Vetores

Fundamentos: Problemas Resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

UVA 11093 – Just Finish it Up

Problema

Along a circular track, there are N gas stations, which are numbered clockwise from 1 up to N. At station i, there are p_i gallons of petrol available. To race from station i to its clockwise neighbor one need q_i gallons of petrol. Consider a race where a car will start the race with an empty fuel tank. Your task is to find whether the car can complete the race from any of the stations or not. If it can then mention the smallest possible station i from which the lap can be completed.

Entrada e saída

Input

First line of the input contains one integer T the number of test cases. Each test case will start with a line containing one integer N, which denotes the number of gas stations. In the next few lines contain $2N$ integers. First N integers denote the values of p_is (petrol available at station i), subsequent N integers denote the value of q_is (amount of patrol needed to go to the next station in the clockwise direction).

Output

For each test case, output the case number in the format "`Case` c`:`" , where c is the case number starting from 1. Then display whether it is possible to complete a lap by a car with an empty tank or not. If it is not possible to complete the lap then display "`Not possible`". If possible, then display "`Possible from station` X", where X is the first possible station from which the car can complete the lap.

Exemplo de entradas e saídas

Constraints

- $T < 25$
- $N < 100001$

Sample Input

```
2
5
1 1 1 1 1
1 1 2 1 1
7
1 1 1 10 1 1 1
2 2 2 2 2 2 2
```

Sample Output

```
Case 1: Not possible
Case 2: Possible from station 4
```

Solução com complexidade $O(N^2)$

- A solução do problema com complexidade $O(N^2)$ consiste em simular, a partir de todos os postos possíveis, uma corrida, verificando se é possível completar uma volta ou não
- Importante: uma corrida equivale a uma volta completa, isto é, partir de um posto i e retornar a i
- A cada estação, é preciso ver se o valor de q_i é menor ou igual ao combustível disponível
- Se for, esta quantia é subtraída, e o combustível disponível no novo posto deve ser adicionado ao total
- Ao iniciar a corrida em i, o combustível inicial é igual a p_i
- Esta solução deveria levar ao TLE, dado que $N \leq 10^5$, mas os casos de teste do UVA são fracos e levam ao AC

Solução com complexidade $O(N)$

- Contudo, é possível resolver este problema com complexidade $O(N)$
- Para tal, é preciso verificar uma propriedade da solução
- Suponha que a corrida comece no posto i e que, no posto k, verifique-se que não é possível chegar ao posto $k+1$
- Na solução quadrática, a simulação recomeçaria no posto $i+1$
- Contudo, é possível começar direto no posto $k+1$, saltando todos os intermediários
- Eis a prova: se é possível chegar em $i+1$ a partir de i, começar a prova em $i+1$ significa ter ou a mesma quantidade de combustível resultante de se começar em (se $p_i = q_i$), ou ter menos combustível (se $p_i > q_i$)
- Assim, se partir de i significa não alcançar $k+1$, partir de $i+1$ levará ao mesmo obstáculo
- Portanto, cada posto será visitado, no máximo, duas vezes, levando a uma solução linear

Solução com complexidade $O(N)$

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solve(int N, const vector<int>& ps, const vector<int>& qs)
{
    int start = 0;

    while (start < N)
    {
        auto fuel = ps[start];
        auto stations = 0;
        auto now = start;
        auto next = (now + 1) % N;

        while (stations < N and qs[now] <= fuel)
        {
            ++stations;
            fuel -= qs[now];
            fuel += ps[next];
```

Solução com complexidade $O(N)$

```
22
23              now = next;
24              next = (next + 1) % N;
25          }
26
27          if (stations == N)
28              return start + 1;
29
30          if (next <= start)
31              break;
32
33          start = next;
34      }
35
36      return 0;
37  }
38
39  int main()
40  {
41      ios::sync_with_stdio(false);
42
```

Solução com complexidade $O(N)$

```
43      int T;
44      cin >> T;
45
46      for (int test = 1; test <= T; ++test)
47      {
48          int N;
49          cin >> N;
50
51          vector<int> ps(N), qs(N);
52
53          for (int i = 0; i < N; ++i)
54              cin >> ps[i];
55
56          for (int i = 0; i < N; ++i)
57              cin >> qs[i];
58
59          auto ans = solve(N, ps, qs);
60
61          cout << "Case " << test << ": ";
62
63          if (ans > 0)
```

Solução com complexidade $O(N)$

```
64              cout << "Possible from station " << ans << '\n';
65          else
66              cout << "Not possible\n";
67      }
68
69      return 0;
70  }
```

Codeforces Beta Round #98 – Problema B: Permutation

Problema

"Hey, it's homework time" – thought Polycarpus and of course he started with his favourite subject, IT. Polycarpus managed to solve all tasks but for the last one in 20 minutes. However, as he failed to solve the last task after some considerable time, the boy asked you to help him.

The sequence of n integers is called a permutation if it contains all integers from 1 to n exactly once.

You are given an arbitrary sequence $a_1, a_2, \ldots, a_n$ containing n integers. Each integer is not less than 1 and not greater than 5000. Determine what minimum number of elements Polycarpus needs to change to get a permutation (he should not delete or add numbers). In a single change he can modify any single sequence element (i. e. replace it with another integer).

Entrada e saída

Input

The first line of the input data contains an integer n $(1 \leq n \leq 5000)$ which represents how many numbers are in the sequence. The second line contains a sequence of integers a_i $(1 \leq a_i \leq 5000, 1 \leq i \leq n)$.

Output

Print the only number – the minimum number of changes needed to get the permutation.

Exemplo de entradas e saídas

Sample Input	Sample Output
3 3 1 2	0
2 2 2	1
5 5 3 3 3 1	2

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

- É possível resolver este problema com complexidade $O(N^2)$: basta, para cada valor $i = 1, 2, \ldots, N$, percorrer todo o vetor em busca deste valor
- Se o valor não for localizado, basta incrementar a resposta
- Contudo, há uma solução com complexidade $O(N \log N)$
- Basta criar um vetor auxiliar v com $N + 1$ elementos, todos iguais a zero
- Para cada elemento a do vetor da entrada, faça $v_a = 1$
- O número de elementos a serem alterados é igual ao total N, subtraído do número de encontrados (a soma de todos os valores armazenados em v)

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

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int solve(int N, vector<int>& as)
{
    vector<int> found(N + 1, 0);

    for (const auto& a : as)
        found[a] = 1;

    int total = 0;

    for (const auto& x : found)
        total += x;

    return N - total;
}

```

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

```
22 int main()
23 {
24     ios::sync_with_stdio(false);
25
26     int N;
27     cin >> N;
28
29     vector<int> as(N);
30
31     for (int i = 0; i < N; ++i)
32         cin >> as[i];
33
34     cout << solve(N, as) << '\n';
35
36     return 0;
37 }
```

Referências

1. UVA 11093 – Just Finish it Up
2. Codeforces Beta Round #98 – Problema B: Permutation