

Problem A

Backward numbers

Input file: backw.in

Output file: backw.out

Backward numbers are numbers written in ordinary Arabic numerals but the order of the digits is reversed. The first digit becomes the last, and vice versa. For example, the number 1245 becomes 5421. Note that all leading zeroes are omitted. This means that if the number ends with a zero, the zero is lost by the reversal; e.g., 1200 gives 21. Also note that the reversed numbers never have any trailing zeroes.

We need to calculate with backward numbers and your task is to write a program that can add two backward numbers and output their sum as a backward number. Of course, the result is not unique (e.g., 21 could be 12, 120, or 1200 before the reversal). Thus, we assume that no zeroes were lost in the reversal (e.g., we assume that the original number was 12).

Input specifications

The input consists of N cases. The first input line contains only the integer N , and we assume that $N > 0$. The follow the cases. Each case consists of exactly one line with two non-negative integers separated by a space. These are the reversed numbers you are to add.

We assume that all numbers are in the range $0 \leq n < 10^9$.

Output specifications

For each case, print exactly one line containing only one integer—the backward sum of the two backward numbers. Omit any leading zeroes in the output.

Sample input

4
24 1
101 0
4358 754
305 794

Output for sample input

34
101
1998
1

Problem B

Christmas presents

Input file: christmas.in

Output file: christmas.out

The local Santa Claus needs help! He has realized that he cannot afford to give every child a present any longer, so he needs your help to select those lucky children that will get a present. He has made a record for each child containing the child's level of excitement and frustration when getting or not getting a present, and now he wants a program that ensures that maximum total satisfaction is reached while preventing the total expenses to exceed his upper limit. The total satisfaction is defined as the sum of all excitements for those children that get presents, minus the sum of frustrations for the others.

Input specifications

First line: n — the number of children (integer less or equal 1000), and p (integer less or equal 1000) — the expense limit.

Next n lines: 3 integers: price, excitement level and frustration level for each child. You can assume that all the prices, and excitement and frustration levels are nonnegative integers.

Output specifications

First line: Total satisfaction. Second line: A string of n zeroes and ones — 0 means that this child does not get a present, 1 means that s/he gets a present.

Sample input

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

Output for sample input

11
11001

Problem C

Tournament ranking

Input file: rank.in

Output file: rank.out

You are given a number of teams and the results of games played by pairs of these teams in a tournament. The teams could be football teams, bowling teams, basketball teams, depending on your favorite sport. Each team will be identified by an upper case letter. The maximum number of teams is 26.

Your mission is to rank the teams if possible. For this you are given results from games played between the teams, i.e., you should produce a linearly ordered list starting with the best team, and progressing down to the worst one. We assume that if team A beats team B then team A is always better than team B and if team B in its turn beats team C then team A is also better than team C. Thus given the above results the teams would be ranked A B C. Note that it might not always be possible to produce a legal ranking: if team C again beats team A then it is not possible to rank the teams.

Your job is to produce a legal ranking or to determine that no such ranking exists. If you cannot determine the order of two teams then the lexicographically largest should be listed first, i.e. A should be listed before B, and the lexicographically largest ordering should be preferred, i.e. B C A D should be preferred before C A B D.

Input specifications

First there will be a line containing the number of tournaments you are asked to rank. For each tournament there will first be a line containing the number of teams and the number of games played. You may assume that the teams are named consecutively starting from A. For each game there will be a line containing two upper case letters, separated by one blank space. The two letters on a line indicate the winner and loser (in that order) of a game played by the two teams.

Output specifications

The output will list all teams on one line, with each team appearing only once in its correct position. There should be one blank space between

adjacent letters representing the teams. If no ranking is possible your program should print No legal ranking possible.

Sample input

```
2
4 4
A B
B D
C D
A C
3 3
A B
B C
C A
```

Output for sample input

```
A B C D
No legal ranking possible
```

Problem D

Directed mazes

Input file: maze.in

Output file: maze.out

Directed mazes are—as most mazes—traversed by moving from intersection to intersection until the goal intersection is reached. As each intersection is approached from a given direction, a sign near the entry to the intersection indicates in which directions the intersection can be exited. These directions are always left, forward, right, or any combination of these.

Figure 1 on the following page illustrates a directed maze. The intersections are identified as “(row,column)” pairs, with the upper left being (1,1). The “Entrance” intersection for Figure 1 is (3,1) and the “Goal” intersection is (3,3). You begin the maze by moving north from (3,1). As you walk from (3,1) to (2,1), the sign at (2,1) indicates that as you approach (2,1) from the south (traveling north) you may continue to go only forward. Continuing forward takes you toward (1,1). The sign at (1,1) as you approach from the south indicates that you may exit (1,1) only by making a right turn. This turns you to the east now walking from (1,1) toward (1,2). So far there have been no choices to be made. This is also the case as you continue to move from (1,2) to (2,2) to (2,3) to (1,3). Now, however, as you move west from (1,3) toward (1,2), you have the option of continuing straight on or turning left. Continuing straight on would take you on toward (1,1), while turning left would take you south to (2,2). The actual (unique) solution to this maze is the following sequence of intersections: (3,1), (2,1), (1,1), (1,2), (2,2), (2,3), (1,3), (1,2), (1,1), (2,1), (2,2), (1,2), (1,3), (2,3), (3,3).

If you arrive at an intersection having no sign for any direction (for instance, when traveling south to (3,1) in Figure 1), you have come to a dead end and may not proceed beyond that intersection.

You must write a program to solve valid directed mazes. Solving a maze means (if possible) finding a route through the maze that leaves the Entrance in the prescribed direction, and ends in the Goal. This route should not be longer than necessary, of course.

Input specifications

The input file will consist of one or more directed mazes. The first line of each maze description contains the name of the maze, which is an

column 1, and all other lines should start in column 3, i.e., indented two spaces. Solutions should be output as a list of intersections in the format “(R,C)” in the order they are visited from the entrance to the goal, should be delimited by a single space, and all but the last line of the solution should contain exactly 10 intersections.

The first maze in the following sample input is the maze in Figure 1.

Sample input

```
Sample
3 1 N 3 3
1 1 WL NR *
1 2 WLF NR ER *
1 3 NL ER *
2 1 SL WR NF *
2 2 SL WF ELF *
2 3 SFR EL *
0
NoSolution
3 1 N 3 2
1 1 WL NR *
1 2 NL ER *
2 1 SL WR NFR *
2 2 SR EL *
0
END
```

Output for sample input

```
Sample
(3,1) (2,1) (1,1) (1,2) (2,2) (2,3) (1,3) (1,2) (1,1) (2,1)
(2,2) (1,2) (1,3) (2,3) (3,3)
NoSolution
No solution possible
```

Problem E

Connecting islands

Input file: islands.in

Output file: islands.out

In order to win the last election the politicians promised the inhabitants of the Skofoten islands that they would connect all the islands by building bridges between them so that any island is reachable from any other island. After the election it occurred to them that this could get very costly, so in order to keep the cost down they have asked you to write a program that will determine the minimum cost needed to fulfill their promise. You can assume that the cost of a bridge is proportional to its length, thus we wish to calculate the minimum total length of the bridges needed to connect all the islands. Each island is represented by a polygon and to make things easier you can assume that bridges only run between corners of two polygons. Note that a bridge can only run over water, however, bridges can cross each other. Also note that the shape of islands can be non-convex.

Input specifications

The first line of input contains the number of test cases. A test case consists of one line holding the number of islands ($2 \leq N \leq 15$), followed by N lines that describe the islands. An island is a polygon, which is described as a number ($1 \leq P \leq 25$) that gives the number of points followed by P pairs of coordinates. Each coordinate is an integer in the range $[-1000 \dots 1000]$. The points are listed in order such that by connecting consecutive points, and the last point to the first, the perimeter of the island is given. It is guaranteed that islands do not touch or intersect.

Output specifications

For each test case output two lines reporting the minimal interconnect as follows:

The minimal interconnect consists of N bridges
with a total length of L .

where N is the number of bridges, and L is the total length, which should be printed as a floating point number with an accuracy of three digits.

Sample input

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

Output for sample input

The minimal interconnect consists of 2 bridges
with a total length of 2.000.

Problem F

A language for constants

Input file: yacl.in
Output file: yacl.out

An obscure computer science professor wants to become famous developing a new programming language YACL (“Yet Another Constant Language”). This language is very simple; it only has four constructs:

C+1 creates the constant 1.

C-1 creates the constant -1 .

INCR adds 1 to the constant being generated.

DBL multiplies the constant being generated by 2.

A program in this language is a sequence of these constructs, one on each line, executed sequentially. The professor wants to keep the programs in this language simple, small, and fast. To achieve his goal, he adds the following constraints:

- Every program must begin either with **C+1** or with **C-1**.
- A given constant C must be generated with the smallest number of instructions possible.
- If a constant C can be generated by different programs (all with equal number of instructions), then the fastest program should be used. For this purpose, suppose that the instruction **DBL** is executed in T nanoseconds and the instruction **INCR** in $2T$ nanoseconds.

You were hired by the professor to write several sample programs, so that he can use them at various conferences to demonstrate his powerful new language. The professor will give you a few constants, and your task is to write programs to generate these constants, obeying the constraints above.

Input specifications

The input file may contain several instances of the problem. Each instance of the problem is just one line containing the numeric constant to be generated. All numbers are non-zero integers between -32768 and 32767 . A line containing the integer zero terminates the input file.

Output specifications

For each instance of the problem, your program should print one line saying "Constant n ", where n is the constant for that instance, followed by the most efficient program to generate that constant, one instruction per line. Insert a blank line between each instance.

Sample input

```
3
-5
1
7
0
```

Output for sample input

```
Constant 3
C+1
DBL
INCR
```

```
Constant -5
C-1
DBL
DBL
INCR
DBL
INCR
```

```
Constant 1
C+1
```

```
Constant 7
C+1
DBL
INCR
DBL
INCR
```

Problem G

Bridge placements

Input file: bridge.in
Output file: bridge.out

A large company is planning a new office campus. The campus will consist of two buildings, A and B, and they anticipate that much people will have to cross over from one building to the other one during a working day. To save people from having to go all the way down to the ground floor and up again, they plan to make some bridges between the buildings.

You are asked to make a program that finds the optimum placement of the bridges. All bridges are to be horizontal, from floor x in A to the same floor x in B. The optimum placement is defined as the placement where the sum of traversed stairs is at a minimum, when the sum is taken over all possible starting floors in A and ending floors in B. (I.e., we suppose that the number of traverses from floor x in A to floor y in B is a constant (=1) for every x and y .) Note that once the bridges are built, some people still might prefer to walk down to the ground floor if it results in less stairs to traverse.

In some cases, more than one set of bridge placements can give optimum. In these cases, choose the placement with the lowest bridges; thus, bridges 1, 2, 3 is preferred to 1, 2, 4, and 1, 8, 9 is preferred to 2, 3, 4.

Input specifications

One or more lines, each with three positive integers: the heights of the buildings and the number of bridges.

The input is terminated by a line containing the number -1.

Output specifications

For each input line we want two lines. The first with an integer showing the minimum sum of traversed stairs, and the next line showing the bridge placements. Ground level is reported as 0; the highest floor is $x - 1$ when the height is x .

Sample input

```
10 6 2
15 20 8
-1
```

Output for sample input

```
200
2 5
1882
1 2 4 6 8 10 12 14
```

Problem H

Request for permission

Input file: voronoi.in

Output file: voronoi.out

The World's superpower is preparing for an air attack in another part of the globe. For that reason they need to send some airplanes over the continent. Among others, a little country TidyLand received a request for a permission to fly through their air space.

The borders of TidyLand have a shape of a convex polygon. The air space of TidyLand is divided into segments. Each segment is monitored and controlled by one air control station. The segments are designed in such a way that each point of the TidyLand's air space is controlled by the control station, which is the nearest to that point.

The request for a permission specifies the starting and ending coordinates for the flight (both of them are outside of TidyLand). The planes will fly along a straight horizontal line. TidyLand Air Space Central needs to know which segments will be crossed or touched by the planes of the foreign army.

Input specifications

First part of the input specifies the borders of TidyLand. First line contains a single integer N , the number of sides that form the TidyLand borders polygon. ($3 \leq N \leq 20$). The following N lines contain integer coordinates $X_B[i], Y_B[i]$ ($i = 1..N$) of the vertices of the polygon. ($1 \leq X_B[i], Y_B[i] \leq 100$). The vertices are enumerated systematically in a clockwise direction. The second part of the input specifies the locations of the air control stations. First, a single integer M , the number of control stations, is given at a separate line, ($1 \leq M \leq 20$). It is followed by M lines with integer coordinates $X_C[i], Y_C[i]$ ($i = 1..M$) of the air control stations: i -th line contains the coordinates of the i -th control station. You can assume that all control stations have the same altitude. Last part describes the line of flight. It consists of a single line with 4 integers X_1, Y_1, X_2, Y_2 (all are between 0 and 100), where $[X_1, Y_1]$ are starting coordinates, and $[X_2, Y_2]$ are ending coordinates. Both points lie outside of TidyLand.

Output specifications

The output will consist of two lines. The first line will contain the number of segments, which will be crossed by the planes. The second line will list the numbers of segments through which the planes will cross in the same order as they will be entered by the planes. If the plane will not enter the air space of TidyLand, the output should consist of a single line containing 0.

Sample input

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

Output for sample input

```
2
1 3
```

