Old Mutual - CSSA Computer Olympiad DAY 2

Overview

Problem	Paper	Atoms	Tree
Author	Shen Tian	Harry Wiggins	Carl Hultquist
Program name	paper.exe	atoms.exe	tree.exe
Source name	paper.pas paper.cpp paper.c paper.java	atoms.pas atoms.cpp atoms.c atoms.java	tree.pas tree.cpp tree.c tree.java
Input file	paper.in	atoms.in	tree.in
Output file	paper.out	atoms.out	tree.out
Time limit per test	1 second	1 second	2 seconds
Number of tests	10	10	10
Points per test	10	10	10
Total points	**100**	**100**	**100**

The maximum total score for Day2 is 300 points.

Paper

Author

Shen Tian

Introduction

It is time for the Guji people's annual harvesting festival. According to tradition, the village prepared a mural in praise of the Great Cow. At end of the festival, the mural is to be divided amongst the various farmer of the village as a token of good fortunes for the coming year. The parts for each farmer have been drawn out on the mural already. It is time to cut it. In order to do this honorably, the mural has to be cut into two pieces each time, with a straight cut from one side to the other.

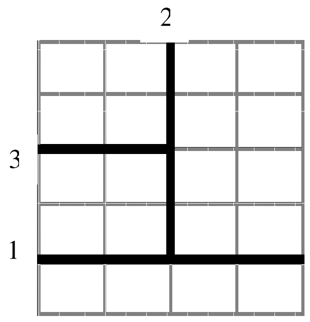
Task

The mural is a rectangle of integer side lengths. The parts marked out for each farmer are smaller rectangles, whose sides are parallel to the sides of the mural. In addition, the distances between the sides of the smaller rectangles are integer length away from the sides of the mural.

Each cut must be a straight cut, parallel to a side of the mural, and go all the way through the current piece (Once cut, the piece becomes two other pieces). In other words, you cannot cut half-way through a piece of the mural. Cuts can cross each other, but may not overlap, or be along the perimeter of the mural.

You are to decide the order and location of the cuts to be made in order to divide the mural into the separate pieces for each farmer.

Example



Consider this mural. To divide it, the line labeled 1 should be cut first. Then, since the mural is now in two pieces, cut 2 can be made. Cut 3 to divide the mural into the required 4 pieces will follow.

Sample Input

The first line of input will consist of two integers M and N, separated by a single space. The next M lines will each contain N characters (each character being one of a-z), representing the mural. Each piece of the mural will be represented using a different character. It is guaranteed that the mural will be entirely divided into rectangles and that no two distinct pieces will be represented by the same character.

Example:

```
5 4
aabb
aabb
ccbb
ccbb
dddd
```

Sample Output

The first line of the output will contain a single integer x, indicating the number of cuts needed. The next x lines represent the cuts in the order they were made. Each of these lines contains 4 space-separated integers, m_1, n_1, m_2, n_2. Each line represents a cut, from (m_1, n_1) to (m_2, n_2) and m_1<= m_2, n_1<=n_2. The top right left corner of the mural is (0, 0) and the bottom right corner is (M, N). If there are no valid ways of cutting the mural, print only a single line containing "-1".

Example:

```
3
4 0 4 4
0 2 4 2
2 0 2 2
```

Constraints

0 <= M, N <=100

Time limit

1 Second per test case.

Scoring

A correct solution earns 10 points. Any solution that makes illegal cuts or does not correctly divide the mural scores 0 points.

Atoms

Author

Harry Wiggins

Introduction

The Guji physicist discovered a molecule consisting out of N different atoms arranged in a single row. He numbered them 1, 2, 3, … N. He built an "atommeter" that can access any two distinct atoms simultaneously to work out the distance between them. The distance between any two consecutive atoms is the same.

The physicist would like to work out in which order the atoms are arranged, but accessing the atoms excites them. If the physicist accesses an atom more than 5 times the entire molecule gets destroyed.

Task

Your task is to discover the order of the atoms with the least number of accesses.
This interactive program uses stdin and stdout. The first integer that you should read is the number of atoms in the molecule. Then for each query of the atommeter you should produce (via stdout) two integers separated by a space. After each query you should accept an integer (via stdin)., representing the distance between those two atoms.
When you have worked out the order of the atoms you should print "-1 -1" (stdout) and on the next line you should print your answer (stdout) and then exit your program.
Your answer should be a single line of space-separated integers representing the atoms in the correct order.

Example

Output by your program by stdout	Returned to your program via stdin
	3
1 2	
	1
2 3	
	1
3 1	
	2
-1 -1	
3 2 1	

Constraints

3<=N<=100

Time limit

1 second

Scoring

Let G be the maximum number of times an atoms was accessed in your experiment. If your order is the correct order or in reverse, you'll get

100% if G = 3
70% if G = 4
30% if G = 5
0% if G>5

Using stdout and stdin

Pascal instructions

Do **NOT** use the CRT unit in your program. It can interfere with the flow of data between your program and the evaluator.

Perform the interaction as follows:

```
writeln(guess);
flush(output);
readln(reply);
```

C/C++ instructions

Perform the interaction as follows:

```
printf(”%d\n”, guess);
fflush(stdout);
scanf(”%s”, reply);
```

C++ instructions for C++ streams

Perform the interaction as follows.

```
cout << guess << ”\n”;
cout.flush();
cin >> reply;
```

Java instructions

Perform the interaction as follows, where `in` is a

```
BufferedReader in =
                 new BufferedReader(
                 new InputStreamReader(
                      System.in));
System.out.println(guess);
System.out.flush();
guess = in.readLine();
```

Hut Tree

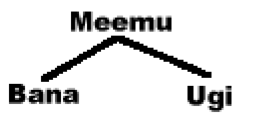
Author

Carl Hultquist

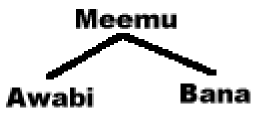
Introduction

The Guji tribe has a number of huts in their village, which they wish to connect up with footpaths. Each hut is named according to the family that lives inside that hut. Seeking a challenge, the tribe wants to connect up the huts in a special way so that the paths and huts form what is known as a *binary search tree*.

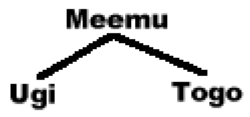
This means that there is a "head hut" which forms what is called the *root* of the tree. Each hut then has at most two paths leading from it to other huts: the path to the left must lead to the huts whose names are lexicographically (alphabetically) less, whilst the path to the right must lead to the huts whose names are lexicographically greater. For example, in the figures below, (a) is valid because "Bana" is lexicographically less than "Meemu" and so can be found by following the path to the left, whilst "Ugi" is lexicographically greater than "Meemu" and so can be found by following the path to the right. Figures (b) and (c) are invalid: in (b), "Bana" is lexicographically less than "Meemu" but is found by following the *right* path (and not the left one), and in (c), "Ugi" is lexicographically greater than "Meemu" but it found by following the *left* path (and not the right one).



(a)



(b)



(c)

To complicate matters, the tribe doesn’t want to do more work creating paths than is absolutely necessary. For this reason, the total length of path created must be minimised.

Task

Your job is to create the paths between the huts such that:

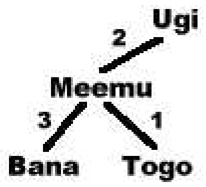
- The paths and huts form a binary search tree
- There is no other binary search tree whose total path length is shorter than your total path length.

Example

The tribe has 4 huts that house the families of Ugi, Togo, Bana and Meemu, and the distances between the huts is shown in the table below:

	Ugi	**Togo**	**Bana**	**Meemu**
Ugi	-	5	6	2
Togo	5	-	4	1
Bana	6	4	-	3
Meemu	2	1	3	-

Below is an example of a solution for this problem, which has a total path length of 6. In this example, there is only one binary search tree that has a total path length of 6. There may be cases with multiple solutions, but you will only need to find one of them.



Input Format

Input should be read from the file **tree.in**

The first line of input will consist of a single integer, N, specifying the number of huts in the village. The next N lines of input will then each consist of the name of each hut. The following N lines will each contain N integers that indicate the distance to the other huts in the village. Each hut will have a unique name, and each name will consist of a series of at most 20 lowercase letters. The distance from any hut to itself is 0, and will be specified as such in the input (but you may not build a path from a hut to itself). Also, if the distance from hut I to hut J is D, then you are guaranteed that the distance form hut J to hut I is also D.

Sample Input

```
4
ugi
togo
bana
meemu
0 5 6 2
5 0 4 1
6 4 0 3
2 1 3 0
```

Output Format

Output should be written to the file **tree.out**

The first line of output must contain an integer L and name R, separated by a space. L must be the total path length of your solution, and R is the name of the root hut in the binary search tree that you have found. The next N - 1 lines will describe the paths that you have created. Each line must contain two names, A and B, that represent the two huts between which you have created a path. The paths can be listed in any order, and the two huts forming the path can be in any order.

Sample Output

```
6 ugi
ugi meemu
meemu bana
meemu togo
```

Constraints

- 3 • N • 250
- 1 • d • 100000, where d represents the distance between any two huts

Time limit

2 seconds

Scoring

Any solution that does not produce output in the format specified above will score 0. Any solution whose paths do not form a binary search tree with the root specified will score 0. If the total path length of the paths specified does not match the total path length L, the solution will score 0.

Otherwise, if the optimal total path length is P and the total path length in the solution is L, the solution will score:

$S = 10 * [(P/L)^3]$