2005 APICS Programming Contest

The 2005 Atlantic Canada programming Contest was held at Acadia University, Wolfville Nova Scotia Friday Oct 21 2005. There were 20 teams of three students from 10 universities in the four Atlantic provinces of Canada.

Each team of 3 students was given 6 problems to try to solve in 5 hours using one computer following the rules for the ACM programming contests.

a. Span-a-lot - Todd Wareham
b. Round About and Back Again - Patricia Evans
c. Convex Hull - Michael Mcallister, Arthur Sedgwick
d. Shortest Path (longest edge) - Arthur Sedgwick
e. Designating Contest Seats - Owen Kaser
f. Temperatures in No Man's Land - Porter Scobey

The participating teams came from the following universities:

```
Acadia University, Wolfville N.S.
Dalhousie University, Halifax N.S.
Memorial University of Newfoundland, St. John's.
Mount Allison University, Sackville N.B.
St Marys University, Halifax  N.S.
St Francis Xavier University, Antigonish N.S.
University of Prince Edward Island, Charlottetown.
University of New Brunswick, Fredericton Campus.
University of New Brunswick, St John Campus.
Universite de Moncton, Moncton N.B.
```

The top 4 teams advanced to the next level of competition in Rochester N.Y. At most 1 team from an institution advances.

Problem A: Span-A-Lot

Given a set $P = \{p_1, p_2, \ldots, p_n\}$, $n \geq 4$, of points in positive three-dimensional integer space $\mathcal{Z}^3$, *i.e.*, $p_i = \langle x, y, z\rangle$ such that $0 \leq x, y, z$ and x, y, and z are integers, a **Manhattan spanning path** for P consists of three lines in $\mathcal{Z}^3$ such that each of the three lines is parallel to one of the x- y-, and z-axes, each of these lines is parallel to a different axis, the endpoints defining each line are in P, the pair of endpoints defining each line are distinct, these three lines form a path, and each point in P is on one of these lines. Note that certain P do not have a Manhattan spanning path, *i.e.*, all points in P are on one line. Write a program which, given a set P of points, determines the size of the largest subset $P' \subseteq P$ that has a Manhattan spanning path. Your input file is structured such that the first line gives the number of datasets in the file, and each dataset d_i consists of $(n_i + 1)$ lines, where line 1 specifies the number n_i of points in P for d_i and each of the remaining n_i lines specifies the x-, y-, and z-co-ordinates of one of the points in P. You may assume that no point is repeated and that all input files are formatted correctly.

Our contest software this year requires all data sets to be in one file (`A.in` for this problem). You should put a copy of file `A.in` in the directory with your program. However your program should take input from `System.in/stdin/cin`, which will be connected to `A.in` when you submit.

Sample input (available as file "A.in"):

```
2
4
0 0 0
0 0 1
0 1 1
1 1 1
7
0 0 0
0 0 1
0 0 4
0 1 4
1 1 4
5 5 5
4 1 4
```

Sample output (available as file "A.out"):

```
Max size of Manhattan-span set  = 4
Max size of Manhattan-span set  = 6
```

Problem B: Round About and Back Again

Evenings at the university student centre haven't been the same since the formation dancers took over. Each night passing students are stunned to see yet another complex arrangement of people, with dancers moving from position to position in a way that confuses the casual observer. You, however, have been asked to try to make sense of it.

A dance configuration consists of n positions. All moves consist of dancers going between positions at synchronized times, and each position can be occupied by at most one dancer at one time. Each position has a set of moves (dependent on the position), which enable the dancer to move to another position at the end of the next interval.

For a given position i, a different position j is considered k-danceable from i if and only if there is a sequence of exactly k moves that moves a dancer from position i to position j and back to i.

Ultimately, you would like to find sequences of dance moves for the entire formation. As a preliminary step, you need to determine for each position i, which positions are k-danceable from i. Since there are different formations on each night, you will be given cases for a number of nights; the first line in the input will consist of the number of nights (formation test cases) that follow. Test cases are also followed by a blank line. Label your output for each case with which night it is for (numbering consecutively starting from 1).

Each test case consists of an initial line with two numbers that give how many positions the formation has (n) and the value of k. This is followed by n lines (one for each position) that indicates which positions you can move into from that position in a single move. Positions are numbered from 1, and moves can be repeated in a sequence. Your output for each case should consist of a single line for each position, listing which positions are k-danceable from that position, in ascending order. Print "none" if there are no k-danceable positions from there. Note the blank line after each test case output.

Sample Input: (14 lines)

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

3 5
2 3
1
2
```

Sample Output: (13 lines

```
Night 1:
3 4 5
3 5
1 2 4 5
1 3 5
1 2 3 4
none

Night 2:
2 3
1 3
1 2
```

Problem C Earthquake Recovery

A massive pyramid has been discovered in Central America. It consists of huge circular rock slabs placed on top of each other - each of smaller diameter than the one below it. Unfortunately the pyramid is in an earthquake zone and slabs have shifted so they are no longer concentric. An archaeologist has viewed the pyramid from far above and recorded x-y co-ordinates of points on the perimeter of each stone. She knew that 3 points were enough to determine the position of each slab but tended to record extra points just in case. She also observed that each slab had not shifted so much that it protruded over the edge of the slab below. Unfortunately there was a helicopter crash and the data was scrambled but recovered all out of order.

Ann Cherry has been given the set of points and asked to unscramble them. She is thorough and tests her software on several test cases. Each test case has 3 or more points from the perimeter of 1 or more nested circles but not ordered in any useful way. The good news is that the polygons formed by the points on each circle also nest properly thanks to the foresight of the archaeologist, i.e. the convex polygon formed by the points on the perimeter of one slab is strictly inside the polygon formed by the points on the perimeter of the slab below.

We need to know, for each test case, how many circles are present and how many points are on each. There will not be more than 300 points or 30 circles.

Input: The input file (from standard in) contains multiple test cases. The first number in the file indicates the number of test cases in the file. There is no special separation between test cases.
Each test case begins with a single integer that indicates how many points are in the test case. It is followed by that many points.

Let C(x,y,r) denote a circle of radius r with center x,y. The following sample input (available for testing as file C.in) has two test cases.
The first has 19 points, 4 on C(0,0,5),   6 on C(0,0,13), 9 on C(0,0,25).
The second has 18 points, 4 on C(12,9,20), 7 on C(5,15,50), 7 on C(0,0,101).
Note that the order of the last 18 points is scrambled.

```
2
19
 3  4   -4  3  -3  -4    4 -3
 5 12  -12  5  -5 -12   12 -5    0 -13   13   0
24  7   15 20  -7  24  -15 20  -24  -7  -20 -15  -15 -20  7 -24  20 -15
18
-101  0 -99  20  -35  45 -25 -25 -20 -99 -9 63 -8  9  0 101  12 -11
  12 29  19 -33   20 -99  20  99  35  55 32  9 53 29 55  15 101   0
```

Output: Output a single line for each test case. The line for each test case lists the number of points in each of the circles, given from the innermost circle to the outermost circle. The output for the sample input data is

```
4 6 9
4 7 7
```

Note 1: Here are some Pythagorean Triples that may prove useful for your testing:

```
 3  4   5   5 12 13   8 15 17   7 24 25  15 20 25  20 21 29  14 48  50
16 63  65  25 60 65  39 52 65  33 56 65  48 55 73  39 80 90  20 99 101
```

Note 2: Consider a path from (x1,y1) to (x2,y2) and then to (x3,y3).
The expression (x2-x1)*(y3-y1)-(x3-x1)*(y2-y1) is
 > 0 if a LEFT turn occurs at (x2,y2),
 < 0 if a RIGHT turn occurs at (x2,y2),
 = 0 if the 3 points are COLINEAR.

Problem D Rescue in New Orleans

The city is now 'dry' but plans are being made to improve the response the next time the city, built below sea-level, floods. Most buildings will be required to have a small rescue platform attached which is above sea-level. In the diagram these platforms are shown as short horizontal or vertical line segments labelled P if person(s) are waiting to be rescued there, or E if empty, or S if this is a place of safety which people should get to. If another flood occurs the plan is for people to wait at a platform until a floating flexible dock walkway (FFW) is towed by a boat and connected to the platform and another one nearby. People will walk from platform to platform until they reach an S platform which is connected to the outside world by above sea-level routes. Each platform is a short straight walkway which must be oriented east-west or north-south only. A FFW can connect to a platform anywhere along its length. These FFWs should be as short as possible to save on storage and other costs and make it easier to tow them around. You are to write a program to calculate the length of the longest of the collection of FFWs which will be sufficient to rescue everyone. This longest FFW should be as short as possible. If necessary, it alone could be used to rescue everyone.

```
   |
  7+   +++++++++   +++++   +++++
   |       E         E       E +
   |                           +
  6+   +                       +
   |  S+                      P+
   |   +                       +
  5+   +                       +
   |
   |         E       P       E
  4+       +++++   +++++   +++++
   |
   |
  3+   +
   |  S+
   |   +
  2+   +
   |
   |         E       E       ?
  1+       +++++   +++++   +++++
   |
   |
  0+---+---+---+---+---+---+---+---
       1   2   3   4   5   6   7
```

Input: Each platform is represented by three integer values X, Y, L. The orientation of the platform is east-west if L>=0, otherwise north-south. |L| is the length in either case. X and Y are the east-west and north-south coordinates of the west end of a platform if L>=0 or south end if L<0.

The first line of input contains the number of test cases to follow.
For each test case there will be:
- a line containing X Y L for each S platform,
- a line containing X Y L for each P platform with people waiting,
- zero or more lines containing X Y L for each E platform with no one waiting,
- then an empty line.
The total number of platforms in any test case will not exceed 250.

File D.in (which will be redirected to standard input) contains 9 lines:

```
2
1 2 -1  1 5 -1
4 4  1  7 5 -2
6 1 1   2 4  1  6 4  1  1 7 2  4 7 1  6 7 1  2 1 1  4 1 1

1 2 -1  1 5 -1
4 4  1  7 5 -2  6 1 1
        2 4  1  6 4  1  1 7 2  4 7 1  6 7 1  2 1 1  4 1 1
```

There are two test cases here. The second is almost the same as the first except that 6 1 1 (labelled ? in the diagram) has moved to the second line from the third meaning there are people to be rescued there in the second test case.

Output
The output should contain one double value for each data set rounded to the second digit after the decimal point. This is the length of the longest FFW necessary to rescue everyone (which should be as small as possible).
For the input shown above the output should be the 2 lines:

```
1.00
1.41
```

In the first test case people can be rescued by taking a counter-clockwise route to the upper S platform. The 1.41 is the rounded value of sqrt(2.0) which is necessary to rescue the people at the platform labelled ?.

Problem E: Designating Contest Seats

Linear University is hosting the APICS programming competition and you are the person organizing the contest. Their computer lab (Long Hall) contains an extremely long table, with computers placed at regular intervals along the table. The table is placed against a wall and people can sit on only one side of the table. There are up to 256 computers on the table, and they are numbered uniquely from 0 to 255. (A computer's number is the last 8 bits of its IP address.) A computer's number cannot be changed, and computers cannot be moved...EXCEPT that every night, elves sneak into Long Hall and they move the computers, rearranging their order on the table.

On the day of the competition, teams will be let into Long Hall and allowed to use designated computers. You need a program to choose the designated computers. Certain criteria must be met:

1. First, you must never designate two computers that are adjacent on the table. The programming teams would be too close.

2. Second, the designated computers' numbers must fall within a single range. (The network firewall for Linear University can thus be set, with a single rule, to block Internet access to all machines in the range.) A range can begin at any address between 0 and 255, and it can also end at any address between 0 and 255. For instance, the range 25-50 contains up to 26 computers, those (if any) with addresses 25, 26, 27, ..., 49, 50. (Since there may be fewer than 256 computers, there may be certain numbers that are not the address of any computer.)

3. Third, the range containing the designated computers MUST NOT contain any non-designated computers. (Long Hall is the ONLY computer lab in Linear University, so all computers not being designated for the contest must be available for the Arts students at Linear University, who will complain bitterly if the firewall blocks their Internet access.)

Thus, the required program will run early in the morning of the competition, after the elves have finished their nightly mischief. It will input the machine addresses, in the order in which the machines occur along the table.

It will output the maximum number of computers that can be designated for the competition. It will also output a range within which the designated computers' numbers lie.

INPUT FORMAT

Our contest software this year requires all test cases to be in one file (E.in for this problem). You should put a copy of E.in in the directory with your program. However your program should be written to take input from System.in/stdin/cin. This will be connected to E.in when you submit.

In E.in, the first line contains a decimal number (without any additional white space) between 1 and 99, and it specifies the number of test cases that follow.

Each test case is a sequence of lines, and each line contains a single number (with no white space). The jth line in a test case contains the machine address of the jth computer on the table.
A single blank line (which might contain white space) follows every test case, including the last one.
You may assume that there is at least one computer on the table, in each test case.

OUTPUT FORMAT

The output consists of a group of four lines for every input test case.
* The first line contains the maximum number of computers that can be designated for the competition.
* The second line contains the smaller endpoint of the range.
* The third line contains the larger endpoint.
* The fourth line is blank.

EXAMPLE (with two test cases)

If E.in contains
```
2
10
100

1
5
2
6
10
11
8
```

one possible output would be
```
1
10
12

3
4
8
```

because, in the case of the first test case, address range [10,12] contains 1 machine. Since the table contains only 2 machines, they are adjacent and hence we can designate only one machine for the contest. Any other range containing only one machine would be okay.

For the second test case, the address range [4,8] contains 3 machines, those with addresses 5, 6, and 8. None of these machines are adjacent on the table,

Note that other address ranges (eg, [3,8], [3,9], [5,8]) would also contain the same 3 machines and lead to different possible answers. However, there is no range containing 4 computers that meets the requirements.

Problem F: Temperatures in No Man's Land

Jake has lived out in No Man's Land for years, and still continues to send us valuable data, even though it's not in our preferred form, so we don't rock the boat. Every day at noon Jake reads the temperature on his back porch, then writes down the date of the month and the day's temperature on a slip of paper, which he tosses into a nearby shoebox. At the end of each month he goes to the shoebox, takes out the pieces of paper, one at a time, and copies each pair of numbers on to a single sheet of paper, in the order he draws them out of the shoebox, one pair per line.

Fortunately, Jake also writes the month and year at the top of each of these sheets, and eventually he sends them to us. Our data entry expert, Sylvia, enters the number pairs into a file, one pair per line, and gives the file an appropriate name reflecting the month and year of record. Sometimes she enters the data for two or more months into a single file, and when she does she indicates the end of each month of data by entering a line with 6 dashes on it. She also enters on the first line of the file the number of months of data in that particular file. Trouble is, at minimum wage we can't get either Jake or Sylvia to do any sorting, so the order of these entries in each file is the same as the order they came out of Jake's shoebox.

One of the things we have learned, however, is that both Jake and Sylvia are very reliable. That is, we always get a full month's data on one of Jake's sheets, and Sylvia just never makes a mistake when performing data entry.

The weather in No Man's Land seems to be stabilizing of late, so our latest interest is in determining the longest sequence of days in each month for which the temperature remained the same at Jake's noontime reading. So far, for each month there has been one sequence of days on which Jake observed a stable noontime temperature that was longer than any other such sequence of days during that month, and we have no reason to believe that this weather pattern will change in the near future.

Your job is to write a program that will read a file like the one we've described, and, since we're paying you several times minimum wage, output one or more lines summarizing the results for each month's data found in the file in exactly the form shown below.

```
Month 1 : 31 days : 23 degrees for 10 days from day 15 to day 24
```