Árvores

Árvores Binárias de Busca na STL: Problemas Resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

URI 1897 – Jogo Esperto

Problema

Enquanto Bino descansava, inventou um jogo esperto. Dado um número N e um número M, Bino quer saber qual a quantidade mínima de operações para converter N em M.

Existe seis operações permitidas.

- Operação 1: $N = N * 2$
- Operação 2: $N = N * 3$
- Operação 3: $N = N/2$
- Operação 4: $N = N/3$
- Operação 5: $N = N + 7$
- Operação 6: $N = N - 7$

Entrada e saída

Entrada

A entrada contém dois inteiros N $(0 \leq N \leq 10000)$ e M $(0 \leq M \leq 10000)$.

Saída

A saída é o número mínimo de operações para converter N em M.

Exemplo de entradas e saídas

Exemplo de Entrada	Exemplo de Saída
10 15	2
45 15	1
84 63	3

Solução com complexidade $O(|N - M|)$

- Observe que a sequência de operações 1, 2, 5, 4, 3 efetivamente acrescenta uma unidade em N:

$$N \to 2N \to 6N \to 6N + 7 \to 2N + 2 \to N + 1$$

- De forma semelhante, a sequência 1, 2, 6, 4, 3 subtrai uma unidade de N
- Assim, em no máximo $5|N - M|$ operações é possível alcançar M e o problema sempre terá solução
- Para determinar o mínimo de soluções com complexidade $O(|N - M|)$, basta armazenar N em uma fila e inserir seu valor em um conjunto
- A cada etapa, se o próximo elemento da fila x não for igual a M, todos os resultados das operações em x que não tiverem sido inseridos no conjunto ainda devem entrar na fila e no conjunto
- Na fila deve ser armazenado, além do valor x, quantas operações foram necessárias para encontrá-lo

Solução com complexidade $O(|N - M|)$

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

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

int solve(int N, int M)
{
    queue<ii> ns;
    set<int> found;

    ns.push(make_pair(N, 0));
    found.insert(N);

    while (not ns.empty())
    {
        auto p = ns.front();
        ns.pop();

        int n = p.first;
        int ops = p.second;

```

Solução com complexidade $O(|N - M|)$

```
22          if (n == M)
23              return ops;
24
25          vector<int> xs { n * 2, n * 3, n / 2, n / 3, n + 7, n - 7 };
26
27          for (const auto& x : xs)
28          {
29              if (found.count(x) == 0)
30              {
31                  ns.push(ii(x, ops + 1));
32                  found.insert(x);
33              }
34          }
35      }
36
37      return -1;
38  }
39
```

Solução com complexidade $O(|N - M|)$

```
40 int main()
41 {
42     int N, M;
43     cin >> N >> M;
44
45     auto ans = solve(N, M);
46
47     cout << ans << endl;
48
49     return 0;
50 }
```

UVA 12049 – Just Prune The List

Problema

You are given two list of integers. You can remove any number of elements from any of them. You have to ensure that after removing some elements both of the list will contain same elements, but not necessarily in same order. For example consider the following two lists

List # 1	1	2	3	2	1
List # 2	1	2	5	2	3

After removing 1 from first list and 5 from second list, both lists will contain same elements. We will find the following lists after removing two elements.

List # 1	1	2	3	2
List # 2	1	2	2	3

What is the minimum number of elements to be removed to obtain two list having same elements?

Entrada e saída

Input

The first line of the input file contains an integer T $(T \leq 100)$ which denotes the total number of test cases. The description of each test case is given below:

First line will contain two integers N $(1 \leq N \leq 10000)$, the number of element in the first list and M $(1 \leq M \leq 10000)$, the number of element in the second list. The next line will contain N integers representing the first list followed by another line having M elements representing the second list. Each integers in the input is 32 bit signed integer.

Output

For each test case output a single line containing the number of elements to be removed. See sample output for exact format.

Exemplo de entradas e saídas

Sample Input

```
1
5 5
1 2 3 2 1
1 2 5 2 3
```

Sample Output

```
2
```

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

- O problema consiste em determinar quantos elementos, em ambas listas, não pertencem a interseção de ambas
- Observe que as listas podem conter elementos repetidos: armazená-los em um `set` levaria a perda de informações importantes
- Uma solução é armazenar ambas listas nos `multisets` s e r e, para cada elemento de r, verificar se ele está ou não em s
- Se estiver, uma cópia dele deve ser removida de s
- Caso contrário, é um elemento a ser eliminado, acrescentando uma unidade à resposta
- Ao final do processo, cada elemento restante em s acrescenta uma unidade ao resultado final
- A STL do C++ pode ser utilizada: a função `set_intersection()` computa a interseção entre dois contêiners ordenados
- Daí a resposta para a ser a soma $N + M$, subtraída do dobro da interseção

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

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

using namespace std;

int solve(multiset<int>& s, const multiset<int>& r)
{
    vector<int> xs;
    set_intersection(s.begin(), s.end(), r.begin(), r.end(),
        back_inserter(xs));
    return s.size() + r.size() - 2*xs.size();
}

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

    while (T--)
    {
        int N, M, x;
        cin >> N >> M;
```

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

```
22
23          multiset<int> s, r;
24
25          while (N--)
26          {
27              cin >> x;
28              s.insert(x);
29          }
30
31          while (M--)
32          {
33              cin >> x;
34              r.insert(x);
35          }
36
37          cout << solve(s, r) << '\n';
38      }
39
40      return 0;
41  }
```

AtCoder Beginner Contest 110 – Problem C: String Transformation

Problema

You are given strings S and T consisting of lowercase English letters.

You can perform the following operation on S any number of times:

Operation: Choose two distinct lowercase English letters c_1 and c_2, then replace every occurrence of c_1 with c_2, and every occurrence of c_2 with c_1.

Determine if S and T can be made equal by performing the operation zero or more times.

Entrada e saída

Constraints

- $1 \leq |S| \leq 2 \times 10^5$
- $|S| = |T|$
- S and T consists of lowercase English letters.

Input

Input is given from Standard Input in the following format:

$$S$$
$$T$$

Output

If S and T can be made equal, print `'Yes'`; otherwise, print `'No'`.

Exemplo de entradas e saídas

Exemplo de Entrada	Exemplo de Saída
azzel apple	Yes
chokudai redcoder	No
abcdefghijklmnopqrstuvwxyz ibyhqfrekavclxjstdwgpzmonu	Yes

Solução com complexidade $O(|S|)$

- Observe que, segundo as regras da operação dada, a correspondência entre dois caracteres de S e T é biunívoca
- Assim, um caractere c de S deve ser trocado por um caractere d de T que ocorra exatamente o mesmo número de vezes em ambas strings
- Além do número de ocorrências, é preciso determinar a localização destas ocorrências
- Assim, devem ser mantidos dois dicionários D_S e D_T que mantém o registro das substituições a serem realizadas
- Assim, para cada caractere c de S, se ainda não tiver sido definida sua substituição e nem a transformação do caractere d de T, ambas substituições são registradas
- Se uma substituição estiver definida e a outra não, será impossível transformar S em T
- Se a substituição de c já estiver definida mas for diferente de d, também será impossível obter T a partir de S

Solução com complexidade $O(|S|)$

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

using namespace std;

bool solve(const string& S, const string& T)
{
    auto N = S.size();
    map<char, char> s_table, t_table;

    for (size_t i = 0; i < N; ++i)
    {
        auto c = S[i];
        auto d = T[i];
        auto it = s_table.find(c);

```

Solução com complexidade $O(|S|)$

```
16          if (it == s_table.end())
17          {
18              auto jt = t_table.find(d);
19
20              if (jt == t_table.end())
21              {
22                  s_table[c] = d;
23                  t_table[d] = c;
24              } else
25                  return false;
26          } else if (it->second != d)
27                  return false;
28      }
29
30      return true;
31  }
32
```

Solução com complexidade $O(|S|)$

```
33 int main()
34 {
35     ios::sync_with_stdio(false);
36
37     string S, T;
38     cin >> S >> T;
39
40     auto ans = solve(S, T);
41
42     cout << (ans ? "Yes" : "No") << '\n';
43
44     return 0;
45 }
```

Codeforces Round #492 – Problema C: Sonya and Robots

Problema

Since Sonya is interested in robotics too, she decided to construct robots that will read and recognize numbers.

Sonya has drawn n numbers in a row, a_i is located in the i-th position. She also has put a robot at each end of the row (to the left of the first number and to the right of the last number). Sonya will give a number to each robot (they can be either same or different) and run them. When a robot is running, it is moving toward to another robot, reading numbers in the row. When a robot is reading a number that is equal to the number that was given to that robot, it will turn off and stay in the same position.

Sonya does not want robots to break, so she will give such numbers that robots will stop before they meet. That is, the girl wants them to stop at different positions so that the first robot is to the left of the second one.

Problema

For example, if the numbers $[1, 5, 4, 1, 3]$ are written, and Sonya gives the number 1 to the first robot and the number 4 to the second one, the first robot will stop in the 1-st position while the second one in the 3-rd position. In that case, robots will not meet each other. As a result, robots will not be broken. But if Sonya gives the number 4 to the first robot and the number 5 to the second one, they will meet since the first robot will stop in the 3-rd position while the second one is in the 2-nd position.

Sonya understands that it does not make sense to give a number that is not written in the row because a robot will not find this number and will meet the other robot.

Problema

Sonya is now interested in finding the number of different pairs that she can give to robots so that they will not meet. In other words, she wants to know the number of pairs (p, q), where she will give p to the first robot and q to the second one. Pairs (p_i, q_i) and (p_j, q_j) are different if $p_i \neq p_j$ or $q_i \neq q_j$.

Unfortunately, Sonya is busy fixing robots that broke after a failed launch. That is why she is asking you to find the number of pairs that she can give to robots so that they will not meet.

Entrada e saída

Input

The first line contains a single integer n $(1 \leq n \leq 10^5)$ – the number of numbers in a row.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 10^5)$ – the numbers in a row.

Output

Print one number – the number of possible pairs that Sonya can give to robots so that they will not meet.

Exemplo de entradas e saídas

Sample Input	Sample Output
5 1 5 4 1 3	9
7 1 2 1 1 1 3 2	7

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

- Para computar o número de pares (a, b) distintos, é preciso observar que a pode assumir apenas valores distintos
- Para cada a, b pode assumir todos os valores distintos que estão à direita da primeira ocorrência de a no vetor
- Um conjunto pode ser usado manter o registro dos valores de a já processados
- O registro do valores à direita de a pode ser mantido através do uso de um histograma
- Com estas duas estruturas auxiliares, para cada elemento a_i do vetor, na ordem dada na entrada, são três passos a serem realizados
- O primeiro é a atualização do histograma, removendo o valor a_i se for a última ocorrência no vetor
- O segundo é verificar se a_i já foi processado ou não: em caso positivo, deve-se seguir para o próximo valor
- Por fim, se a_i não foi processado, devem ser acumulados na resposta todos os valores distintos ainda presentes no histograma

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

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

using namespace std;

long long solve(const vector<int>& as)
{
    map<int, int> R;    // Histograma à direita
    set<int> processed;
    long long ans = 0;

    for (const auto& a : as)
        ++R[a];

    for (const auto& a : as)
    {
        --R[a];

        if (R[a] == 0)
            R.erase(a);

```

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

```
21          if (processed.count(a))
22              continue;
23
24          ans += R.size();    // Soma os elementos distintos à direita
25          processed.insert(a);
26      }
27
28      return ans;
29  }
30
```

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

```
31 int main()
32 {
33     ios::sync_with_stdio(false);
34
35     int N;
36     cin >> N;
37
38     vector<int> as(N);
39
40     for (int i = 0; i < N; ++i)
41         cin >> as[i];
42
43     auto ans = solve(as);
44
45     cout << ans << '\n';
46
47     return 0;
48 }
```

Referências

1. URI 1897 – Jogo Esperto
2. UVA 12049 – Just Prune The List
3. AtCoder Beginner Contest 110 – Problem C: String Transformation
4. Codeforces Round #495 – Problem C: Sonya and Robots