Pilhas e Filas

Pilhas: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2020

Sumário

Educational Codeforces Round 4 – Problema C: Replace To Make Regular Bracket Sequence

Problema

You are given string s consists of opening and closing brackets of four kinds <>, {}, [], (). There are two types of brackets: opening and closing. You can replace any bracket by another of the same type. For example, you can replace < by the bracket {, but you can't replace it by) or >.

The following definition of a regular bracket sequence is well-known, so you can be familiar with it.

Let's define a regular bracket sequence (RBS). Empty string is RBS. Let s_1 and s_2 be a RBS then the strings <s_1>s_2, {s_1} s_2, [s_1]s_2, (s_1)s_2 are also RBS.

For example the string "[[(){}]<>]" is RBS, but the strings "[)()" and "][()()" are not.

Determine the least number of replaces to make the string s RBS.

Entrada e saída

Input

The only line contains a non empty string s, consisting of only opening and closing brackets of four kinds. The length of s does not exceed 10^6.

Output

If it's impossible to get RBS from s print `Impossible`.

Otherwise print the least number of replaces needed to get RBS from s.

Exemplo de entradas e saídas

Sample Input	Sample Output
[<}){}	2
{()}[]	0
]]	Impossible

Solução com complexidade $O(N)$

- Para que a string seja uma RBS, é preciso que cada símbolo aberto seja fechado pelo símbolo correspondente
- Em tais expressões, o símbolo de fechar será associado ao símbolo de abrir correspondente mais próximo à esquerda
- Logo, os símbolos ainda em aberto devem ser armazenados em uma pilha
- A cada símbolo de fechar, o topo da pilha deve ser consultado: se não for o símbolo correspondente, uma substituição deve ser feita
- Caso a pilha esteja vazia, ou tenha ao menos um símbolo pendente após o processamento de toda string S, a sequência não corresponde a uma RBS
- Como cada caractere é avaliado, no máximo, 2 vezes (uma na inserção e outra na remoção da pilha), esta solução tem complexidade $O(N)$

Solução AC com complexidade $O(N)$

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

using namespace std;

map<char, char> open { {'>', '<'}, {']', '['}, {'}', '{'}, {')', '('} };

int solve(const string& S)
{
    stack<char> st;
    int ans = 0;

    for (const auto& c : S)
    {
        switch (c) {
        case '(':
        case '<':
        case '{':
        case '[':
            st.push(c);
            break;

```

Solução AC com complexidade $O(N)$

```
22          default:
23              if (st.empty())
24                  return -1;
25
26              ans += (open[c] == st.top() ? 0 : 1);
27              st.pop();
28          }
29      }
30
31      return st.empty() ? ans : -1;
32  }
33
34  int main()
35  {
36      ios::sync_with_stdio(false);
37
38      string S;
39      cin >> S;
40
41      auto ans = solve(S);
42
```

Solução AC com complexidade $O(N)$

```
43      if (ans == -1)
44          cout << "Impossible\n";
45      else
46          cout << ans << '\n';
47
48      return 0;
49  }
```

OJ 11111 – Generalized Matrioshkas

Problema

Vladimir worked for years making matrioshkas, those nesting dolls that certainly represent truly Russian craft. A matrioshka is a doll that may be opened in two halves, so that one finds another doll inside. Then this doll may be opened to find another one inside it. This can be repeated several times, till a final doll – that cannot be opened – is reached.

Recently, Vladimir realized that the idea of nesting dolls might be generalized to nesting toys. Indeed, he has designed toys that contain toys but in a more general sense. One of these toys may be opened in two halves and it may have more than one toy inside it. That is the new feature that Vladimir wants to introduce in his new line of toys.

Problema

Vladimir has developed a notation to describe how nesting toys should be constructed. A toy is represented with a positive integer, according to its size. More precisely: if when opening the toy represented by m we find the toys represented by $n_1, n_2, \ldots, n_r$, it must be true that $n_1 + n_2 + ... + n_r < m$. And if this is the case, we say that toy m contains directly the toys $n_1, n_2, \ldots, n_r$. It should be clear that toys that may be contained in any of the toys $n_1, n_2, \ldots, n_r$ are not considered as directly contained in the toy m.

A *generalized matrioshka* is denoted with a non-empty sequence of non zero integers of the form:

$$a_1 \quad a_2 \quad \ldots \quad a_N$$

such that toy k is represented in the sequence with two integers $-k$ and k, with the negative one occurring in the sequence first that the positive one.

Problema

For example, the sequence

$$-9 \quad -7 \quad -2 \quad 2 \quad -3 \quad -2 \quad -1 \quad 1 \quad 2 \quad 3 \quad 7 \quad 9$$

represents a generalized matrioshka conformed by six toys, namely, 1, 2 (twice), 3, 7 and 9. Note that toy 7 contains directly toys 2 and 3. Note that the first copy of toy 2 occurs left from the second one and that the second copy contains directly a toy 1. It would be wrong to understand that the first -2 and the last 2 should be paired.

On the other hand, the following sequences do not describe generalized matrioshkas:

- $$-9 \quad -7 \quad -2 \quad 2 \quad -3 \quad -1 \quad -2 \quad 2 \quad 1 \quad 3 \quad 7 \quad 9$$

  because toy 2 is bigger than toy 1 and cannot be allocated inside it.

Problema

- $$-9 \quad -7 \quad -2 \quad 2 \quad -3 \quad -2 \quad -1 \quad 1 \quad 2 \quad 3 \quad 7 \quad -2 \quad 2 \quad 9$$

  because 7 and 2 may not be allocated together inside 9.

- $$-9 \quad -7 \quad -2 \quad 2 \quad -3 \quad -1 \quad -2 \quad 3 \quad 2 \quad 1 \quad 7 \quad 9$$

  because there is a nesting problem within toy 3.

Your problem is to write a program to help Vladimir telling good designs from bad ones.

Entrada e saída

Input

The input file contains several test cases, each one of them in a separate line. Each test case is a sequence of non zero integers, each one with an absolute value less than 10^7.

Output

Output texts for each input case are presented in the same order that input is read.

For each test case the answer must be a line of the form

```
:-) Matrioshka!
```

if the design describes a generalized matrioshka. In other case, the answer should be of the form

```
:-( Try again.
```

Exemplo de entradas e saídas

Sample Input	Sample Output
-9 -7 -2 2 -3 -2 -1 1 2 3 7 9	:-) Matrioshka!
-9 -7 -2 2 -3 -1 -2 2 1 3 7 9	:-(Try again.
-9 -7 -2 2 -3 -1 -2 3 2 1 7 9	:-(Try again.
-100 -50 -6 6 50 100	:-) Matrioshka!
-100 -50 -6 6 45 100	:-(Try again.
-10 -5 -2 2 5 -4 -3 3 4 10	:-) Matrioshka!
-9 -5 -2 2 5 -4 -3 3 4 9	:-(Try again.

Solução com complexidade $O(N)$

- Primeiramente, é preciso avaliar se cada brinquedo aberto ($-x$) é fechado corretamente (tem um x associado)
- Entre os valores $-x$ e x devem haver brinquedos que abrem e fecham corretamente, cuja soma dos valores seja menor do que x
- Este problema é uma variante do problema do pareamento de parêntesis
- O uso de uma pilha permite a verificação de ambas condições simultaneamente
- Para cada valor x da entrada, duas providências devem ser tomadas
- Se x por negativo, ele deve ser inserido no topo da pilha

Solução com complexidade $O(N)$

- Se x for positivo, deve se contabilizar os valores de todos os brinquedos que estão dentro de x
- Para tal, basta acumular os valores positivos que forem removidos da pilha até se alcançar um valor negativo
- Caso o valor negativo seja diferente de $-x$, ou se a pilha se tornar vazia antes de atingir um valor negativo, ou se o total dos valores dos brinquedos dentro de x for maior ou igual a x, o design não é válido
- Caso o valor atingido seja $-x$ e o total acumulado seja menor que x, o valor $-x$ deve ser retirado da pilha e o valor x inserido no topo
- Ao final do processo, o design será válido se a pilha contiver apenas valores positivos

Solução com complexidade $O(N)$

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

using namespace std;

bool matrioska(const vector<int>& xs)
{
    stack<int> s;

    for (auto x : xs)
    {
        if (x < 0)
            s.push(x);
        else
        {
            int stacked = 0;

            while (not s.empty() and s.top() > 0)
            {
                stacked += s.top();
                s.pop();
            }
```

Solução com complexidade $O(N)$

```
22
23              if (stacked >= x or s.empty() or s.top() != -x)
24                  return false;
25
26              s.pop();
27              s.push(x);
28          }
29      }
30
31      while (not s.empty() and s.top() > 0)
32          s.pop();
33
34      return s.empty();
35  }
36
37  int main()
38  {
39      ios::sync_with_stdio(false);
40
41      string line;
42
```

Solução com complexidade $O(N)$

```
43      while (getline(cin, line))
44      {
45          istringstream is(line);
46
47          vector<int> xs;
48          copy(istream_iterator<int>(is), istream_iterator<int>(),
49               back_inserter(xs));
50
51          cout << ":-" << (matrioska(xs) ? ") Matrioshka!" :
52              "( Try again.") << '\n';
53      }
54
55      return 0;
56  }
```

Referências

1. Educational Codeforces Round 4 – Problem C: Replace To Make Regular Bracket Sequence
2. OJ 11111 – Generalized Matrioshkas