Input

The input is read from standard input as integers and the answers to the queries are written to standard output as integers. The input is encoded as follows. Each input comes on a separate line, and consists of one instruction integer and a number of parameter integers according to the following table.

Instruction	Parameters	Meaning
0	S	Initialize the matrix size to $S \times S$ containing all zeros. This instruction is given only once and it will be the first instruction

Entrada e saída

Instruction	Parameters	Meaning
1	$X\ Y\ A$	Add A to the number of active phons in table square (X, Y). A may be positve or negative.
2	$L\ B\ R\ T$	Query the current sum of numbers of active mobile phones in squares (X, Y), where $L \leq X \leq R, B \leq Y \leq T$
3		Terminate program. This instruction is given only once and it will be the last instruction.

Entrada e saída

The values will always be in range, so there is no need to check them. In particular, if A is negative, it can be assumed that it will not reduce the square value below zero. The indexing starts at 0, e.g. for a table of size 4×4, we have $0 \leq X \leq 3$ and $0 \leq Y \leq 3$.

Table size: $1 \times 1 \leq S \times S \leq 1024 \times 1024$

Cell value V at any time: $0 \leq V \leq 32767$

Update amount: $-32768 \leq A \leq 32767$

No of instructions in input: $3 \leq U \leq 60002$

Maximum number of phones in the whole table: $M = 2^{30}$

Entrada e saída

Output

Your program should not answer anything to lines with an instruction other than 2. If the instruction is 2, then your program is expected to answer the query by writing the answer as a single line containing a single integer to standard output.

Exemplo de entradas e saídas

Sample Input

```
0 4
1 1 2 3
2 0 0 2 2
1 1 1 2
1 1 2 -1
2 1 1 2 3
3
```

Sample Output

```
3
4
```

Solução

- Este problema pode ser resolvido diretamente através da implementação de uma árvore de Fenwick bidimensional
- Deve-se tomar alguns cuidados, porque o juiz roda em servidores antigos e os compiladores não são os mais recentes disponíveis
- Evite qualquer elemento que foi introduzido na linguagem C++ após a versão 98
- Evite os contêiners da STL
- Os índices usados começam em zero, e isto deve ser refletido na implementação
- O cabeçalho `bits/stdc++.h` não está disponível

Solução com complexidade $O(T(N+K)\log(N+K))$

```cpp
#include <iostream>
#include <vector>

using namespace std;

const int MAX { 1030 };
int ft[MAX][MAX];

class BITree2D {
public:
    int N;

    BITree2D() : N(0) {}

    void set_n(size_t n) { N = n; }

    // Range query
    int RSQ(int a, int b, int c, int d)
    {
        return RSQ(c, d) - RSQ(c, b-1) - RSQ(a-1, d) + RSQ(a-1, b-1);
    }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
23      // Point update
24      void add(int x, int y, int v)
25      {
26          for (int i = x; i <= N; i += LSB(i))
27              for (int j = y; j <= N; j += LSB(j))
28                  ft[i][j] += v;
29      }
30
31  private:
32      int LSB(int n) { return n & -n; }
33
34      int RSQ(int a, int b)
35      {
36          int sum = 0;
37
38          for (int i = a; i > 0; i -= LSB(i))
39              for (int j = b; j > 0; j -= LSB(j))
40                  sum += ft[i][j];
41
42          return sum;
43      }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
44 };
45
46 int main()
47 {
48     int cmd, a, b, c, d;
49     BITree2D ft;
50
51     while (cin >> cmd, cmd != 3)
52     {
53         switch (cmd) {
54         case 0:
55             cin >> a;
56             ft.set_n(a);
57             break;
58
59         case 1:
60             cin >> a >> b >> c;
61             ft.add(a + 1, b + 1, c);
62             break;
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
63
64          default:
65              cin >> a >> b >> c >> d;
66              cout << ft.RSQ(a + 1, b + 1, c + 1, d + 1) << '\n';
67          }
68      }
69
70      return 0;
71  }
```

Referências

1. UVA 12532 – Interval Product
2. SPOJ - Inversion Count
3. Codeforces Round #179 (Div. 1) – Problem A: Greg and Array
4. POJ 1195 – Mobile Phones