CSC Programming Contest
30 September 1995

Some rules and advice

1. All questions require you to read the test data from standard input and write results to standard output.
   - Output must correspond *exactly* to the provided sample output, including spelling and spacing.
   - All lines should be terminated with a new-line, and no whitespace should appear at the end of a line unless explicitly specified.
   - Tabs should never be used.
2. All programs will be re-compiled prior to testing with the judges' data. Non-standard libraries may not be used in your solutions. You are also not allowed to include any files in your program. The following files will be included standard in all C programs: `stdio.h`, `math.h`, `ctype.h`, `strings.h`, and `string.h`.
3. Programming style is not considered in this contest. You are free to code in whatever style you prefer.
4. The CPU time limit for all questions is the same and will be announced before the contest.
5. All questions regarding the contest material should be submitted to the judges through the `submit` command. You can ask the helpers in the room for any non-contest related matters or for getting your printer output. The helpers will not answer any questions regarding the contest material.
6. Judges' decisions are to be considered final. No cheating will be tolerated.

Success!

Problem A
Divisors

Mathematicians love all sorts of odd properties of numbers. For instance, they consider 945 to be an interesting number, since it is the first odd number for which the sum of its divisors is larger than the number itself.

To help them search for interesting numbers, you are to write a program that scans a range of numbers and determines the number that has the largest number of divisors in the range. Unfortunately, the size of the numbers, and the size of the range is such that a too simple-minded approach may take too much time to run. So make sure that your algorithm is clever enough to cope with the largest possible range in just a few seconds.

Input Specification

The first line of input specifies the number N of ranges, and each of the N following lines contains a range, consisting of a lower bound L and an upper bound U, where L and U are included in the range. L and U are chosen such that $1 \leq L \leq U \leq 1\,000\,000\,000$ and $0 \leq U - L \leq 10\,000$.

Output Specification

For each range, find the number P which has the largest number of divisors (if several numbers tie for first place, select the lowest), and the number of positive divisors D of P (where P is included as a divisor). Print the text '`Between` L `and` H`,` P `has a maximum of` D `divisors.`', where L, H, P, and D are the numbers as defined above.

Example input

```
3
1 10
1000 1000
999999900 1000000000
```

Example output

```
Between 1 and 10, 6 has a maximum of 4 divisors.
Between 1000 and 1000, 1000 has a maximum of 16 divisors.
Between 999999900 and 1000000000, 999999924 has a maximum of 192 divisors.
```

Problem B
Fatman

Some of us may be so fortunate to be thin enough to squeeze through the tiniest hole, others are not. Getting from A to B in a crowded supermarket (even without a cart) can be tough and may require sophisticated navigation: there may seem to be enough room on the one side, but then you may run into trouble with that lady further down...

Let's consider this in an abstract fashion: given an aisle of a certain width, with infinitely small obstacles scattered around, just how fat can a person be and still be able to get from the left side to the right side. Assume that seen from above a (fat) person looks like a circle and the person is incompressible (a person with diameter d cannot go between two obstacles having distance less than d).

Input Specification

The first line of input specifies the number of test cases your program has to process. The input for each test case consists of the following lines:

- One line with the integer length L ($0 \leq L \leq 100$) and integer width W ($0 \leq W \leq 100$) of the aisle, separated by a single space.
- One line with the number of obstacles N ($0 \leq N \leq 100$) in the aisle.
- N lines, one for each obstacle, with its integer coordinates X and Y ($0 \leq X \leq L$, $0 \leq Y \leq W$) separated by a single space.

Output Specification

For each test case given in the input, print a line saying '`Maximum size in test case` *N* `is` *M*`.`', where *M* is rounded to the nearest fractional part of exactly four digits. *M* is the maximum diameter of a person that can get through the aisle specified for that test case. *N* is the current test case number, starting at one.

Example Input

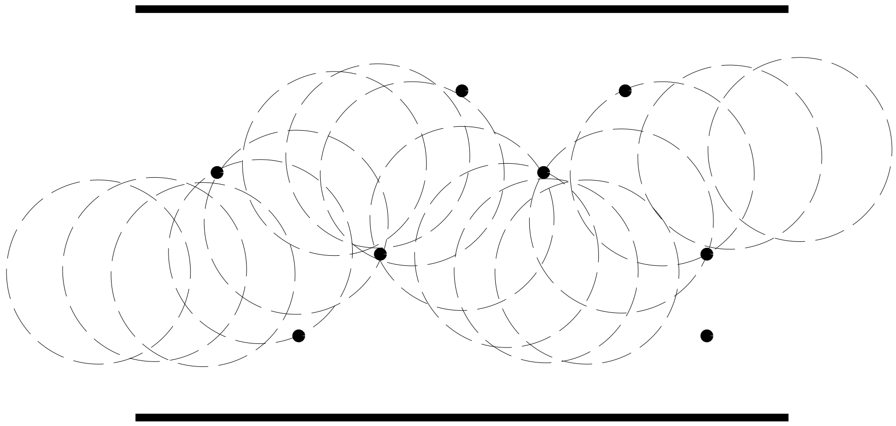
```
1
8 5
8
2 1
1 3
3 2
4 4
5 3
6 4
7 2
7 1
```

Example Output

```
Maximum size in test case 1 is 2.2361.
```

Additional Data

The Example Input looks like:



Problem C
Safebreaker

We are observing someone playing the game of Mastermind. The object of this game is to find a secret code by intelligent guess work, assisted by some clues. In this case the secret code is a 4-digit number in the inclusive range from 0000 to 9999, say "3321". The player makes a first random guess, say "1223" and then, as for each of the future guesses, gets a clue telling him how good his guess is. A clue consists of two numbers: the number of correct digits (in this case 1: the "2" at the third position) and the additional number of digits guessed correctly but in the wrong place (in this case 2: the "1" and the "3"). The clue would in this case be: "1/2".

Write a program that given a set of guesses and corresponding clues, tries to find the secret code.

Input Specification

The first line of input specifies the number of test cases (N) your program has to process. Each test case consists of a first line containing the number of guesses G ($0 \leq G \leq 10$), and G subsequent lines consisting of exactly 8 characters: a code of four digits, a blank, a digit indicating the number of correct digits, a '/' and a digit indicating the number of correct but misplaced digits.

Output Specification

For each test case, the output contains a single line saying either:

`impossible` if there is no code consistent with all guesses.

the secret code if there is exactly one code consistent with all guesses.

`indeterminate` if there is more than one code which is consistent with all guesses.

Example Input

```
4
6
9793 0/1
2384 0/2
6264 0/1
3383 1/0
2795 0/0
0218 1/0
1
1234 4/0
1
1234 2/2
2
6428 3/0
1357 3/0
```

Example Output

```
3411
1234
indeterminate
impossible
```

Problem D
Race Tracks

Many boring math classes have been spent playing Race Tracks, where two players have to maneuver their cars on a race track drawn on a piece of paper, while their cars can only accelerate by a limited (positive or negative) amount per move.

A variant of Race Tracks involves Hoppers. Hoppers are people on a jump stick who can jump from one square to the other, without touching the squares in between (a bit like a knight in chess). Just like the aforementioned cars, they can pick up speed and make bigger hops, but their acceleration per move is limited, and they also have a maximum speed.

Let's be a bit more formal: our variant of Race Tracks is played on a rectangular grid, where each square on the grid is either empty or occupied. While hoppers can fly over any square, they can only land on empty squares. At any point in time, a hopper has a velocity (x,y), where x and y are the speed (in squares) parallel to the grid. Thus, a speed of (2,1) corresponds to a knight jump, (as does (-2,1) and 6 other speeds).

To determine the hops a hopper can make, we need to know how much speed he can pick up or lose: either -1, 0, or 1 square in both directions. Thus, while having speed (2,1), the hopper can change to speeds (1,0), (1,1), (1,2), (2,0), (2,1), (2,2), (3,0), (3,1) and (3,2). It is impossible for the hopper to obtain a velocity of 4 in either direction, so the x and y component will stay between -3 and 3 inclusive.

The goal of Hopping Race Tracks is to get from start to finish as quickly as possible (i.e. in the least number of hops), without landing on occupied squares. You are to write a program which, given a rectangular grid, a start point S, and a finish point F, determines the least number of hops in which you can get from S to F. The hopper starts with initial speed (0,0) and he does not care about his speed when he arrives at F.

Input specification

The first line contains the number of test cases (N) your program has to process. Each test case consists of a first line containing the width X ($1 \leq X \leq 30$) and height Y ($1 \leq Y \leq 30$) of the grid. Next is a line containing four integers separated by blanks, of which the first two indicate the start point (x1,y1) and the last two indicate the end point (x2,y2) ($0 \leq x1, x2 < X$, $0 \leq y1, y2 < Y$). The third line of each test case contains an integer P indicating the number of obstacles in the grid. Finally, the test case consists of P lines, each specifying an obstacle. Each obstacle consists of four integers: x1, x2, y1 and y2, ($0 \leq x1 \leq x2 < X$, $0 \leq y1 \leq y2 < Y$), meaning that all squares (x,y) with $x1 \leq x \leq x2$ and $y1 \leq y \leq y2$ are occupied. The start point will never be occupied.

Output specification

The string '`No solution.`' if there is no way the hopper can reach the finish point from the start point without hopping on an occupied square. Otherwise, the text '`Optimal solution takes` N `hops.`', where N is the number of hops needed to get from start to finish point.

Example input

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

Example output

```
Optimal solution takes 7 hops.
No solution.
```

Problem E
Train Swapping

At an old railway station, you may still encounter one of the last remaining "train swappers". A train swapper is an employee of the railroad, whose sole job it is to rearrange the carriages of trains.

Once the carriages are arranged in the optimal order, all the train driver has to do, is drop the carriages off, one by one, at the stations for which the load is meant.

The title "train swapper" stems from the first person who performed this task, at a station close to a railway bridge. Instead of opening up vertically, the bridge rotated around a pillar in the center of the river. After rotating the bridge 90 degrees, boats could pass left or right. The first train swapper had discovered that the bridge could be operated with at most two carriages on it. By rotating the bridge 180 degrees, the carriages switched place, allowing him to rearrange the carriages (as a side effect, the carriages then faced the opposite direction, but train carriages can move either way, so who cares).

Now that almost all train swappers have died out, the railway company would like to automate their operation. Part of the program to be developed, is a routine which decides for a given train the least number of swaps of two adjacent carriages necessary to order the train. Your assignment is to create that routine.

Input Specification

The input contains on the first line the number of test cases (N). Each test case consists of two input lines. The first line of a test case contains an integer L, determining the length of the train ($0 \leq L \leq 50$). The second line of a test case contains a permutation of the numbers 1 through L, indicating the current order of the carriages. The carriages should be ordered such that carriage 1 comes first, then 2, etc. with carriage L coming last.

Output Specification

For each test case output the sentence: '`Optimal train swapping takes` S `swaps.`' where S is an integer.

Example Input

```
3
3
1 3 2
4
4 3 2 1
2
2 1
```

Example Output

```
Optimal train swapping takes 1 swaps.
Optimal train swapping takes 6 swaps.
Optimal train swapping takes 1 swaps.
```