

# Paradigmas de Resolução de Problemas

Programação Dinâmica – Maior Subsequence Crescente:  
Exercícios Resolvidos

---

Prof. Edson Alves - UnB/FGA

2020

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

## **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  $\{i_0, i_1, i_2, i_3, \dots, i_k\}$  such that  $0 \leq i_0 < i_1 < i_2 < i_3 < \dots < i_k < N$  and  $A[i_0] < A[i_1] < A[i_2] < \dots < A[i_k]$ . 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.

### Input

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

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

### Output

One line containing the length 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 base 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 $O(N^2)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int solve(int N, const vector<int>& xs)
6 {
7     vector<int> st(N, 1);
8
9     st[0] = 1;
10
11     for (int i = 1; i < N; ++i)
12     {
13         for (int j = i - 1; j >= 0; --j)
14             if (xs[i] > xs[j])
15                 st[i] = max(st[i], st[j] + 1);
16     }
17
18     return *max_element(st.begin(), st.end());
19 }
20
```



## 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.

## 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  $i$ th ( $1 \leq i \leq N$ ) of the next  $N$  lines contains the description of the  $i$ th 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  $N$ .

## 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 $O(TN^2)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ii = pair<int, int>;
5 using is = pair<int, string>;
6
7 vector<string> fs { "front", "back", "left", "right", "top", "bottom" };
8
9 vector<is> solve(int N, const vector<vector<int>>& xs)
10 {
11     vector<vector<int>> lis(N, vector<int>(6, 1));
12     vector<vector<ii>> ps(N, vector<ii>(6, ii(0, 0)));
13
14     int size = 1;
15     auto best = ii(N - 1, 0);
16
17     for (int i = N - 2; i >= 0; --i)
18     {
19         for (int t = 0; t < 6; ++t)
20         {
21             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**

---

Iahub recently has learned Bubble Sort, an algorithm that is used to sort a permutation with  $n$  elements  $a_1, a_2, \dots, a_n$  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 permutation  $a$  in procedure `bubbleSortGraph`.

## Input

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

## 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 $O(1)$

- Observe que a tentativa de se computar o valor exato de  $1378^n$  leva a dois problemas:
  1. a complexidade da solução seria  $O(n)$ , e  $n = 10^9$  no pior caso
  2. porém esta complexidade assume que o cálculo de cada produto pode ser feito em  $O(1)$ , que não é o caso pois tais número crescem exponencialmente, o que leva também a problemas de memória
- Ainda assim, computar tais valores para os primeiros valores de  $n$  (por exemplo,  $n = 20$ ), pode revelar padrões na solução que não são óbvios à primeira vista
- O primeiro padrão que surge é que os últimos dígitos dos resultados, a partir de  $n = 1$ , formam uma sequência periódica:

$8, 2, 4, 6, 8, 2, 4, 6, \dots$

- Cuidado com o *corner case*  $n = 0$ : neste caso, a resposta deve ser igual a um
- Exceto no caso especial, uma operação de resto da divisão determina o resultado correto, de modo que a solução tem complexidade  $O(1)$

# Solução com complexidade $O(1)$

```
1 // Codeforces Round #198 - Problem D: Bubble Sort Graph (https://codeforces.com/p)
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 using ll = long long;
7 using ii = pair<ll, ll>;
8
9 #define REP(i, a, b) for (int (i) = (a); (i) <= (b); (i)++)
10 #define REPN(i, N) for (int (i) = 0; (i) < N; (i)++)
11
12 const int MAX { 2000010 };
13 const int oo { 2000000010 };
14
15 int as[MAX], lis[MAX];
16
17 // Não há arestas se os valores estão em ordem crescente
18 // Logo o conjunto procurado equivale a LIS
19 void solve(int N)
```

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