Hash

Hash em C++: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

Codeforces Round #150 (Div. 2) – Problem A: Dividing Orange

Problema

One day Ms Swan bought an orange in a shop. The orange consisted of $n \cdot k$ segments, numbered with integers from 1 to $n \cdot k$.

There were k children waiting for Ms Swan at home. The children have recently learned about the orange and they decided to divide it between them. For that each child took a piece of paper and wrote the number of the segment that he would like to get: the i-th $(1 \leq i \leq k)$ child wrote the number a_i $(1 \leq a_i \leq n \cdot k)$. All numbers a_i accidentally turned out to be different.

Problema

Now the children wonder, how to divide the orange so as to meet these conditions:

- each child gets exactly n orange segments;
- the i-th child gets the segment with number a_i for sure;
- no segment goes to two children simultaneously.

Help the children, divide the orange and fulfill the requirements, described above.

Entrada e saída

Input

The first line contains two integers n, k $(1 \leq n, k \leq 30)$. The second line contains k space-separated integers $a_1, a_2, \ldots, a_k$ $(1 \leq a_i \leq n \cdot k)$, where a_i is the number of the orange segment that the i-th child would like to get.

It is guaranteed that all numbers a_i are distinct.

Output

Print exactly $n \cdot k$ distinct integers. The first n integers represent the indexes of the segments the first child will get, the second n integers represent the indexes of the segments the second child will get, and so on. Separate the printed numbers with whitespaces.

You can print a child's segment indexes in any order. It is guaranteed that the answer always exists. If there are multiple correct answers, print any of them.

Exemplo de entradas e saídas

Sample Input

```
2 2
4 1

3 1
2
```

Sample Output

```
2 4
1 3

3 2 1
```

Solução com complexidade $O(N)$

- Este problema pode ser resolvido com complexidade linear por meio do uso de um `unordered_set` S
- Este conjunto representa os segmentos da laranja que já foram atribuídos a uma das crianças
- Inicialmente deve ser inseridos em S todos os valores a_i
- Como o `unordered_set` utiliza *hashes* em sua implementação, cada consulta ou inserçaõ em S tem complexidade média $O(1)$
- Em seguida, inicializa-se uma variável i apontando para o próximo segmento a ser considerado
- Assim, para cada uma das crianças, avalia-se se i está ou não disponível
- Se estiver disponível, i é atribuído à criança e acrescido em S
- Quando uma criança tiver N segmentos o processamento para a criança seguinte
- Como cada segmento será processado uma única vez, a complexidade da solução é $O(N)$

Solução AC com complexidade $O(N)$

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

using namespace std;

vector<vector<int>> solve(size_t N, int K, const vector<int>& as)
{
    unordered_set<int> used(as.begin(), as.end());
    vector<vector<int>> ans(K);
    int nxt = 1;

    for (int i = 0; i < K; ++i)
    {
        ans[i].push_back(as[i]);

        while (ans[i].size() < N)
        {
            if (used.count(nxt) == 0)
            {
                ans[i].push_back(nxt);
                used.insert(nxt);
            }
```

Solução AC com complexidade $O(N)$

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

Solução AC com complexidade $O(N)$

```
43
44      for (const auto& xs : ans)
45          for (int i = 0; i < N; ++i)
46              cout << xs[i] << (i + 1 == N ? '\n' : ' ');
47
48      return 0;
49  }
```

URI 1256 – Tabelas Hash

Problema

As tabelas Hash, também conhecidas como tabelas de dispersão, armazenam elementos com base no valor absoluto de suas chaves e em técnicas de tratamento de colisões. Para o cálculo do endereço onde deve ser armazenada uma determinada chave, utiliza-se uma função denominada função de dispersão, que transforma a chave em um dos endereços disponíveis na tabela.

Suponha que uma aplicação utilize uma tabela de dispersão com 13 endereços-base (índices de 0 a 12) e empregue a função de dispersão $h(x) = x \text{ mod } 13$, em que x representa a chave do elemento cujo endereço-base deve ser calculado.

Problema

Se a chave x for igual a 49, a função de dispersão retornará o valor 10, indicando o local onde esta chave deverá ser armazenada. Se a mesma aplicação considerar a inserção da chave 88, o cálculo retornará o mesmo valor 10, ocorrendo neste caso uma colisão. O Tratamento de colisões serve para resolver os conflitos nos casos onde mais de uma chave é mapeada para um mesmo endereço-base da tabela. Este tratamento pode considerar, ou o recálculo do endereço da chave ou o encadeamento externo ou exterior.

O professor gostaria então que você o auxiliasse com um programa que calcula o endereço para inserções de diversas chaves em algumas tabelas, com funções de dispersão e tratamento de colisão por encadeamento exterior.

Entrada e saída

Entrada

A entrada contém vários casos de teste. A primeira linha de entrada contém um inteiro N indicando a quantidade de casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha contém um valor M $(1 \leq M \leq 100)$ que indica a quantidade de endereços-base na tabela (normalmente um número primo) seguido por um espaço e um valor C $(1 \leq C \leq 200)$ que indica a quantidade de chaves a serem armazenadas. A segunda linha contém cada uma das chaves (com valor entre 1 e 200), separadas por um espaço em branco.

Saída

A saída deverá ser impressa conforme os exemplos fornecidos abaixo, onde a quantidade de linhas de cada caso de teste é determinada pelo valor de M. Uma linha em branco deverá separar dois conjuntos de saída.

Exemplo de entradas e saídas

Exemplo de Entrada

```
2
13 9
44 45 49 70 27 73 92 97 95
7 8
35 12 2 17 19 51 88 86
```

Exemplo de Saída

```
0 -> \
1 -> 27 -> 92 -> \
2 -> \
3 -> \
4 -> 95 -> \
5 -> 44 -> 70 -> \
6 -> 45 -> 97 -> \
7 -> \
8 -> 73 -> \
9 -> \
10 -> 49 -> \
11 -> \
12 -> \

0 -> 35 -> \
1 -> \
2 -> 2 -> 51 -> 86 -> \
3 -> 17 -> \
4 -> 88 -> \
5 -> 12 -> 19 -> \
6 -> \
```

Solução com complexidade $O(NM)$

- O problema consiste em implementar uma tabela *hash* com resolução de colisão por encadeamento
- Esta tabela pode ser representa por um vetor de vetores da linguagem C++, por um `multimap` ou por um `unordered_set`
- A solução com `multimap` adiciona um fator $O(\log N)$ nas operações de inserção
- As outras duas soluções tem inserção em $O(1)$
- A primeira alternativa permite a impressão de cada célula por meio da impressão do vetor correspondente
- Já usando o `unordered_set` os elementos de cada célula podem ser obtidos utilizando as informações do método `bucket_size()` e um iterator que aponta inicialmente para o primeiro elemento

Solução com complexidade $O(NM)$

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

using namespace std;

vector<vector<int>> solve(int M, const vector<int>& ks)
{
    vector<vector<int>> hs(M);

    for (const auto& k : ks)
        hs[k % M].push_back(k);

    return hs;
}

int main()
{
    int N;
    cin >> N;

    for (int test = 0; test < N; ++test)
    {
```

Solução com complexidade $O(NM)$

```
22          int M, C;
23          cin >> M >> C;
24
25          vector<int> ks(C);
26
27          for (int i = 0; i < C; i++)
28              cin >> ks[i];
29
30          auto hs = solve(M, ks);
31
32          if (test)
33              cout << '\n';
34
35          for (int i = 0; i < M; i++)
36          {
37              cout << i << " -> ";
38
39              for (const auto& x : hs[i])
40                  cout << x << " -> ";
41
42              cout << "\\" << '\n';
```

Solução com complexidade $O(NM)$

```
43            }
44        }
45
46        return 0;
47    }
```

UVA 12504 – Updating the Dictionary

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;
41
```

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      {
58          getline(cin, line);
59          auto x = read_dict(line);
60
61          getline(cin, line);
62          auto y = read_dict(line);
```

Solução com complexidade $O(TN)$

```
64          auto ans = solve(x, y);
65
66          if (x == y)
67              cout << "No changes\n";
68          else
69          {
70              const string prefix { "+-*" };
71
72              for (int i = 0; i < 3; ++i)
73              {
74                  if (ans[i].empty())
75                      continue;
76
77                  sort(ans[i].begin(), ans[i].end());
78
79                  cout << prefix[i];
80
81                  auto N = ans[i].size();
82
83                  for (size_t j = 0; j < N; ++j)
84                      cout << ans[i][j] << (j + 1 == N ? '\n' : ',');
```

Solução com complexidade $O(TN)$

```
85                }
86            }
87
88            cout << '\n';
89        }
90
91        return 0;
92    }
```

Referências

1. Codeforces Round #150 (Div. 2) – Problem A: Dividing Orange
2. URI 1256 – Tabelas Hash
3. UVA 12504 – Updating a Dictionary