Busca e Ordenação

Algoritmos de ordenação linearítmicos: problemas resolvidos

Prof. Edson Alves

2020

Faculdade UnB Gama

Sumário

OJ 10327 – Flip Sort

Problema

Sorting in computer science is an important part. Almost every problem can be solved effeciently if sorted data are found. There are some excellent sorting algorithm which has already acheived the lower bound $n \log n$. In this problem we will also discuss about a new sorting approach. In this approach only one operation (Flip) is available and that is you can exchange two adjacent terms. If you think a while, you will see that it is always possible to sort a set of numbers in this way.

A set of integers will be given. Now using the above approach we want to sort the numbers in ascending order. You have to find out the minimum number of flips required. Such as to sort '1 2 3' we need no flip operation whether to sort '2 3 1' we need at least 2 flip operations.

Entrada e saída

Input

The input will start with a positive integer N ($N \leq 1000$). In next few lines there will be N integers.

Input will be terminated by EOF.

Output

For each data set print '`Minimum exchange operations :` M' where M is the minimum flip operations required to perform sorting. Use a seperate line for each case.

Exemplo de entradas e saídas

Exemplo de Entrada

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

Exemplo de Saída

```
Minimum exchange operations : 0
Minimum exchange operations : 2
```

Solução $O(N \log N)$

- Observe que o problema consiste, de fato, na contagem das inversões existentes no vetor a
- Uma inversão consiste em um par de índices (i, j), com $i < j$, tal que $a_i > a_j$
- Comparar todos os pares de índices possíveis leva a uma solução $O(N^2)$, a qual pode ser aceita, uma vez que $N \leq 10^3$
- Porém há uma solução mais eficiente, com complexidade $O(N \log N)$
- Basta observar que, durante a etapa de fusão do *MergeSort*, é possível contar as inversões de forma eficiente
- Considere dois vetores L e R a serem fundidos, com L à esquerda e R à direita
- Para cada elemento de L a ser copiado no vetor que conterá a fusão, haverão r inversões associadas a este elemento, onde r é o número de elementos do vetor R já copiados
- Logo a solução terá a mesma complexidade do *MergeSort*

Solução $O(N \log N)$

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

using namespace std;
using ll = long long;

template<typename RandIt>
ll merge(size_t N, RandIt first, RandIt middle, RandIt last)
{
    vector<typename iterator_traits<RandIt>::value_type> temp(N);
    auto it = first, jt = middle;
    auto k = 0;
    ll inversions = 0;

    while (it != middle and jt != last) {
        temp[k++] = min(*it, *jt);

        if (temp[k - 1] == *it)
            inversions += (jt - middle);

        temp[k - 1] == *it ? ++it : ++jt;
    }
```

Solução $O(N \log N)$

```
22
23      while (it != middle)
24      {
25          temp[k++] = *it++;
26          inversions += (last - middle);
27      }
28
29      while (jt != last)
30          temp[k++] = *jt++;
31
32      for (const auto& elem : temp)
33          *first++ = elem;
34
35      return inversions;
36  }
37
38  template<typename RandomAccessIterator>
39  ll mergesort(RandomAccessIterator first, RandomAccessIterator last)
40  {
41      auto N = last - first;
42
```

Solução $O(N \log N)$

```
43      if (N == 1)
44          return 0;
45
46      auto middle = first + (N + 1)/2;
47      ll inversions = 0;
48
49      inversions += mergesort(first, middle);
50      inversions += mergesort(middle, last);
51      inversions += merge(N, first, middle, last);
52
53      return inversions;
54  }
55
56  int main()
57  {
58      ios::sync_with_stdio(false);
59
60      int N;
61
```

Solução $O(N \log N)$

```
62      while (cin >> N)
63      {
64          vector<int> xs(N);
65
66          for (int i = 0; i < N; ++i)
67              cin >> xs[i];
68
69          auto ans = mergesort(xs.begin(), xs.end());
70
71          cout << "Minimum exchange operations : " << ans << '\n';
72      }
73
74      return 0;
75  }
```

OJ 10810 – Ultra-QuickSort

Problema

In this problem, you have to analyze a particular sorting algorithm. The algorithm processes a sequence of n distinct integers by swapping two adjacent sequence elements until the sequence is sorted in ascending order. For the input sequence

$$9\ 1\ 0\ 5\ 4,$$

Ultra-QuickSort produces the output

$$0\ 1\ 4\ 5\ 9.$$

Your task is to determine how many swap operations Ultra-QuickSort needs to perform in order to sort a given input sequence.

Entrada e saída

Input

The input contains several test cases. Every test case begins with a line that contains a single integer $n < 500\ 000$ – the length of the input sequence. Each of the the following n lines contains a single integer $0 \leq a[i] \leq 999\ 999\ 999$, the i-th input sequence element. Input is terminated by a sequence of length $n = 0$. This sequence must not be processed.

Output

For every input sequence, your program prints a single line containing an integer number op, the minimum number of swap operations necessary to sort the given input sequence.

Exemplo de entradas e saídas

Exemplo de Entrada	Exemplo de Saída
5 9 1 0 5 4 3 1 2 3 0	6 0

Solução $O(N \log N)$

- Este problema é idêntico ao anterior, com algumas diferenças pontuais
- A primeira diferença está na entrada, que termina com $N = 0$, ao invés de EOF
- A saída também é diferente, sem nenhuma outra informação a ser impressa além do número de inversões
- Como o número de elementos do vetor pode chegare a 10^5, o algoritmo quadrático leva ao TLE
- Exceto os pontos apresentados, ambas soluções são idênticas

Solução $O(N \log N)$

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

using namespace std;
using ll = long long;

template<typename RandIt>
ll merge(size_t N, RandIt first, RandIt middle, RandIt last)
{
    vector<typename iterator_traits<RandIt>::value_type> temp(N);
    auto it = first, jt = middle;
    auto k = 0;
    ll inversions = 0;

    while (it != middle and jt != last) {
        temp[k++] = min(*it, *jt);

        if (temp[k - 1] == *it)
            inversions += (jt - middle);

        temp[k - 1] == *it ? ++it : ++jt;
    }
```

Solução $O(N \log N)$

```
22
23      while (it != middle)
24      {
25          temp[k++] = *it++;
26          inversions += (last - middle);
27      }
28
29      while (jt != last)
30          temp[k++] = *jt++;
31
32      for (const auto& elem : temp)
33          *first++ = elem;
34
35      return inversions;
36  }
37
38  template<typename RandomAccessIterator>
39  ll mergesort(RandomAccessIterator first, RandomAccessIterator last)
40  {
41      auto N = last - first;
42
```

Solução $O(N \log N)$

```
43      if (N == 1)
44          return 0;
45
46      auto middle = first + (N + 1)/2;
47      ll inversions = 0;
48
49      inversions += mergesort(first, middle);
50      inversions += mergesort(middle, last);
51      inversions += merge(N, first, middle, last);
52
53      return inversions;
54  }
55
56  int main()
57  {
58      ios::sync_with_stdio(false);
59
60      int N;
61
```

Solução $O(N \log N)$

```
62      while (cin >> N, N)
63      {
64          vector<int> xs(N);
65
66          for (int i = 0; i < N; ++i)
67              cin >> xs[i];
68
69          auto ans = mergesort(xs.begin(), xs.end());
70
71          cout << ans << '\n';
72      }
73
74      return 0;
75  }
```

Referências

1. UVA 10327 – Flip Sort
2. UVA 10810 – Ultra-QuickSort