ACM South American Regional

Collegiate Programming Contest

Contest Session

December 13, 1997

Institute of Computing

University of Campinas, Brasil

Problem A

The JustaPox Language

File Names

- **Source File:** justa.pas or justa.c or justa.cpp
- **Input File:** justa.in
- **Output File:** justa.out

Statement of the Problem

A new programming language is being designed. One of its new features is related to the specification of arrays. The language deals only with arrays of integers, and there are two ways of specifying new arrays:

1. by explicit declaration of its size. For example:

$$A \text{ is } [1]$$

$$B\ is\ [3]$$

   These sentences specify A and B as arrays of 1 and 3 elements, respectively. This implies that any sequence with the specified size is a possible value for the array.

2. by inclusion of previously defined arrays. For example:

$$C \text{ is } A \quad B$$

$$D \text{ is } A \quad A$$

$$SEQ \text{ is } [1] \quad A \quad [4] \quad C \quad B \quad A$$

   The size for arrays specified in this manner is determined by the size of its sub-arrays. Besides, if a specification includes the same array more than once, the subsequences in the array corresponding to the repeated parts have to hold the same value.
   For the aforementioned specification of A and B, we have C with size 4, D with size 2, and SEQ with size 14.

Assignments are expressed in the language by the following statement:

$$< \text{array} >=< \text{sequence} \quad \text{of} \quad \text{integers} >$$

For example,

$$B = 3 \quad 4 \quad 15$$

is a valid assignment for B. Notice that the assignment $B = 3 \;\; 4$ would be inconsistent with B's specification (which stated that B would hold a sequence of integers with size 3). The assignment $D = 1 \;\; 2$ would also be inconsistent, since D's specification implies that D is formed by the repetition of a size 1 sequence of integers. $D = 1 \;\; 1$ is an example of a consistent assignment for D.

In other words, for the specification

$$S \text{ is } S_1 \;\; S_2 \;\; \ldots \;\; S_m,$$

the assignment

$$S \;\; = \;\; i_1 \; i_2 \; \ldots \; i_n$$

is said to be **consistent** if and only if S is a sequence

$$i_{11} i_{12} \ldots i_{1k_1} i_{21} \ldots i_{2k_2} \ldots i_{m1} i_{m2} \ldots i_{mk_m},$$

where

- $k_i = |S_i| \;\; (1 \leq i \leq m)$

and

- for every previously defined subsequences S_a, S_b $(1 \leq a, b \leq m)$ such that $S_a = S_b$ we have $(i_{a1}, i_{a2}, \ldots, i_{ak_a}) = (i_{b1}, i_{b2}, \ldots, i_{bk_b})$.

The language designers want you to cooperate with the compiler construction effort: you have to write a program that receives as input a collection of array specifications and a sequence of assignments, and produces as output the list of inconsistent assignments.

Input Format

The input file is composed by a list of array specifications followed by a list of assignments.

An array specification has the following format:
$< id >$ is $d_1 \;\; d_2 \;\; \ldots \;\; d_m$.

$< id >$ is a sequence of characters representing the array name. You can assume that (1) the length of this sequence is between 1 and 32 and (2) the sequence follows the rules for identifier names in the programming language that you are using.

d_i is

either $[k]$ where k is a strictly positive integer (there may be spaces between k and its surrounding brackets)
or $< id >$, where $< id >$ is the name of a previously specified array.

The sequence of d_i's is terminated by a period.

An assignment sepcification has the following format:
$< id > \quad = \quad i_1 \;\; i_2 \;\; \ldots \;\; i_n.$

$< id >$ is the name of a previously specified array and i_j are positive integers. The list of i_j's is terminated by a period.

In both types of specifications, an arbritrary number of spaces or empty lines can appear between the tokens.

Output Format

The output file must contain the list of assignment specifications from the input file that are inconsistent. The output for each of the inconsistent assignments should be its specification as it appeared in the input file.

Examples

Suppose the following input:

```
A is [1].
B is [3].
C is A B.
SEQ is [1 ] A [4] B A.
A = 10.
B = 1 2 3.
C = 1 1 2 3 4.
SEQ = 1 10 1 1 1 1 2 2 2 10.
SEQ = 1 10 1 1 1 1 1 2 3         9.
```

Then the corresponding output is

```
C = 1 1 2 3 4.
SEQ = 1 10 1 1 1 1 1 2 3         9.
```

Problem B

Driving in City Squares

File Names

- **Source File:** city.pas or city.c or city.cpp
- **Input File:** city.in
- **Output File:** city.out

Statement of the Problem

The city *AllSquared* has been designed by many famous architects and engineers. The city lays in a retangular area of $n * m$ squared miles (n refers to the retangle basis, and m to its height). The streets are horizontal divisions distributed uniformly at every one mile, and the avenues are vertical divisions also uniformly distributed at every one mile. Streets are numbered from north to south, starting from zero; avenues from west to east, also starting from zero.

The figure (a) below portrays the city layout for $n = 11$ and $m = 8$. The city is also

divided in horizontal and vertical strips, with the intersection of these strips defining the city subregions (counties). Figure (b) pictures a city divided into 12 counties.

Every county is associated with a vehicle circulation fee, which should be paid every time a car enters the county. If the car origin is a street or avenue delimiting the county, there is no fee. For example, in the above figure a car whose origin is street = 2 and avenue = 3 and whose target is street = 2 and avenue = 9 will have to pay only the county crossing fee relative to avenue 6.

You should write a program that has the following input:

- two strictly positive integers n and m specifying the city dimensions;
- two positive integers h and v specifying the number of horizontal and vertical strips, respectively (for the city in the figure, $h = 4$, $v = 3$);
- $h - 1$ integers $s_1, \ldots, s_{h-1} (1 \leq s_i \leq n - 1)$ which denote the streets in which there is a county division line (in the previous figure, the numbers are 1, 6, and 7);
- $v - 1$ integers $a_1, \ldots, a_{v-1} (1 \leq a_i \leq m - 1)$ which denote the avenues in which there is a county division line (in the the previous figure, the numbers are 3 and 6);
- the circulation fees for each county in the city, i.e., $h * v$ positive real numbers

$$\begin{array}{l} p_{11}, p_{12}, \ldots, p_{1v} \\ p_{21}, p_{22}, \ldots, p_{2v} \\ \ldots \\ p_{h1}, p_{h2}, \ldots, p_{hv} \end{array}$$

  where p_{ij} is the circulation fee for the county delimited by a_{i-1} (west), a_{i+1} (east), s_{j-1} (north), and s_{j+1} (south);
- two pairs of positive integers $\text{local}_1 = (\text{str}_1, \text{av}_1)$ e $\text{local}_2 = (\text{str}_2, \text{av}_2)$

The output should be the smallest possible price to pay for going from local_1 to local_2.

Input Format

The input format may contain several instances of the problem. Each instance is terminated by a line starting with % (percentage symbol).

The description for one instance has the following format:

n m
h v
$s_1 \;\; s_2 \;\; \ldots \;\; s_{h-1}$
$a_1 \;\; a_2 \;\; \ldots \;\; a_{v-1}$
$p_{11} \;\; p_{12} \;\; \ldots \;\; p_{1v}$
$p_{21} \;\; p_{22} \;\; \ldots \;\; p_{2v}$
$\ldots$
$p_{h1} \;\; p_{h2} \;\; \ldots \;\; p_{hv}$
$w_1 \;\; t_1 \;\; w_2 \;\; t_2$
%

where

- n, m, h, v are integers such that $0 < n, 0 < m, 0 < h \leq n, 0 < v \leq m$;

- $s_1, \ldots, s_{h-1}$ are such that $1 \leq s_i \leq n-1$ for $1 \leq i \leq h-1$;
- $a_1, \ldots, a_{v-1}$ are such that $1 \leq a_i \leq m-1)$ for $1 \leq v-1$;
- p_{ij} are positive real numbers;
- w_1, t_1, w_2, t_2 are integers such that $0 \leq w_1, w_2 \leq n$ and $0 \leq t_1, t_2 \leq m$. (w_1, t_1) is the origin and $(w_2, t_2$ is the target point for the problem;
- there can be an arbitrary number of spaces or empty lines between the numbers in the file.

Output Format

For an input file with k instances of the problem, the output file should follow the format:

c_1
c_2
...
c_k
where c_i is the result for the i^{th} instance in the input file.

Examples

Suppose the following input:

```
100 100
3 4
30 60
10 70 80
5 100 1 20
1 100 1 20
1 1 1 20
5 5 10 75
%
11 8
4 3
1 6 7
3 6
10 10 10
10 10 10
10 10 10
10 10 10
2 3 2 9
%
```

Then the corresponding output is

```
6
10
```

Problem C

Hamiltonian Cycle

File Names

- **Source File:** cycle.pas or cycle.c or cycle.cpp
- **Input File:** cycle.in
- **Output File:** cycle.out

Statement of the Problem

A few definitions first:

Definition 1 A graph $G = (V, E)$ is called "dense" if for each pair of non-adjacent vertices u and v, $d(u) + d(v) \geq n$ where $n = |V|$ and $d(\bullet)$ denotes the degree of the vertex $\bullet$.

Definition 2 A "Hamiltonian cycle" on G is a sequence of vertices $(v_{i_1} v_{i_2} \dots v_{i_n} v_{i_1})$ such that $v_{i_\ell} \neq v_{i_h}$ for all $\ell \neq h$ and $\{v_{i_\ell}, v_{i_\ell}\}$ is an edge of G.

The problem is: write a program that, given a dense graph $G = (V, E)$ as input, determines whether G admits a Hamiltonian cycle on G and outputs that cycle, if there is one, or outputs "N" if there is none.

Input Format

A file containing descriptions of graphs, each one ending with a %, in the form:
n_1
u_{i_1} u_{j_1}
u_{i_2} u_{j_2}
etc...
%
n_2
u_{i_1} u_{j_1}
u_{i_2} u_{j_2}
etc...
%
where n_i is the number of vertices and u_{i_h} u_{i_ℓ} are integers between 1 and n indicating that there exists an edge between vertex u_{i_h} and u_{i_ℓ}.

Output Format

The output file must contain the sequence of vertices that form a Hamiltonian cycle in the form:
$u_{i_1}\ u_{i_2}\ u_{i_3}\ \ldots$
or containing:
N

Examples

Suppose the following input:

```
4
1 2
2 3
2 4
3 4
3 1
%
6
1 2
1 3
1 6
3 2
3 4
5 2
5 4
6 5
6 4
%
```

The corresponding output could be:

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

Problem D

Monkeys in a Regular Forest

File Names

- **Source File:** monkey.pas or monkey.c or monkey.cpp
- **Input File:** monkey.in
- **Output File:** monkey.out

Statement of the Problem

Consider the situation of an ideal forest, where trees grow on a regular finite euclidean lattice. At every site only one tree grows, and it can be of one among n species. Each species is denoted by a single character ($\{A, B, C, \ldots\}$ are valid species, for instance). Two trees of the same species are considered neighbors if the maximum absolute difference between their coordinates is one.

Families of (rather specialized) monkeys are released, one at a time, in this euclidean forest. Each family will occupy all neighboring tress of a single species which have not been taken yet by another family.

Given the map of the forest, build the map of the monkeys families, starting with "1" and numbering them consecutively.

Input Format

File monkey.in has the lines of a matrix of single characters, separated by single blank spaces.

Next matrices (each matrix is a different instance to the problem) will be preceded by a line with a single "%" character and then the same structure as before.

Output Format

File monkey.out has to show lines of integers separated by as many blank spaces as required to align columns to the right.

The solution to each instance must be finished by a line with a single "%" character.

Examples

Suppose the following input:

```
A B D E C C D
F F W D D D D
P W E W W W W
```

```
%
a A b B c d E t
a a a a a c c t
e f g h c a a t
```

Then the corresponding output is

```
1  2  3  4  5  5  3
6  6  9  3  3  3  3
8  9 10  9  9  9  9
%
1  2  3  4  5  6  7 12
1  1  1  1  1  5  5 12
8  9 10 11  5  1  1 12
%
```

Problem E

Codebreakers

File Names

- **Source File:** code.pas or code.c or code.cpp
- **Input File:** code.in
- **Output File:** code.out

Statement of the Problem

Alice and Bob use a key, a code, to communicate in privacy. This code is just an ordered sequence of m different characters out of a set S of n characters. For example, if $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $m = 4$, then one possible key is $(3, 8, 1, 5)$.

No one knows the key except Alice and Bob. However, Steve the Spy knows the set S and length m of the key, and he will try to deduce the key by posing different guesses of the code. To each guess, Alice or Bob will reply with an answer of the form (c, w), where c is the number of correctly placed digits in Steve's guess, and w is the number of digits in the guess which are present in the code but in the wrong position, i.e., they are misplaced.

For instance, these are guesses and answers for the aforementioned code.

```
(1,1,1,1) -> (1,0)
(2,2,2,2) -> (0,0)
(9,8,3,1) -> (1,2)
```

Note that a correct guess will produce the answer $(m, 0)$. You will be given S, m and a list (of arbitrary length) of trials and answers, and your program has to take either one of the following courses of action:

1. claim that the given information is not enough to find out the correct code, or
2. give the correct code, i.e., the code for which the answer is $(m, 0)$.

Input Format

First line of file code.in has the set S with its elements separated by blank spaces (without brackets). Second line has the value of m. The following lines are the guesses, in the form "$(g_1, \ldots, g_m)$" with $g_i \in S$ for all $1 \leq i \leq m$, the "->" symbol and the answer, in the form "(c, w)". Each instance ends with a line with a single "%" character; next instance has the same presentation structure as the previous one.

Output Format

Either the solution in the form "$(s_1, \ldots, s_m)$" with $g_i \in S$ for all $1 \leq i \leq m$ or the character "N" to denote the impossibility of finding a solution.

Examples

Suppose the following input:

```
1 2 3 4 5 6 7 8 9
3
(1,1,1) -> (0,0)
(2,2,2) -> (0,0)
(3,3,3) -> (0,0)
(4,4,4) -> (1,0)
(5,5,5) -> (0,0)
(6,6,6) -> (0,0)
(7,7,7) -> (1,0)
(8,8,8) -> (1,0)
(4,7,8) -> (1,2)
(4,8,7) -> (0,3)
(8,7,4) -> (0,3)
(7,8,4) -> (1,2)
(8,4,7) -> (1,2)
%
1 2 3 4 5 6 7 8 9
3
(1,1,1) -> (0,0)
(2,2,2) -> (0,0)
(3,3,3) -> (0,0)
(4,4,4) -> (1,0)
(5,5,5) -> (0,0)
(6,6,6) -> (0,0)
(7,7,7) -> (1,0)
(8,8,8) -> (1,0)
(9,9,9) -> (0,0)
%
```

Then the corresponding output is:

```
(7,4,8)
N
```

Problem F

Recording a tape

File Names

- **Source File:** tape.pas or tape.c or tape.cpp
- **Input File:** tape.in
- **Output File:** tape.out

Statement of the Problem

The market offers several types of cassette tapes, with standard time durations. You have some of these types at home, and you want to record all songs from a given list. The songs you hope to record have their time durations expressed, as usual, in minutes and seconds. Given the list of songs, find the list per side in which they have to be recorded in order to use the shortest possible available cassette for recording them.

Input Format

The first line in the file tape.in informs the cassette durations available; each of the following lines contain the duration of one song. The input file contains several instances, that follow this same specification, which finish with a line containing only a % (percentage symbol).

Output Format

The first line should contain the duration of the cassette. Next, "Side A" should be printed followed by lines with the song durations for this side. "Side B" follows in the same manner. The output file should have a % (percentage symbol) printed after the output of each of the several input instances.

Examples

Suppose the following input:

```
56 90 120
20m 44s
4m 36s
7m 18s
13m 8s
9m 6s
8m 12s
%
30 45
3m 11s
4m 45s
13m 45s
6m 8s
%
```

Then the corresponding output is

```
90
Side A
20m 44s
4m 36s
7m 8s
Side B
13m 8s
9m 6s
8m 12s
%
30
Side A
3m 11s
4m 45s
6m 8s
Side B
13m 45s
%
```

Observations

No pair of songs have the same duration.