UCF Programming Team Practice
September 26, 2015
Developmental Team
Individual Problem Set

Problem Name	**Filename**
Chinary Search	chinary
Programming Contractor	contractor
Cow Routing	cowroute
Cow Routing 2	cowroute2
Hexagon Perplexagon	hexagon
Ali Needs a Hug	hug
Meeting Time	meeting
How Many Perfect Ways to Work?	paths
Sweet!	sweet
It's All About the Base	whatbase

Chinary Search

Filename: `chinary`

A more appropriate name for this problem would be Chisection, but part of the title was lost in translation by our Chinese to English dictionary. Do worry not, problem translated statement perfect.

In Chinese markets, the only communication between the vendors and the customer (you) is through the LCD of a four-function calculator. You push through crowds, scooters and bicycles to find your desired item in the market. You point at it to let the vendor know that you are interested, and she types in a number into the calculator. This initial price, *n*, is not completely unreasonable, but one of the fun parts about being a tourist in China is haggling with the shop keepers, so you punch in a much lower number, *p*, than the one initially presented. The vendor laughs and mentions something about her family and how she has mouths to feed at home. Although she is not yet willing to sell you the item at your low offer, she will be; it just takes patience and persistence. She wants to get as much of your money as she possibly can, so she lowers her price by a percentage, *r*. But, you have been in China for a couple of days and know that the vendor will eventually reduce her price to yours, so you punch your initial offer back in the calculator to let her know that you mean business. Again, she repeats the process of lowering her current price by the percentage *r*, and you again type in your initial offer. This process repeats until the price she enters is less than or equal to your initial offer. How can you sleep at night knowing you haggled her down to such a low price, thus taking food out of her children's mouths?

The Problem:

You will be given three numbers: two integers representing the vendor's initial price of the item and the price that you are willing to pay. The third value will be a floating-point number indicating the percentage the vendor lowers her price each iteration. Your job is to determine how many times the Chinary Search algorithm iterates before you are satisfied.

Here is an example to illustrate how the algorithm works:

n = 150, *p* = 50, *r* = .25

Iteration	*n*	*p*
0	150	50
1	112	50
2	84	50
3	63	50
4	47	50

Notice that in iteration 1, the initial price 150 was dropped by 25% (37.50). This produces a new value for *n*, 112.50, but because prices in China rarely contain fractions of a Yuan (Chinese dollar), the vendor will always round down to set her new price when the new value contains fractions. The answer to this example would be 4.

The Input:

Each line of input will contain 3 numbers (*n, p* and *r)*, separated by one or more spaces ($0 < p < n < 2000$ and $0.001 \leq r \leq 0.999$). The input terminates when *p* equals zero.

The Output:

For each input case, print out the number of iterations required before the Chinary Search yields a result less than or equal to *p*.

Sample Input:

```
150 50 0.25
350 80 0.01
1999 2 0.20
0 0 0.0
```

Sample Output:

```
4
115
28
```

Programming Contractor

Filename: contractor

The Problem

After graduating from UCF with your Computer Science degree, you've decided that you'd like to work for yourself, instead of some big corporation. In starting your business, you find that various companies want to outsource jobs that you can do. For each job, you've become very good at determining the number of days it will take you to finish it. Naturally, each of these jobs comes with a fixed amount of compensation, regardless of how long the work takes. Due to your superior education, you may receive more offers for jobs than you can take. Given the number of days you are willing to work in the year, write a program to determine the maximal amount of money you can make if you accept the appropriate set of jobs.

The Input

The first line of the input file will contain a single positive integer, c $(c \le 1000)$, representing the number of input cases to analyze. The first line of each input case will have two space separated positive integers, n $(n \le 20)$, and d $(d \le 365)$, representing the number of job offers you've received for the year and the number of days you are willing to work during the year, respectively. The next n lines will each contain two space separated integers, d_i $(d_i \le 365)$ and p_i $(p_i \le 1000000)$, representing the number of days and amount of payment, respectively, you would receive if you accepted the ith job.

The Output

For each input case, output a single integer representing the maximal amount of money in dollars you can make by taking a set of the possible jobs that you can finish within the number of days given (or fewer days).

Sample Input

```
2
2 5
3 10000
4 8000
3 100
20 20000
40 50000
40 30000
```

Sample Output

```
10000
100000
```

Problem: Cow routing

(Richard Peng, 2015)

Tired of the cold winter weather on her farm, Bessie the cow plans to fly to a warmer destination for vacation. Unfortunately, she discovers that only one airline, Air Bovinia, is willing to sell tickets to cows, and that these tickets are somewhat complicated in structure.

Air Bovinia owns N planes, each of which flies on a specific "route" consisting of two or more cities. For example, one plane might fly on a route that starts at city 1, then flies to city 5, then flies to city 2, and then finally flies to city 8. No city appears multiple times in a route. If Bessie chooses to utilize a route, she can board at any city along the route and then disembark at any city later along the route. She does not need to board at the first city or disembark at the last city. Each route has a certain cost, which Bessie must pay if she uses any part of the route, irrespective of the number of cities she visits along the route.

Bessie would like to find the cheapest way to travel from her farm (in city A) to her tropical destination (city B). Since she does not want to be confused by a complicated itinerary, she wants to use only a single route. Please help her decide the minimum cost she must pay.

Input

The first line of input will contain a single positive integer, C, the number if input cases to consider.

The first line of each test case contains A (1 <= A <= 10000), B (1 <= B <= 10000), and N (1 <= N <= 500), the starting city, ending city and number of cities, respectively, separated by spaces.

The next 2N lines describe the available routes, in two lines per route. The first line contains the cost of using the route (an integer in the range 1..1000), and the number of cities along the route (an integer in the range 1..500). The second line contains a list of the cities in order along the route. Each city is identified by an integer in the range 1..10,000.

Output

For each input case, output the minimum cost of a single route Bessie can use to travel from city A to city B, on a line by itself. If there is no such route, output -1.

Sample Input	Sample Output

1
1 2 3
3 3
3 2 1
4 4
2 1 4 3
8 5
4 1 7 8 2
	8

Problem: Cow routing II

(Richard Peng and Brian Dean, 2015)

Tired of the cold winter weather on her farm, Bessie the cow plans to fly to a warmer destination for vacation. Unfortunately, she discovers that only one airline, Air Bovinia, is willing to sell tickets to cows, and that these tickets are somewhat complicated in structure.

Air Bovinia owns N planes, each of which flies on a specific "route" consisting of two or more cities. For example, one plane might fly on a route that starts at city 1, then flies to city 5, then flies to city 2, and then finally flies to city 8. No city appears multiple times in a route. If Bessie chooses to utilize a route, she can board at any city along the route and then disembark at any city later along the route. She does not need to board at the first city or disembark at the last city. Each route has a certain cost, which Bessie must pay if she uses any part of the route, irrespective of the number of cities she visits along the route.

Bessie would like to find the cheapest way to travel from her farm (in city A) to her tropical destination (city B). Since she does not want to be confused by a complicated itinerary, she wants to use ***at most two routes.*** Please help her decide the minimum cost she must pay.

Input

The first line of input will contain a single positive integer, C, the number if input cases to consider.

The first line of each test case contains A (1 <= A <= 10000), B (1 <= B <= 10000), and N (1 <= N <= 500), the starting city, ending city and number of cities, respectively, separated by spaces.

The next 2N lines describe the available routes, in two lines per route. The first line contains the cost of using the route (an integer in the range 1..1000), and the number of cities along the route (an integer in the range 1..500). The second line contains a list of the cities in order along the route. Each city is identified by an integer in the range 1..10,000.

Output

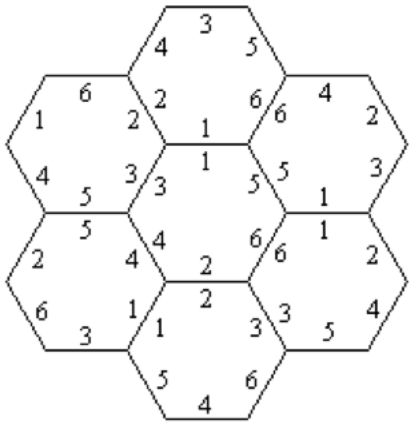
For each input case, output the minimum cost of a single route Bessie can use to travel from city A to city B, on a line by itself. If there is no such route, output -1.

Sample Input	Sample Output

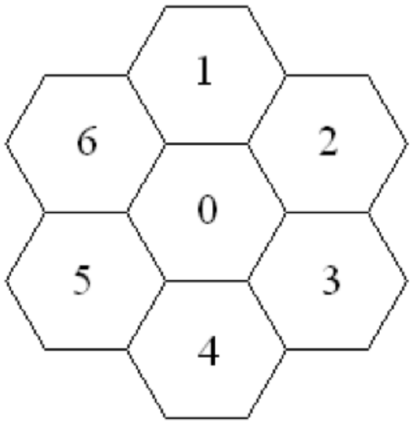
1
1 2 3
3 3
3 2 1
4 4
2 1 4 3
8 5
4 1 7 8 2
	7

Problem C: Hexagon Perplexagon

A well known puzzle consists of 7 hexagonal pieces, each with the numbers 1 through 6 printed on the sides. Each piece has a different arrangement of the numbers on its sides, and the object is to place the 7 pieces in the arrangement shown below such that the numbers on each shared edge of the arrangement are identical. Figure (a) is an example of one solution:



(a) Example Solution



(b) Position Notation for Output

Rotating any solution also gives another trivially identical solution. To avoid this redundancy, we will only deal with solutions which have a 1 on the uppermost edge of the central piece, as in the example.

Input

The first line of the input file will contain a single integer indicating the number of test cases. Each case will consist of a single line containing 42 integers. The first 6 represent the values on piece 0 listed in clockwise order; the second 6 represent the values on piece 1, and so on.

Output

For each test case, output the case number (using the format shown below) followed by either the phrase `No solution` or by a solution specification. A solution specification lists the piece numbers in the order shown in the Position Notation of Figure (b). Thus if piece 3 is in the center, a 3 is printed first; if piece 0 is at the top, 0 is printed second, and so on. Each test case is guaranteed to have at most one solution.

Sample Input

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

Sample Output

```
Case 1: 3 0 5 6 1 4 2
Case 2: No solution
```

Ali Needs a Hug

Filename: hug

Dr. Orooji has spent the last three days at UCF preparing for the High School Programming Contest without food, water, or sleep. Ali needs a hug.

Fortunately, he knows several people at the contest that will gladly hug him if he gets close enough, and they've all gathered in the contest area. Dr. Orooji is still very busy so he will only cross the room once, starting from the west wall and walking only due east. Some people like Ali more than others, and will move farther to hug him than others, so Ali must be careful when deciding where to walk to yield the most of hugs.

The Problem:
Given the locations of Ali's friends in the rectangular room, along with how far each is willing to walk to give Ali a hug, determine the most hugs Ali can get by walking straight across the room from the some point along the west wall directly to the east. Ali is patient, so he will wait for hugs for as long as it takes each person to reach him.

The Input:
There will be multiple data sets. Each data set will begin with a line containing two positive integers, w and h, separated by a single space, which specify the width and height of the contest area for that data. A room of size 0, 0 indicates the end of the input data and should not be processed.

Following the dimensions of the room is a line containing a single integer, P, representing the number of people Ali knows in the contest area. The next P lines each contain three non-negative integers, x, y, and d, separated by spaces. x and y ($0 \leq x \leq w$, $0 \leq y \leq h$) are the x- and y-coordinates of the person in the room (their distance from the west and south walls, respectively). d represents the maximum distance at which they will move to hug Ali.

The Output:
For each data set, print a line containing only the following message:

`Ali can get` *n* `hug(s)!`

Where n is the maximum number of hugs Ali can receive.

Sample Input:
```
10 10
3
5 5 5
2 8 2
10 0 5
0 0
```

Sample Output:
```
Ali can get 2 hug(s)!
```

Problem: Meeting Time

(Nick Wu, 2015)

Bessie and her sister Elsie want to travel from the barn to their favorite field, such that they leave at exactly the same time from the barn, and also arrive at exactly the same time at their favorite field.

The farm is a collection of N fields numbered 1..N, where field 1 contains the barn and field N is the favorite field. The farm is built on the side of a hill, with field X being higher in elevation than field Y if X < Y. An assortment of M paths connect pairs of fields. However, since each path is rather steep, it can only be followed in a downhill direction. For example, a path connecting field 5 with field 8 could be followed in the 5 -> 8 direction but not the other way, since this would be uphill. Each pair of fields is connected by at most one path.

It might take Bessie and Elsie different amounts of time to follow a path; for example, Bessie might take 10 units of time, and Elsie 20. Moreover, Bessie and Elsie only consume time when traveling on paths between fields -- since they are in a hurry, they always travel through a field in essentially zero time, never waiting around anywhere.

Please help determine the shortest amount of time Bessie and Elsie must take in order to reach their favorite field at exactly the same moment.

Input

The first line of input will contain a single positive integer, C, the number if input cases to consider.

The first line of each test case contains N (1 <= N <= 16) and M (1 <= M <= N(N-1)/2), the number of fields and number of paths connecting fields, respectively, separated by spaces.

Output

For each input case, output a single integer, giving the minimum time required for Bessie and Elsie to travel to their favorite field and arrive at the same moment. If this is impossible, or if there is no way for Bessie or Elsie to reach the favorite field at all, output the word IMPOSSIBLE on a single line.

Sample Input	Sample Output

2
3 3
1 3 1 2
1 2 1 2
2 3 1 2
3 3
1 2 10 1
1 3 15 2
2 3 10 1
	2
	IMPOSSIBLE

How Many Perfect Ways to Work?

Filename: `paths`

Nick enjoys the beauty and perfection in nature. Unfortunately, Nick lives in a huge city that is mostly devoid of the trees and serenity of nature. Instead, Nick's city has many, many roads set up in a perfect grid, so he must settle for finding perfect paths within this grid. In the grid, a path consists of a sequence of movements in one of four possible directions: north(N), south(S), east(E) or west(W). Assume that each movement in a direction is a single block. A perfect path does not contain any movements in directly opposite directions. Thus, the sequence of movements NNENE forms a perfect path, but the sequence NEESN does not, since it contains both a north and south movement. To complicate matters, there are some street intersections that have graffiti which Nick considers imperfect. Thus, any path that goes through these intersections is not a perfect path.

The Problem:

Given grid coordinates for Nick's starting location and destination, as well as coordinates of all the imperfect intersections Nick is to avoid, you are to determine the number of perfect paths Nick can take.

The Input:

There will be several sets of input. The first line will contain a single positive integer n $(n < 100)$ describing the number of test cases in the data set. The first line in each data set has a single integer m $(0 \leq m < 10)$, which represents the number of intersections in the town for that particular data set that Nick must avoid. The following m lines contain the coordinates (x, y) of the m intersections to avoid. All x and y coordinates will be non-negative integers less than 10, with the x coordinate appearing first on a line, followed by the y coordinate, separated by a single space. The next line in the data set will be a single positive integer p $(0 < p < 10)$ that represents the number of trips for which you will be calculating the number of perfect paths for that data set. The last p lines of the data set contain the p trips. Each line will contain two pairs of (x,y) coordinates. The first pair will be the coordinates for Nick's starting location and the second pair will be the coordinates for Nick's destination. Each coordinate on each line is separated by a single space from the previous and subsequent coordinates. You are guaranteed that none of Nick's starting locations or destinations will be an intersection he is supposed to avoid. All of these coordinates will also be non-negative integers less than 10 and be separated by spaces on each line.

The Output:

For each data set, you will output a single line header of the following format:

`Data Set` *`k`*`:`

where k is an integer in between 1 and n, inclusive.

Follow this with a blank line, and then *p* lines, each with one of the two following formats.

```
  Test Case c: Nick can take P perfect paths.

  Test Case c: Nick can take 1 perfect path.
```

where *c* is an integer in between 1 and *p*, inclusive and `P` will be a non-negative integer less than 2147483647. Use the second format only if `P`=1. Please indent each of these lines exactly 2 spaces from the left margin. Also, leave a blank line in between data sets.

Sample Input:

```
2
4
2 2
3 5
1 0
4 4
5
0 0 1 9
0 1 2 3
2 1 0 5
0 1 4 5
0 0 0 0
3
1 1
2 2
3 3
1
4 4 9 7
```

Sample Output:

```
Data Set 1:

  Test Case 1: Nick can take 9 perfect paths.
  Test Case 2: Nick can take 3 perfect paths.
  Test Case 3: Nick can take 5 perfect paths.
  Test Case 4: Nick can take 0 perfect paths.
  Test Case 5: Nick can take 1 perfect path.

Data Set 2:

  Test Case 1: Nick can take 56 perfect paths.
```

Sweet!

Filename: `sweet`

Brandon loves to play on his Nintendo GameCube. He's always trying to find ways to get the latest game, adding up his allowance as quickly as possible and often lobbying for more. Birthday money also comes in handy. Whenever he gains enough for the latest game he wants, he yells "Sweet!" and heads off to the store with his parents to buy it.

The Problem:

Given the various amounts of money Brandon receives in allowances and gifts, determine at which points Brandon has enough to buy a new game. Assume all games cost $50 and that Brandon keeps the change left over after a purchase (which he will put towards the next game).

The Input:

The first line will contain a positive integer, *n*, indicating the number of scenarios. For each scenario, there will be a line containing a single integer, *m*, which represents the number of money transactions given to Brandon. On each of the next *m* lines there will be a single positive integer representing an allowance or gift. Each amount will be less than or equal to $50.

The Output:

For each scenario, determine whether Brandon can purchase a game after receiving the money from each transaction. Begin the output for each scenario with a header "`Scenario` *`i`*`:`" where *i* begins with 1 and increases for each scenario. For each transaction, if Brandon cannot purchase a game yet, output "`Bummer, I need to wait.`" or output "`Sweet!`" if he can. Leave a blank line between the output for each scenario.

Sample Input:

```
2
1
25
3
30
30
40
```

Sample Output:

```
Scenario 1:
Bummer, I need to wait.

Scenario 2:
Bummer, I need to wait.
Sweet!
Sweet!
```

Problem: It's All About the Base

(Brian Dean, 2015)

Bessie the cow has been taking computing classes at her local college (or "cow-ledge", in her case), and she has been very excited to recently learn about writing numbers in different bases.

Recall that a number written in base B has digit places representing 1, B, B^2, B^3, and so on from right to left. For example, in our familiar base 10 numbering system, we have digits representing 1's, 10's, 100's, 1000's and so on. The sequence of digits 1234, interpreted in base 10, really means 1(1000) + 2(100) + 3(10) + 4(1). The same sequence of digits 1234, interpreted in base 5, would mean 1(125) + 2(25) + 3(5) + 4(1), which adds up to the number 194 in base 10. Bessie notices that if the base increases, so does the number represented by a sequence of digits -- for example, 1234 in base 7 represents a larger number than 1234 in base 6.

When writing numbers in base B, each digit can range from 0 through B-1, so for example in base 10 each digit is in the range 0..9, and in base 5 each digit is in the range 0..4. It is entirely possible to consider bases larger than 10. Computer scientists often use base 16 ("hexadecimal"), where the letters A..F represent digits of values 10..15. For example, BEEF in hexadecimal corresponds to 11(4096) + 14(256) + 14(16) + 15, which adds up to the number 48879 in base 10.

Bessie is intrigued by the concept of using bases much larger than 10. She takes a number N and writes it down in two different bases X and Y, where both X and Y are in the range 10..15,000. Interestingly, in both cases, she gets a sequence of 3 digits, each of which happens to be only in the range 1..9. Unfortunately, due to Bessie's poor memory, she has now forgotten N, X, and Y! Given just the two 3-digit sequences she wrote down, please help her figure the two bases X and Y that she used.

Note that due to the potential size of X and Y, a program that searches exhaustively over every possible value of X and Y (nearly 15,000^2 possibilities!) will not run within the time limit.

Input

The input file starts with an integer C, then it contains C lines each specifying a separate test case. Each test case consists of two 3-digit numbers. The first is a number N written in base X, and the second is N written in base Y (N, X, and Y are possibly different for each test case).

Output

Your output should contain K lines, one for each test case. On each line, output the two numbers X and Y for the relevant test case, separated by a space. A unique solution for each case is guaranteed to exist.

Sample Input	Sample Output

2
419 792
358 655
	47 35
	2213 1565