27th ACM International Collegiate
Programming Contest, 2002-2003
Asia Region, Tehran Site
Sharif University of Technology, 15 Nov. 2002

Sponsored by IBM®

Problem A
(Program filename: A.CPP, A.DPR, or A.PAS)

Grandpa's Other Estate

From our previous contest, we know that Kamran the Believer inherited many of his grandpa's belongings. Apparently, his grandpa had been a mathematician in his life with interests in puzzle solving, since he has made Kamran solve another programming problem!

Grandpa had a big garden with many valuable walnut trees. He has written in his will that Kamran can inherit one piece of square shaped land of a given size in the garden, such that its sides be parallel to the x and y axes. Taking advantage of the fact that no other restrictions have been mentioned in the will, Kamran wants to choose the land in which the most number of trees lie. Kamran is too wealthy now and thus too lazy to spend time and solve another algorithmic problem. He has hired you to solve this problem for him.

You are given the location of all trees in the big garden and the size of the land to choose. You are to write a program to find out where to choose the land so that the most number of trees lie in it. You may consider trees as points in the plane and the land as a square. You are to find the position of the square such that it includes as many points as possible. Note that the points on the border of the square are considered to be inside it.

Input (filename: A.IN)

The first line of the input file contains a single integer t ($1 \le t \le 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains an integer n ($1 \le n \le 100$), the number of trees, and an integer r ($1 \le r \le 1000$), the length of the land's side, followed by n lines, each containing two integers x and y ($0 \le x, y \le 100{,}000$) representing the coordinates of a walnut tree. Note that all coordinates are pairwise distinct.

Output (filename: A.OUT)

There should be one line per test case containing the maximum number of trees that Kamran can own.

Sample Input

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

Sample Output

```
2
```

Problem B
(Program filename: B.CPP, B.DPR, or B.PAS)

Afshung Pizza Delivery

Afshung-Pizza chain, a door-to-door pizza delivery service for Hamedung, a Sildavya district, needs your help for fastest possible pizza delivery plan. With a GIS device that shows all streets of the Hamedung, each delivery boy can find a fast route to deliver the pizza to the place of order. The Elyasung Company that sells and supports this GIS device needs your help to reprogram the device to provide even a faster route plan.

Hamedung is a rectangular shape district with many two-way streets that are all rectilinear. To show the map, the GIS device uses text characters as shown in the sample test data. In this format, each one kilometer of a street is shown by a dash (`-`) or a pipe (`|`) showing that the street is either in west-east or in north-south direction. A plus (+) on the map indicates a sharp 90 degree turn (with length zero) on that position without any traffic light. All such turns are marked with `+`. An intersection is shown by an integer τ on that position. τ is the timing of the traffic light at that intersection which can be either three or four way. To have a smooth and accident-free district, the municipality of Hamedung has forced that every traffic light has only one green light and two or three red lights for three or four intersections respectively. One of the lights in that intersection remains green for τ minutes (i.e., during $[x, x + \tau)$ time for some x) and others are red. In the next τ minutes, one other direction turns green as if the green light turns counter clockwise. This rule is observed in all intersections.

The time is set to zero at the beginning when the delivery boy starts moving. At this time, all traffic signals are set such that only the southern light (or the northern light if no southern light exists) of each intersection is set to green, and other lights at this intersection are set to red.

Note that the only positions that the driver can change his direction are: a turn (+), or an intersection (represented by a digit). As an example, if we have a pattern like `-|-` in the map, the driver cannot cross the pipe if he is moving from left to right or right to left, neither can he turn left or right, since there is no intersection at this location.

Given the complete map described above, the location of an Afshung-Pizza branch, marked by `S`, and the location of the final delivery place, marked by `D`, you are to write a program for GIS device to automatically find the fastest possible route to deliver pizza from `S` to `D`. Note the following assumptions:

- `S` and `D` are parts of a street (replacing either a `–` or `|`)
- `S` and `D` are not adjacent to any intersection or turn.
- `S` and `D` are not adjacent to each other.
- Intersections and/or turns are at least one kilometer apart.
- `'S'`, `'D'`, `'+'`, and each digit have zero kilometer length.
- Speed of delivery boy is one kilometer per minute.
- Traffic that faces a green light can move in all directions. No straight move or turns are allowed at red light.
- There is no traffic jam or other obstacles on the way.
- Asterisk characters (`*`) show the border of the district.

Input (filename: B.IN)

The first line of the input file contains a single integer t ($1 \le t \le 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains two integer numbers: N ($1 \le N \le 100$) the number of rows of the map, and M ($2 \le M \le 100$) the number of columns of the map. Followed by the first line, there will be N lines, each containing a string of length M, consisting of `'-'`, `'|'`, `'+'`, `' '`, `'*'`, `'S'`, `'D'` or digits from `'1'` to `'9'`. Also, assume that the total number of intersections and turns (+) is at most 100.

Output (filename: B.OUT)

There should be one line in the output per test case containing a single number, the minimum time to drive from `S` to `D`, if there exists, otherwise the word `impossible` (with lower-case letters).

Sample Input

```
1
10 10
**********
*-D-3---+*
*   |   |*
*   | +-+*
*   |-|  *
*---4-1- *
*   | |  *
*   | |  *
*S--2-+  *
**********
```

Sample Output

```
15
```

Problem C
(Program filename: C.CPP, C.DPR, or C.PAS)

Housing Complexes

The Ministry of housing is planning a huge construction project of several housing complexes. Each complex includes several apartments to be sold to government employees at reasonable prices. The ministry has located several big $m \times n$ pieces of land that can potentially be used for such construction project; one complex in each land. The lands are all rectangular, each with $m \times n$ number of 1×1 square blocks. All housing complexes are $h \times w$ rectangles covering exactly $h \times w$ blocks of each containing land.

The problem is that there are originally some old buildings, each covering exactly one block of a land, making it impossible to locate enough free space for all the complexes in order to start the project. Therefore, the ministry has to buy some of these buildings, and demolish them to free the needed space. The old buildings belong to certain number of people. These people are angry of the possibility that their building may be bought and demolished, especially because the government usually pays much less for their buildings compared to the open market prices.

In response to the protests, the ministry announces a "fair" decision that if it buys some buildings in one land, it will only choose those that belong only to one owner, and will buy all of them at reasonable price. And, it promises not to buy buildings belonging to the same owner in other lands. Note that with this constraint, there may be some lands in which building a complex is impossible. Trying to keep its promises, the ministry has asked you to write a program to see how many housing complexes can be constructed at most with these conditions.

Input (filename: C.IN)

The first line of the input file contains a single integer t ($1 \leq t \leq 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains five integers k ($1 \leq k \leq 30$), the number of lands, m and n ($1 \leq m, n \leq 50$), the number of rows and columns in each land respectively, and h and w ($1 \leq h, w \leq 50$), the number of rows and columns a complex occupies. After the first line, there are $k \times m$ lines in the input, representing k lands, each by an $m \times n$ matrix. Each line contains a string of length n with no leading or trailing spaces. Each character in the strings represents a block in the land and may be an upper case alphabetic character 'A'..'Z', indicating the owner of the block, or the character '0' indicating the block is free.

Output (filename: C.OUT)

There should be one line per test case containing the maximum number of housing complexes that can be constructed for that test case.

Sample Input

```
2
3 4 3 3 2
A0B
000
0A0
00B
AA0
00B
0B0
000
A0A
000
B00
B00
3 4 3 3 2
A0B
000
0A0
00B
AA0
00B
0B0
000
A0A
000
0B0
B00
```

Sample Output

```
3
2
```

Problem D
(Program filename: D.CPP, D.DPR, or D.PAS)

Spacecraft Malfunction

It is said that court intrigues started with people lying about other people, and then lying about other people's lying, and so it went. The intriguers constantly looked for scapegoat who inevitably proved to be someone with the least power, though not always the least morality.

We have faced a similar problem, but this time, in our malfunctioning spacecraft! There are a number of units in the spacecraft. The units are so reliable that it would surprise us very much if more than one unit were faulty. If more than one is faulty, we would lose the probe, so we are sure that exactly one unit is faulty in our spacecraft.

We know that each unit checks exactly two others, and each unit will be checked by at least one other unit. A good unit will give accurate diagnosis of the units it checks. For example, if unit X is good and it says that Y is faulty and Z is good, then, in fact, Y is faulty and Z is good. However, a bad unit is unreliable. So, if unit X is faulty and makes the same statements, then Y may or may not be good, and Z may or may not be good either. Note that a unit cannot check itself.

Now suppose that you have the reports from all units and your duty is to find which unit is indeed faulty.

Input (filename: D.IN)

The first line of the input file contains a single integer t ($1 \le t \le 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains an integer n ($3 \le n \le 100$), the number of units, followed by n lines each describing a unit and the result of its checks. The line starts with a positive integer number which shows the identification number of the unit. After the id number, there are two pairs of checked unit id's and check results. A check result is a single character which is either Y or N, showing whether the result of checking is good or faulty respectively. As an example, the fourth line in the Sample Input section shows that *unit 16 has checked unit 8 saying it is good, and has checked unit 32 saying it is faulty*.

Output (filename: D.OUT)

There should be one line per test case containing either the id number of the faulty unit, or the word `impossible` (with lower-case letters), if it is impossible to find the faulty unit from the input data.

Sample Input

```
1
5
2 16 Y 32 N
16 8 Y 32 N
32 8 N 4 Y
8 4 Y 2 Y
4 2 Y 16 Y
```

Sample Output

```
32
```

Problem E

(Program filename: E.CPP, E.DPR, or E.PAS)

Blue x Red = Bang

The RB Company is one of the pioneering companies in making electronic boards. This company has recently faced a difficult problem to solve in designing its special power boards. Each power board is a flat plastic plate with special red and/or blue colored plugs on it. The blue plugs are recognized as null poles, and the red ones are phase poles. This company's special design requires that all the blue plugs should be inter-connected with straight lines to make a simple blue polygon. All vertices of the resulting polygon should be blue plugs, and any blue plug should be a vertex of this polygon. With similar conditions, all the red plugs should make a red polygon. You may assume that no three plugs of the same color are co-linear, i.e. lie on one line.

The design problem is that safety precautions require that there should be no red and blue polygon intersections; otherwise a disastrous explosion would be inevitable. This happens when the two polygons have non-empty intersection. The RB engineers have realized that some configurations of red and blue plugs makes it impossible to have non-intersecting red and blue polygons. They consider such configurations disastrous. Your task is to write a program to help the RB engineers recognize and reject the disastrous configurations.

Input (filename: E.IN)

The first line of the input file contains a single integer t ($1 \le t \le 5$), the number of test cases, followed by the input data for each test case. The first line of each test case contains two integers b and r ($3 \le b$, $r < 10$), the number of blue and red plugs respectively, followed by b lines, each containing two integers x and y representing the coordinates of a blue plug followed by r lines, each containing two integers x and y representing the coordinates of a red plug. Note that all coordinates are pairwise distinct and are in range 0 to 100,000 inclusive.

Output (filename: E.OUT)

There should be one line per test case containing `YES` if there exist non-intersecting polygons or `NO` otherwise. The output is considered to be case-sensitive.

Sample Input

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

Sample Output

```
YES
NO
```

Problem F

(Program filename: F.CPP, F.DPR, or F.PAS)

JaWs

"*Sigh*! Where are those good old bloody days?" Pondered Bob, the old shark, the former slayer of the deep blue waters, with his tears joining the infinite water of the ocean. Butchering for years, Bob's teeth has lost their regular shape, and the poor old shark is now in trouble closing his jaws. He wants to program his PDA to help him find the shape of his teeth when his jaws are closed and we want to help him write this program!

We name the sequence of Bob's lower teeth as LT and the sequence of his upper teeth as UT. For the sake of simplicity, consider LT as a sequence of adjacent equilateral triangles (i.e., with equal sides). All bases of the triangles lie on the same horizontal straight line. UT has a similar structure, except that the triangles are upside-down (Figure 1).

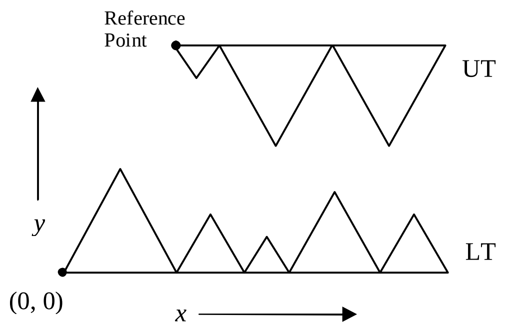

Figure 1. Snapshot of Bob's teeth!

Assume the left endpoint of the base of the leftmost tooth in LT has the coordinates (0, 0), so the bases of all triangles in LT lie on the *x* axis. We name the left end point of the base of the leftmost tooth in UT, the *reference point*. Initially, the coordinates of the reference point is given such that:

- The tip of no two teeth in LT and UT have the same *x* coordinates,
- UT is above LT, i.e. the *y* coordinate of the reference point is greater than zero,
- LT and UT do not overlap at any point.

Given a placement of UT and LT conforming to the above conditions, UT starts falling downward such that the base of the triangles remain horizontal during its fall, i.e. UT does not rotate. UT continues to fall, until it touches some point in LT. At this time, UT slides downward (to the left or right) over LT until it cannot slide any further. During this motion, LT is fixed and UT never rotates. Note that UT may have an initial position such that it slides downward either from left or right, and falls below LT (Imagine the old shark in that state!). Also it is possible that the tips of some upper teeth finally pass the line $y = 0$ (the Dracula-style!). Your program should determine whether UT falls down from left or right, or otherwise, finds the final position of the reference point after it stops moving.

Input (filename: F.IN)

The first line of the input file contains a single integer t ($1 \le t \le 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains integer L ($1 \le L \le 10$) the number of triangles in LT, followed by L lines each describing a triangle in LT, containing one integer number b ($1 \le b \le 100$), the side of the triangle. The next line of the input consists of three numbers x, y, and U. The first two numbers are the initial (x, y) coordinates of the reference point and are arbitrary real numbers. U is the number of triangles in UT ($1 \le U \le 10$). After this line, there are U lines each describing a triangle in LT, containing one integer number b ($1 \le b \le 100$), the side of the triangle.

In order to avoid floating-point arithmetic errors, you may assume that the input has the property that during the motion of UT, the distance between tips of any two triangles in LT and UT is never less than 0.1.

Output (filename: F.OUT)

There should be one line per test case containing a pair of real numbers, rounded to three digits after decimal point, which are *x* and *y* coordinates of the reference point after UT stops moving. If UT falls down from left of LT, the output line should contain the word `WM`, and if it falls down from the right of LT, it should be `MW`. The output is considered to be case-sensitive.

Sample Input

```
2
2
10
10
2 20 2
10
10
1
10
50 50 1
10
```

Sample Output

```
5.000 8.660
MW
```

Problem G

(Program filename: G.CPP, G.DPR, or G.PAS)

Skew Binary

It had been a year since Swamp County Computing established a functional programming group. Your (non-functional programming) group is going to throw a surprise party for the anniversary. Now the functional folks really like skew binary numbers for some reason. "Easy to increment and decrement!" they say. Your task is to write a program to convert decimal integers to skew binary in the format they like. This will help in making banners and other party material.

Number representations are made up of a list of digits. Call the lowest order digit the rank 0 digit, the next, rank 1, etc. For example, in decimal representation, digits are 0-9, the rank 0 digit has weight 1, the rank 1 digit has weight 10, and the rank *i* digit has weight 10^i. In binary representation, the digits are 0 and 1, and the rank *i* digit has weight 2^i. In skew binary representation, the digits are 0, 1, and 2, and the rank *i* digit has weight 2^{i+1} -1.

Rank	Weight
0	1
1	3
2	7
3	15
4	31
5	63
6	127
7	255
⋮	⋮

Allowing the digit 2 in the skew binary means there may be several ways to represent a given number. However the convention is that the digit 2 may only appear as the lowest rank non-zero digit. This makes the representation unique.

In this problem, you should use a special way to write skew binary numbers as a list of ranks of non-zero digits in the number. The digit 2 is represented by the rank of the digit appearing twice in the list. Note that this means that *only* the first two ranks in the list may be equal.

Each rank is a decimal integer, and is separated from the next rank by a comma (`','`). A list is started by a `'['` and ended by a `']'`. For example, the decimal number 5, which has the skew representation 12, should be written as `[0,0,1]`. Decimal 0 is an empty list: `[]`.

Input consists of decimal numbers, one per line,

Input (filename: G.IN)

The first line of the input file contains a single integer t ($1 \le t \le 10$), the number of test cases, followed by t lines, each containing a single decimal number with no leading or trailing white space. Each number will be in the range 0...100663270 (inclusive).

Output (filename: G.OUT)

There should be one line per test case containing the input decimal number, with no leading zeros or spaces, a single space, and the skew binary equivalent in list format with no leading or trailing spaces. Within the list each rank should have no extra leading zeros or leading or trailing spaces.

Sample Input

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

Sample Output

```
0 []
1 [0]
2 [0,0]
3 [1]
4 [0,1]
```