# Algorithmique effective – Examen
Mardi 30 mai 2006, 14h00 – 19h00

## 1 Matriochka

Russian nesting dolls are brightly painted hollow wooden figures. The dolls in a set have roughly the same shape, typically humanoid, but different sizes. When the set is assembled, the biggest doll contains the second-biggest doll, the second-biggest contains the third-biggest, and so on. We can approximate the shape of a doll as a cylinder of height $h$, diameter $d$, and wall thickness $w$. Such a doll would have a hollow of height $h - 2w$ and diameter $d - 2w$.

Boris and Natasha each has a set of dolls. The sets are nearly identical; each has the same number of dolls, which look the same but differ in their dimensions. Last night Boris and Natasha were playing with their dolls and left them in the living room. Their mother tidied them away, dumping them all in one box. Can you help Boris and Natasha separate their sets of dolls?

Standard Input will consist of several test cases. The first line of each test case will contain $n$, the number of dolls in each set ($1 < n \leq 100$). $2n$ lines follow; each gives the dimensions, $h, d, w$ of a different doll ($h, d \geq 2w > 0$). A line containing 0 follows the last test case.

For each test case, separate the dolls into two sets of $n$ nesting dolls such that, within each set, the dolls fit within each other, standing straight up, as described above. The first $n$ lines of output should give the dimensions of the dolls in one set, in decreasing order by height. The next line should contain a single hyphen, "-". The next $n$ lines should give the dimensions of the dolls in the second set, also in decreasing order by height. There will always be a solution. If there are many solutions, any will do. Output an empty line between test cases.

**Sample Input**

```
3
100 100 3
97 97 3
94 94 3
91 91 3
88 88 3
85 85 3
5
100 100 1
97 97 3
98 98 1
96 96 1
94 94 1
92 92 1
90 90 1
88 88 1
86 86 1
84 84 1
0
```

**Possible Output**

```
100 100 3
94 94 3
88 88 3
-
97 97 3
91 91 3
85 85 3

100 100 1
98 98 1
96 96 1
94 94 1
92 92 1
-
97 97 3
90 90 1
88 88 1
86 86 1
84 84 1
```

## 2 Terrific traffic

A city has $n$ intersections and $m$ bidirectional roads connecting pairs of intersections. Each road has a certain traffic flow capacity, measured in cars per minute. There is a path from every intersection to every other intersection along some sequence of roads. The road maintenance department is over budget and needs to close as many roads as possible without disconnecting any intersections. They want to do it in such a way that the minimum capacity among all of the remaining roads is as large as possible.

**Input**

The first line of input gives the number of cases, $N$. $N$ test cases follow. Each one starts with a line containing $n$ $(0 < n \leq 100)$ and $m$ $(0 < m \leq 10{,}000)$. The next $m$ lines will describe the $m$ roads, each one using 3 integers, $u$, $v$ and $c$ $(0 \leq u, v < n)$, $(0 < c \leq 1{,}000)$. $u$ and $v$ are the endpoints of the road and $c$ is its capacity.

**Output**

For each test case, output one line containing `Case #x:` followed by the capacity of the minimum-capacity remaining road.

**Sample Input**

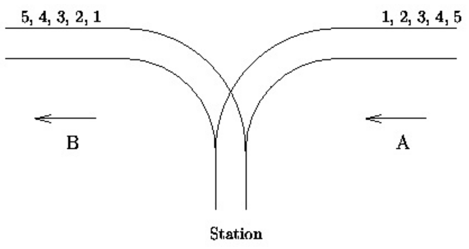
```
2
2 3
0 1 10
0 1 20
0 0 30
4 5
0 1 1
3 1 2
1 2 3
2 3 4
0 2 5
```

**Sample Output**

```
Case #1: 20
Case #2: 3
```

## 3 A famous station

There is a famous railway station in PopPush City. Country there is incredibly hilly. The station was built in last century. Unfortunately, funds were extremely limited that time. It was possible to establish only a surface track. Moreover, it turned out that the station could be only a dead-end one (see picture) and due to lack of available space it could have only one track.



The local tradition is that every train arriving from the direction $A$ continues in the direction $B$ with coaches reorganized in some way. Assume that the train arriving from the direction $A$ has $N \leq 1{,}000$ coaches numbered in increasing order $1, 2, \ldots, N$. The chief for train reorganizations must know whether it is possible to marshal coaches continuing in the direction $B$ so that their order will be $a_1, \ldots, a_N$. Help him and write a program that decides whether it is possible to get the required order of coaches. You can assume that single coaches can be disconnected from the train before they enter the station and that they can move themselves until they are on the track in the direction $B$. You can also suppose that at any time there can be located as many coaches as necessary in the station. But once a coach has entered the station it cannot return to the track in the direction $A$ and also once it has left the station in the direction $B$ it cannot return back to the station.

**Input**

The input file consists of blocks of lines. Each block except the last describes one train and possibly more requirements for its reorganization. In the first line of the block there is the integer $N$ described above. In each of the next lines of the block there is a permutation of $\{1, 2, \ldots, N\}$. The last line of the block contains just 0.

The last block consists of just one line containing 0.

**Output**

The output file contains the lines corresponding to the lines with permutations in the input file. A line of the output file contains `Yes` if it is possible to marshal the coaches in the order required on the corresponding line of the input file. Otherwise it contains `No`. In addition, there is one empty line after the lines corresponding to one block of the input file. There is no line in the output file corresponding to the last “null” block of the input file.

**Sample Input**

```
5
1 2 3 4 5
5 4 1 2 3
0
6
6 5 4 3 2 1
0
0
```

**Sample Output**

```
Yes
No

Yes
```

## 4 Sorting

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. 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 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.

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.

**Sample Input**

```
5
9
1
0
5
4
3
1
2
3
0
```

**Output for Sample Input**

```
6
0
```

## 5 Type equivalence

In programming language design circles, there has been much debate about the merits of "structural equivalence" vs. "name equivalence" for type matching. Pascal purports to have "name equivalence", but it doesn't; C purports to have structural equivalence, but it doesn't. Algol 68, the Latin of programming languages, has pure structural equivalence. A simplified syntax for an Algol 68 type definition is as follows:

```
type_def -> type T = type_expr
type_expr -> T | int | real | char | struct ( field_defs )
field_defs -> T | field_defs T
```

In this syntax, `T` is a programmer-defined type name (in this problem, for simplicity, a single upper case letter). Plain text symbols appear literally in the input, and zero or more spaces or a comma (`,`) may appear where there are spaces in the syntax.

Algol 68 type equivalence say that two types are equivalent if they are the same primitive type or they are both structures containing equivalent types in the same order.

Input consists of several test cases. Each test case is a sequence of Algol 68 definitions, as described above, one per line. A line containing "-" separates test cases. A line containing "--" follows the last test case. The output for each case will consist of several lines; each line should contain a list of type names, all of which represent equivalent types. Each type name should appear on exactly one line of output, and the number of output lines should be minimized. The names in each list should be in alphabetical order; the lines of output should also be in alphabetical order. Output an empty line between test cases.

**Sample Input**

```
type A = int
type B = A
type C = int
type X = struct(A B)
type Y = struct(B A)
type Z = struct(A Z)
type S = struct(A S)
type W = struct(B R)
type R = struct(C W)
--
```

**Output for Sample Input**

```
A B C
R S W Z
X Y
```

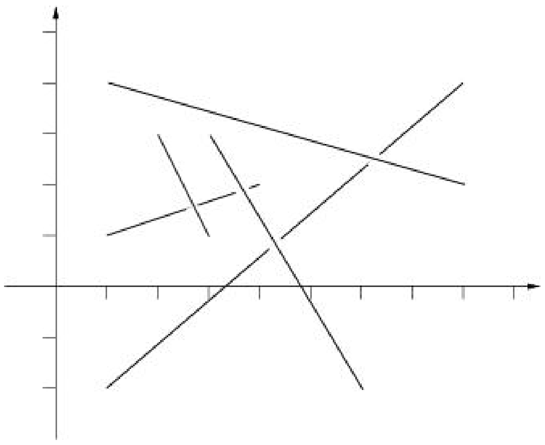
## 6 Mikado

Stan has $n$ sticks of various length. He throws them one at a time on the floor in a random way. After finishing throwing, Stan tries to find the top sticks, that is these sticks such that there is no stick on top of them. Stan has noticed that the last thrown stick is always on top but he wants to know all the sticks that are on top. Stan sticks are very, very thin such that their thickness can be neglected.

Input consists of a number of cases. The data for each case start with $1 \leq n \leq 100{,}000$, the number of sticks for this case. The following $n$ lines contain four numbers each, these numbers are the planar coordinates of the endpoints of one stick. The sticks are listed in the order in which Stan has thrown them. You may assume that there are no more than 1,000 top sticks. The input is ended by the case with $n = 0$. This case should not be processed.

For each input case, print one line of output listing the top sticks in the format given in the sample. The top sticks should be listed in order in which they were thrown.

The picture below illustrates the first case from input.



**Sample input**

```
5
1 1 4 2
2 3 3 1
1 -2.0 8 4
1 4 8 2
3 3 6 -2.0
3
0 0 1 1
1 0 2 1
2 0 3 1
0
```

**Output for sample input**

```
Top sticks: 2, 4, 5.
Top sticks: 1, 2, 3.
```

## 7 Pattern matching

MegaFirm Inc. has created a set of patterns to aid its telephone help-desk operators in responding to customers. A pattern is a phrase consisting of words and placeholders. A word is simply a string of letters. A placeholder is a word enclosed in angle brackets (that is `< ... >`). A phrase matches a pattern if each placeholder in the pattern can be systematically replaced by a word so as to make the pattern and phrase equal. By "systematically replaced" we mean that all placeholders enclosing the same word are replaced by the same word. For example, the phrase

```
to be or not to be
```

matches the pattern

```
<foo> be <bar> not <foo> <baf>
```

because we can replace `<foo>` by `to`, `<bar>` by `or`, and `<baf>` by `be`. Given two patterns, you are to find a phrase that matches both.

The first line of input contains $n$, the number of test cases. Each test case consists of two lines of input; each a pattern. Patterns consist of lowercase words, and placeholders containing lowercase words. No pattern exceeds 100 characters. Words contain at most 16 characters. A single space separates adjacent words and placeholders.

For each test case, output a phrase that matches both patterns. If several phrases match, any will do. If no phrase matches, output a line containing "`-`" (a single minus sign).

**Sample Input**

```
3
how now brown <animal>
<foo> now <color> cow
who are you
<a> <b> <a>
<a> b
c <a>
```

**Possible Output for Sample Input**

```
how now brown cow
-
c b
```