Paradigmas de Resolução de Problemas

Busca Completa – Definição: Exercícios Resolvidos

Prof. Edson Alves - UnB/FGA

2020

Sumário

OJ 471 – Magic Numbers

Problema

Write a program that finds and displays all pairs of integers s_1 and s_2 such that:

1. neither s_1 nor s_2 have any digits repeated; and
2. $s_1/s_2 = N$, where N is a given integer;

Entrada e saída

Input

The input file consist a integer at the beginning indicating the number of test case followed by a blank line. Each test case consists of one line of input containing N.

Two input are separated by a blank line.

Output

For each input the output consists of a sequence of zero or more lines each containing '$s_1/s_2 = N$', where s_1, s_2 and N are the integers described above. When there are two or more solutions, sort them by increasing numerator values.

Two consecutive output set will separated by a blank line.

Exemplo de entradas e saídas

Sample Input

```
1

1234567890
```

Sample Output

```
1234567890 / 1 = 1234567890
2469135780 / 2 = 1234567890
4938271560 / 4 = 1234567890
6172839450 / 5 = 1234567890
8641975230 / 7 = 1234567890
9876543120 / 8 = 1234567890
```

Solução com complexidade $O(N)$

- Como o total de pares deve ser listado, a solução deve utilizar a busca completa
- Observe que não são informados os limites da entrada
- O pior caso aconteceria com $N = 1$, onde todos os pares (x, x), com $x \leq 10^{10}$, seriam válidos
- Assim, é assumido que as entradas levam a uma quantidade razoável de pares
- Da relação apresentada, s_2 é um múltiplo de N
- Assim, basta testar os valores $s_1 = 1, 2, 3, \ldots$, até que o produto $p = s_1 N \geq 10^{10}$
- Isto porque se p tem 11 ou mais dígitos, certamente ele terá duas ou mais repetições de um mesmo dígito

Solução com complexidade $O(N)$

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

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

int digits_count(ll x)
{
    int total = 0;

    do {
        x /= 10;

        ++total;
    } while (x);

    return total;
}

```

Solução com complexidade $O(N)$

```
20 bool has_repeated_digits(ll x)
21 {
22     bitset<10> used;
23     used.reset();
24 
25     while (x)
26     {
27         int d = x % 10;
28         x /= 10;
29 
30         if (used[d])
31             return true;
32 
33         used[d] = true;
34     }
35 
36     return false;
37 }
38 
```

Solução com complexidade $O(N)$

```
39 vector<ii> solve(ll N)
40 {
41     vector<ii> ans;
42
43     for (ll d = 1; digits_count(d*N) <= 10; ++d)
44     {
45         if (not has_repeated_digits(d) and not has_repeated_digits(d*N))
46             ans.push_back(ii(d*N, d));
47     }
48
49     return ans;
50 }
51
52 int main()
53 {
54     ios::sync_with_stdio(false);
55
56     int T;
57     cin >> T;
58
```

Solução com complexidade $O(N)$

```
59      for (int test = 1; test <= T; ++test)
60      {
61          ll N;
62          cin >> N;
63
64          auto ans = solve(N);
65
66          if (test > 1)
67              cout << '\n';
68
69          for (auto p : ans)
70              cout << p.first << " / " << p.second << " = " << N << '\n';
71      }
72
73      return 0;
74  }
```

Codechef – Little Elephant and Music

Problema

The Little Elephant from the Zoo of Lviv likes listening to music.

There are N songs, numbered from 1 to N, in his MP3-player. The song i is described by a pair of integers B_i and L_i – the band (represented as integer) that performed that song and the length of that song in seconds. The Little Elephant is going to listen all the songs exactly once in some order.

The sweetness of the song is equal to the product of the length of that song and the number of different bands listened before (including the current playing song).

Help the Little Elephant to find the order that maximizes the total sweetness of all N songs. Print that sweetness.

Entrada e saída

Input

The first line of the input contains single integer T, denoting the number of test cases. Then T test cases follow. The first line of each test case contains single integer N, denoting the number of the songs. The next N lines describe the songs in the MP3-player. The i-th line contains two space-sparated integers B_i and L_i.

Output

For each test, output the maximum total sweetness.

Constraints

- $1 \leq T \leq 5$
- $1 \leq N \leq 100000$ (10^5)
- $1 \leq B_i, L_i \leq 1000000000$ (10^9)

Exemplo de entradas e saídas

Sample Input

```
2
3
1 2
2 2
3 2
3
2 3
1 2
2 4
```

Sample Output

```
12
16
```

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

- Este problema pode ser resolvido por meio de um algoritmo guloso
- Considere uma ordenação arbitrária $\{m_1, m_2, \ldots, m_N\}$ das músicas
- Considere que $j = i + 1$ e que a banda da música i já apareceu ao menos uma vez antes de i e que a banda j não apareceu antes de j
- Se as duas músicas trocarem de posição a contribuição da música j será a mesma, pois sua duração não muda e o número de bandas que já apareceram será o mesmo
- Contudo, a contribuição da música i aumenta, pois o número de bandas distintas é incrementado
- Assim, as músicas devem ser ordenadas de tal maneira que as B bandas distintas apareçam nas primeiras B posições ao menos uma vez

Solução com complexidade $O(N)$

- Por outro lado, seja $i < j$ índices de duas músicas da mesma banda, com $i \leq B$ e a duração de m_i seja maior do que a duração de m_j
- A troca de ambas músicas de posição melhora a resposta, uma vez que B será multiplicado pela maior duração, e a menor duração será multiplicada pelo número de bandas distintas até i, que é um número menor ou igual a B
- Assim, a ordenação também deve colocar as músicas de menor duração de cada banda nas primeiras posições
- Por fim, $i < j$ são os índices de duas músicas consecutivas dentre as primeiras B posições, se a duração de m_i for maior do que m_j, a troca de posição das duas também melhora a resposta
- Portanto, as músicas devem ser ordenadas de tal modo que as músicas de menor duração de cada banda ocupe as B primeiras posições, em ordem de duração, e as demais ocupam as $N - B$, em posições quaisquer

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

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

using namespace std;
using ll = long long;

struct Song { ll b, l; };

ll solve(vector<Song>& ms)
{
    sort(ms.begin(), ms.end(), [](const Song& x, const Song& y) {
        return x.l < y.l;
    });

    set<ll> bs;
    vector<Song> pending;
    ll ans = 0;

```

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

```
18      for (const auto& m : ms)
19      {
20          if (bs.count(m.b) == 0)
21          {
22              bs.insert(m.b);
23              ans += (m.l * bs.size());
24          } else
25              pending.push_back(m);
26      }
27
28      for (const auto& m : pending)
29          ans += (m.l * bs.size());
30
31      return ans;
32  }
33
34  int main()
35  {
36      ios::sync_with_stdio(false);
37
```

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

```
38      int T;
39      cin >> T;
40
41      while (T--)
42      {
43          int N;
44          cin >> N;
45
46          vector<Song> ms(N);
47
48          for (int i = 0; i < N; ++i)
49             cin >> ms[i].b >> ms[i].l;
50
51          cout << solve(ms) << '\n';
52      }
53
54      return 0;
55  }
```

SPOJ – Saruman of Many Colours

Problema

"For I am Saruman the Wise, Saruman Ring-maker, Saruman of Many Colours!"

'I looked then and saw that his robes, which had seemed white, were not so, but were woven of all colours. And if he moved they shimmered and changed hue so that the eye was bewildered.' - Gandalf the Grey.

And so it was that Saruman decided to brand his Uruk-hai army with the many colours that he fancied. His method of branding his army was as follows.

He straps his army of N Uruk-hai onto chairs on a conveyor belt. This conveyor belt passes through his colouring-room, and can be moved forward or backward. The Uruk-hai are numbered 0 to $N-1$ according to the order in which they are seated. Saruman wishes that the i'th Uruk-hai be coloured with the colour $c[i]$.

Problema

Further, his colouring-room has space for exactly K chairs. Once the chosen K consecutive Uruk-hai are put into the room, a colour jet sprays all K of them with any fixed colour. The conveyor belt is not circular (which means that the $(N-1)$'th and the 0'th Uruk-hai are not consecutive). Note that Uruk-hai can be recoloured in this process.

Saruman wants to find out what is the minimum number of times that the jet needs to be used in order to colour his army in the required fashion. If it is not possible to colour the army in the required fashion, output -1.

Entrada e saída

Input

The first line contains the number of test-cases T.

Each test case consists of 2 lines. The first line contains two space-separated integers, N and K.

This is followed by a single like containing a string of length N, describing the colours of the army. The i'th character of the string denotes the colour of the i'th Uruk-hai in the army.

Entrada e saída

Output

Output T lines, one for each test case containing the answer for the corresponding test case. Remember if it is not possible to colour the army as required, output -1.

Constraints

$1 \leq T \leq 50$

$1 \leq K \leq N \leq 20,000$

The string c has length exactly N and contains only the characters `'a'`, ..., `'z'`.

Exemplo de entradas e saídas

Sample Input

```
2
3 2
rgg
3 3
rgg
```

Sample Output

```
2
-1
```

Solução com complexidade $O(N)$

- Este problema pode ser resolvido por meio de um algoritmo guloso, embora a identificação das escolhas e das restrições não sejam triviais
- Como cada processo de pintura modifica a cor de K Uruk-hais, a última pintura gerará um grupo de K soldados de mesma cor
- Assim, se o arranjo desejado não tiver um grupo como, no mínimo, K soldados vizinhos de mesma cor, não há solução
- Existindo tal grupo, é possível modificar a cor dos demais de forma gulosa, em duas etapas: à esquerda do grupo e à direita do grupo de K soldados de mesma cor
- A partir da esquerda, modifica-se a cor do soldado na posição i até $K-1$ vizinhos à sua direita, desde que tenham todos a mesma cor
- O mesmo processo pode ser feito a partir da direita, em sentido contrário
- Como cada soldados será avaliado, no máximo, duas vezes, este algoritmo tem complexidade $O(N)$

Solução AC com complexidade $O(N)$

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

using namespace std;

int solve(int N, int K, const string& c)
{
    int ans = 0, L = 0, len = 0;

    // Pinta de forma gulosa, em sequências de tamanho no máximo k
    while (L < N)
    {
        // Pinta na posição L
        ++ans;

        // No intervalo [L, R) todos tem mesma cor
        // Este intervalo deve ter no máximo K elementos
        int R = L + 1;

        while (R < N and R - L < K and c[R] == c[L])
            ++R;

```

Solução AC com complexidade $O(N)$

```
22          len = max(len, R - L);
23          L = R;
24      }
25
26      // Deve existir ao menos uma sequência de tamanho K
27      // (a última pincelada cria uma sequência de tamanho K)
28      if (len < K)
29          return -1;
30
31      return ans;
32  }
33
34  int main()
35  {
36      ios::sync_with_stdio(false);
37
38      int T;
39      cin >> T;
40
```

Solução AC com complexidade $O(N)$

```
41      while (T--)
42      {
43          int N, K;
44          cin >> N >> K;
45
46          string c;
47          cin >> c;
48
49          cout << solve(N, K, c) << '\n';
50      }
51
52      return 0;
53  }
```

Referências

1. OJ 471 – Magic Numbers
2. Codechef – Little Elephant and Music
3. SPOJ AMR12I – Saruman of Many Colours