Paradigmas de Resolução de Problemas

Programação Dinâmica – Maior Subsequence Crescente: Exercícios Resolvidos

---

Prof. Edson Alves - UnB/FGA

2020

Sumário

SPOJ ELIS – Easy Longest Increasing Subsequence

Problema

Given a list of numbers A output the length of the longest increasing subsequence. An increasing subsequence is defined as a set $\{i_0, i_1, i_2, i_3, \ldots, i_k\}$ such that $0 \leq i_0 < i_1 < i_2 < i_3 < \ldots < i_k < N$ and $A[i_0] < A[i_1] < A[i_2] < \ldots < A[i_k]$. A longest increasing subsequence is a subsequence with the maximum k (length).

I.e. in the list $\{33, 11, 22, 44\}$ the subsequence $\{33, 44\}$ and $\{11\}$ are increasing subsequences while $\{11, 22, 44\}$ is the longest increasing subsequence.

Entrada e saída

Input

First line contain one number N ($1 \leq N \leq 10$) the length of the list A.

Second line contains N numbers ($1 \leq$ each number ≤ 20), the numbers in the list A separated by spaces.

Output

One line containing the lenght of the longest increasing subsequence in A.

Exemplo de entradas e saídas

Sample Input

```
5
1 4 2 4 3
```

Sample Output

```
3
```

Solução $O(N^2)$

- O título do problema não é um engodo: os limites do problema são tão pequenos que permitem mesmo uma solução de busca completa, avaliando todas as 2^N subsequências de A possíveis
- Uma solução de fácil codificação é a implementação $O(N^2)$ da LIS
- Nela, o estado $lis(i)$ corresponde ao tamanho da maior subsequência crescente que termina no elemento a_i
- O caso baso acontece quando $i = 1$ ($lis(1) = 1$)
- A transição é linear: todos os elementos a_j anteriores ($j < i$) devem ser avaliados
- Assim,

$$lis(i) = \max\{lis(j) + 1, 1\}, \quad \text{para todos } j < i \text{ tais que } a_j < a_i$$

Solução $O(N^2)$

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

using namespace std;

int solve(int N, const vector<int>& xs)
{
    vector<int> st(N, 1);

    st[0] = 1;

    for (int i = 1; i < N; ++i)
    {
        for (int j = i - 1; j >= 0; --j)
            if (xs[i] > xs[j])
                st[i] = max(st[i], st[j] + 1);
    }

    return *max_element(st.begin(), st.end());
}

```

Solução $O(N^2)$

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

OJ 10051 – Tower of Cubes

Problema

In this problem you are given N colorful cubes each having a distinct weight. Each face of a cube is colored with one color. Your job is to build a tower using the cubes you have subject to the following restrictions:

- Never put a heavier cube on a lighter one.
- The bottom face of every cube (except the bottom cube, which is lying on the floor) must have the same color as the top face of the cube below it.
- Construct the tallest tower possible.

Entrada e saída

Input

The input may contain multiple test cases. The first line of each test case contains an integer N $(1 \leq N \leq 500)$ indicating the number of cubes you are given. The ith $(1 \leq i \leq N)$ of the next N lines contains the description of the ith cube. A cube is described by giving the colors of its faces in the following order: front, back, left, right, top and bottom face. For your convenience colors are identified by integers in the range 1 to 100. You may assume that cubes are given in the increasing order of their weights, that is, cube 1 is the lightest and cube N is the heaviest.

The input terminates with a value 0 for N.

Entrada e saída

Output

For each test case in the input first print the test case number on a separate line as shown in the sample output. On the next line print the number of cubes in the tallest tower you have built. From the next line describe the cubes in your tower from top to bottom with one description per line. Each description contains an integer (giving the serial number of this cube in the input) followed by a single whitespace character and then the identification string (front, back, left, right, top or bottom) of the top face of the cube in the tower. Note that there may be multiple solutions and any one of them is acceptable.

Print a blank line between two successive test cases.

Exemplo de entradas e saídas

Sample Input

```
3
1 2 2 2 1 2
3 3 3 3 3 3
3 2 1 1 1 1
10
1 5 10 3 6 5
2 6 7 3 6 9
5 7 3 2 1 9
1 3 3 5 8 10
6 6 2 2 4 4
1 2 3 4 5 6
10 9 8 7 6 5
6 1 2 3 4 7
1 2 3 3 2 1
3 2 1 1 2 3
0
```

Sample Output

```
Case #1
2
2 front
3 front

Case #2
8
1 bottom
2 back
3 right
4 left
6 top
8 front
9 front
10 top
```

Solução $O(TN^2)$

- O problema pode ser reduzido à seis problemas de LIS
- Fazendo uma correspondência entre as faces de um cubo e os números de 0 a 5, na mesma ordem dada na entrada, o subproblema $lis(i, s)$ consiste em identificar a maior subsequência crescente válida que tem o i-ésimo elemento como menor elemento da sequência, cujo topo é a face s
- Como $N \leq 500$ é possível utilizar a implementação $O(N^2)$ da LIS
- Com a numeração proposta para as faces, a face oposta de s será $s + 1$, se s for par, ou $s - 1$, se s for ímpar
- Observe que uma sequência só pode ser ampliada caso a base do elemento coincida com o topo do elemento que ele ira sobrepor
- Também é preciso manter o registros dos elementos escolhidos e das faces utilizadas, para que a sequência possa ser reconstruída

Solução $O(TN^2)$

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

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

vector<string> fs { "front", "back", "left", "right", "top", "bottom" };

vector<is> solve(int N, const vector<vector<int>>& xs)
{
    vector<vector<int>> lis(N, vector<int>(6, 1));
    vector<vector<ii>> ps(N, vector<ii>(6, ii(0, 0)));

    int size = 1;
    auto best = ii(N - 1, 0);

    for (int i = N - 2; i >= 0; --i)
    {
        for (int t = 0; t < 6; ++t)
        {
            auto b = t % 2 ? t - 1 : t + 1;
```

Solução $O(TN^2)$

```
22
23              for (int j = i + 1; j < N; ++j)
24              {
25                  for (int s = 0; s < 6; ++s)
26                  {
27                      if (xs[i][b] == xs[j][s] and lis[j][s] + 1 > lis[i][t])
28                      {
29                          lis[i][t] = lis[j][s] + 1;
30                          ps[i][t] = ii(j, s);
31
32                          if (lis[i][t] > size)
33                          {
34                              size = lis[i][t];
35                              best = ii(i, t);
36                          }
37                      }
38                  }
39              }
40          }
41      }
42
```

Solução $O(TN^2)$

```
43      vector<is> ans(size);
44
45      for (int k = 0; k < size; ++k)
46      {
47          auto [i, s] = best;
48
49          ans[k] = is(i + 1, fs[s]);
50          best = ps[i][s];
51      }
52
53      return ans;
54  }
55
56  int main()
57  {
58      ios::sync_with_stdio(false);
59      int N, test = 0;
60
61      while (cin >> N, N)
62      {
63          vector<vector<int>> xs(N, vector<int>(6));
```

Solução $O(TN^2)$

```
64
65          for (int i = 0; i < N; ++i)
66              for (int j = 0; j < 6; ++j)
67                  cin >> xs[i][j];
68
69          auto ans = solve(N, xs);
70
71          if (test)
72              cout << '\n';
73
74          cout << "Case #" << ++test << '\n';
75
76          cout << ans.size() << '\n';
77
78          for (auto [i, side] : ans)
79              cout << i << ' ' << side << '\n';
80      }
81
82      return 0;
83  }
```

Codeforces Round # 198 (Div. 2) – Problem D: Bubble Sort Graph

Problema

Iahub recently has learned Bubble Sort, an algorithm that is used to sort a permutation with n elements $a_1, a_2, \ldots, a_n$ in ascending order. He is bored of this so simple algorithm, so he invents his own graph. The graph (let's call it G) initially has n vertices and 0 edges. During Bubble Sort execution, edges appear as described in the following algorithm (pseudocode).

Problema

```
procedure bubbleSortGraph()
    build a graph G with n vertices and 0 edges
    repeat
        swapped = false
        for i = 1 to n - 1 inclusive do:
            if a[i] > a[i + 1] then
                add an undirected edge in G between a[i] and a[i + 1]
                swap( a[i], a[i + 1] )
                swapped = true
            end if
        end for
    until not swapped
    /* repeat the algorithm as long as swapped value is true. */
end procedure
```

Problema

For a graph, an independent set is a set of vertices in a graph, no two of which are adjacent (so there are no edges between vertices of an independent set). A maximum independent set is an independent set which has maximum cardinality. Given the permutation, find the size of the maximum independent set of graph G, if we use such permutation as the premutation a in procedure `bubbleSortGraph`.

Entrada e saída

Input

The first line of the input contains an integer n ($2 \leq n \leq 10^5$). The next line contains n distinct integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq n$).

Output

Output a single integer – the answer to the problem.

Exemplo de entradas e saídas

Sample Input

```
3
3 1 2
```

Sample Output

```
2
```

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

- Gerar o grafo G por meio da execução do código do *bubblesort* apresentado leva a um veredito TLE, uma vez que, no pior caso, há $O(N^2)$ arestas (sequência em ordem decrescente)
- Mesmo que fosse possível construir o grafo G em tempo hábil, o maior conjunto independente é um problema *NP-Hard*, e como $N \leq 10^5$, novamente o veredito seria TLE
- O que deve ser observado é que não existirá uma aresta entre a_i e a_j se $a_i < a_j$, com $i < j$
- Observe que, pela transitividade, se não existe uma aresta entre a_i e a_j, e também não há aresta entre a_j e a_k, não haverá uma aresta entre a_i e a_k

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

- Deste modo, um conjunto independente em G será uma sequência crescente de $a = \{a_1, a_2, \ldots, a_N\}$
- A resposta do problema, portanto, será a maior subsequência crescente de a
- Dados os limites do problema, a implementação quadrática levaria ao TLE
- Portanto, o problema deve ser resolvido pela implementação linearítmica da LIS

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

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

using namespace std;

const int oo { 2000000010 };

int solve(int N, const vector<int>& as)
{
    vector<int> lis(N + 1, oo);
    lis[0] = 0;

    auto ans = 0;

    for (int i = 0; i < N; ++i)
    {
        auto it = lower_bound(lis.begin(), lis.end(), as[i]);
        auto pos = (int) (it - lis.begin());

        ans = max(ans, pos);
        lis[pos] = min(as[i], lis[pos]);
    }
```

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

```
23      return ans;
24  }
25
26  int main()
27  {
28      ios::sync_with_stdio(false);
29
30      int N;
31      cin >> N;
32
33      vector<int> as(N);
34
35      for (int i = 0; i < N; ++i)
36          cin >> as[i];
37
38      auto ans = solve(N, as);
39
40      cout << ans << '\n';
41
42      return 0;
43  }
```

Referências

1. SPOJ ELIS – Easy Longest Increasing Subsequence
2. OJ 10051 – Tower of Cubes
3. Codeforces Round #198 (Div. 2) – Problem D: Bubble Sort Graph