OJ 12504

Updating the Dictionary

Prof. Edson Alves – UnB/FGA

Problema

In this problem, a dictionary is collection of key-value pairs, where keys are lower-case letters, and values are non-negative integers. Given an old dictionary and a new dictionary, find out what were changed.

Each dictionary is formatting as follows:

$$key : value, key : value, \ldots, key : value$$

Each key is a string of lower-case letters, and each value is a non-negative integer without leading zeros or prefix '+'. (i.e. -4, 03 and $+77$ are illegal). Each key will appear at most once, but keys can appear in any order

Entrada e saída

Input

The first line contains the number of test cases T ($T \leq 1000$). Each test case contains two lines. The first line contains the old dictionary, and the second line contains the new dictionary. Each line will contain at most 100 characters and will not contain any whitespace characters. Both dictionaries could be empty.

WARNING: there are no restrictions on the lengths of each key and value in the dictionary. That means keys could be really long and values could be really large.

Entrada e saída

Output

For each test case, print the changes, formatted as follows:

- First, if there are any new keys, print '+' and then the new keys in increasing order (lexicographically), separated by commas.
- Second, if there are any removed keys, print '-' and then the removed keys in increasing order (lexicographically), separated by commas.
- Last, if there are any keys with changed value, print '*' and then these keys in increasing order (lexicographically), separated by commas.

If the two dictionaries are identical, print 'No changes' (without quotes) instead.

Print a blank line after each test case.

Exemplo de entradas e saídas

Sample Input

```
3
{a:3,b:4,c:10,f:6}
{a:3,c:5,d:10,ee:4}
{x:1,xyz:123456789123456789123456789}
{xyz:123456789123456789123456789,x:1}
{first:1,second:2,third:3}
{third:3,second:2}
```

Sample Output

```
+d,ee
-b,f
*c

No changes

-first
```

Solução com complexidade $O(TN)$

- Seja N o maior número possível de entradas em um dicionário
- Cada dicionário pode ser representado por um `unordered_map`, o que permite a construção de cada um deles em $O(N)$
- Esta construção pode ser feita por meio de um *parser* escrito manualmente ou por meio de chamadas sucessivas da função `getline()`, usando o separador apropriado
- Para determinar os termos que foram removidos ou alterados, basta passar em todas as entradas do primeiro dicionário (x) e tentar localizá-las no segundo (y)
- Caso a chave $k \in x$ não pertença a y, ela deve ser armazenada entre os elementos removidos
- Se $k \in x$ e $x[k] \neq y[k]$, ela representa um dos elementos que foram alterados
- Uma chave k foi adicionada se $k \in y$ e $k \notin x$

Solução com complexidade $O(TN)$

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

using namespace std;
using dict = unordered_map<string, string>;

vector<vector<string>> solve(const dict& x, const dict& y)
{
    vector<string> added, removed, changed;

    for (const auto& p : x)
    {
        auto k = p.first;
        auto v = p.second;

        if (y.count(k) == 0)
            removed.push_back(k);
        else if (y.count(k) == 1 and v != y.at(k))
            changed.push_back(k);
    }
```

Solução com complexidade $O(TN)$

```
21      for (const auto& p : y)
22      {
23          auto k = p.first;
24
25          if (x.count(k) == 0)
26              added.push_back(k);
27      }
28
29      return { added, removed, changed };
30  }
31
32  dict read_dict(const string& line)
33  {
34      string info = line.substr(1);
35      info.pop_back();
36
37      istringstream is(info);
38      string key, value;
39
40      dict d;
```

Solução com complexidade $O(TN)$

```
42      while (getline(is, key, ':'), getline(is, value, ','))
43          d[key] = value;
44
45      return d;
46  }
47
48  int main()
49  {
50      ios::sync_with_stdio(false);
51      string line;
52
53      getline(cin, line);
54      int T = stoi(line);
55
56      while (T--) {
57          getline(cin, line);
58          auto x = read_dict(line);
59
60          getline(cin, line);
61          auto y = read_dict(line);
```

Solução com complexidade $O(TN)$

```
63          auto ans = solve(x, y);
64
65          if (x == y)
66              cout << "No changes\n";
67          else
68          {
69              const string prefix { "+-*" };
70
71              for (int i = 0; i < 3; ++i)
72              {
73                  if (ans[i].empty())
74                      continue;
75
76                  sort(ans[i].begin(), ans[i].end());
77
78                  cout << prefix[i];
79
80                  auto N = ans[i].size();
```

Solução com complexidade $O(TN)$

```
82                 for (size_t j = 0; j < N; ++j)
83                     cout << ans[i][j] << (j + 1 == N ? '\n' : ',');
84             }
85         }
86
87         cout << '\n';
88     }
89
90     return 0;
91 }
```