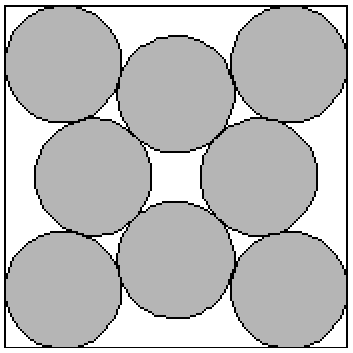
UCF Local Contest — August 31, 2013

Circles Inside a Square

filename: `circle`

You have 8 circles of equal size and you want to pack them inside a square. You want to minimize the size of the square. The following figure illustrates the minimum way of packing 8 circles inside a square:



The Problem:

Given the radius, *r*, find the area of the minimum square into which 8 circles of that radius can be packed.

The Input:

The first input line contains a positive integer, *n*, indicating the number of test cases. Each test case consists of a positive real number (between 0.001 and 1000, inclusive) in a single line denoting the radius, *r*.

The Output:

For each test case, output the area of the minimum square where 8 circles of radius *r* can be packed. Print 5 digits after the decimal. Your output is considered correct if it is within ±0.00001 of the judge's output.

Sample Input:

```
2
0.1
0.2
```

Sample Output:

```
0.34383
1.37532
```

Knightmare

filename: `knightmare`

Probably the night before this contest you had a great sense of anticipation. While sleeping, you had a horrible nightmare about an impossibly difficult chess problem. Well, here it is!

The Problem:

Given a large number of knights on an infinite chessboard, determine the number of squares each of which is currently being threatened by k or more knights. Unfortunately for you, these are not normal knights! They can move in the following way. Each knight has two values, a and b. For any two integers, i $(1 \le i \le a)$ and j $(1 \le j \le b)$, the knight can move to any of the following squares relative to its current location: (i, j), $(-i, j)$, $(i, -j)$, $(-i, -j)$, $(-j, -i)$, $(-j, i)$, $(j, -i)$, (j, i). Any of these squares are considered threatened.

The Input:

The first line of the input contains a single positive integer, n, representing the number of boards to analyze. Each board starts with a new line containing two integers, p $(1 \le p \le 10^5)$ and k $(1 \le k \le 10)$, representing the number of knights and the value k from above, respectively. This is followed by p lines representing each knight. Each of these p lines contains four integers, r, c, a and b $(0 \le r \le 10^9;\ 0 \le c \le 10^9;\ 1 \le a \le 10^9;\ 1 \le b \le 10^9)$, representing the row and column the knight occupies as well as its a and b values (from above).

The Output:

For each board, output the number of squares each of which threatened by k or more knights.

Sample Input:

```
3
2 10
10 10 2 1
10 10 1 2
1 1
10 10 2 1
2 2
10 10 2 2
13 10 2 2
```

Sample Output:

```
0
12
8
```

UCF Local Contest — August 31, 2013

Fixing Traffic

filename: `traffic`

Traffic in Orlando has become unbearable. Assuming *m* intersections in the city, shipping magnate O. Marhem Ali always sends trucks from the city downtown (intersection 0) to intersection *m-1*. Ali has generously agreed to donate costs to widen one street in town in order to increase traffic flow during rush hour. Of course, Mr. Ali benefits as well, since more of his trucks will be able to get from point 0 to point *m-1*. Also, note that he has enough resources to widen any street so that it no longer becomes the bottleneck of traffic flow. As in other cities, each street in Orlando consists of one or more street segments. When a street is widened, all of its segments are widened.

Being the only programmer on city council, your fellow council members are counting on you to determine which street the city council should choose to widen to maximize the flow of traffic at rush hour. Since O. Marhem Ali is vastly wealthy, you must consider processing queries for other cities where he is rumored to have made the same offer.

The Problem:

Given an existing map of street segments between intersections, the maximal flow of traffic (in cars per minute) through all street segments, the starting and ending point of all travelers during rush hour, as well as the names of each street, determine which street should be widened to produce the maximal increase in traffic flow and how much traffic flow will be increased when that street is widened.

The Input:

The first input line contains a positive integer, *n* $(n \leq 50)$, indicating the number of cities to evaluate. The description of each city map follows. Each city map starts with two space separated integers on a line, *m* $(3 \leq m \leq 50)$, which is the number of intersections for the given map, and *e* $(2 \leq e \leq 200)$, the number of street segments for the city. Assume that the intersections are numbered 0 through *m-1* and that the city downtown is at intersection 0 and all traffic flows to intersection *m-1*. The following *e* lines will contain information about each street segment. Each of these lines will contain the starting intersection, ending intersection, the current traffic flow, and street name, all comma separated. The starting and ending intersections will be integers in between 0 and *m-1*, inclusive, such that no street connects intersection 0 to intersection *m-1* and no flow through any street segment will exceed 100 and all of these flows will be positive integers. All street names will consist of only letters, spaces, and periods (with a maximum length of 20); there will be no leading/trailing spaces in street names.

You may assume that no single street connects the starting point to the ending point and that all street segments that comprise a street will be connected. More formally, all street segments comprising a single street in the input can be arranged such that there are a set of intersections, $v_1, v_2, v_3, \ldots v_k,$ where the street segments are the directed edges $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$.

Note that you are allowed to increase the traffic flow in each street segment of one single street as much as possible, well beyond 100, since O. Marhem Ali is *that* rich!

The Output:

For each city, output the name of the street to widen, followed by a space, followed by the increase in traffic flow due to widening that street. You are guaranteed that each city will produce one unique answer.

Sample Input:

```
2
5 6
0,1,2,Smith Street
0,2,5,Campus Blvd.
0,3,7,Main St.
1,4,12,Brown Avenue
2,4,4,Oakberry Circle
3,4,6,Division Ave.
9 12
0,1,1,Campus Drive
1,2,2,Campus Drive
0,3,3,University Blvd.
1,4,4,Main Street
2,5,5,MLK Blvd.
3,4,6,Gemini Dr.
4,5,7,Gemini Dr.
3,6,8,University Blvd.
4,7,9,Main Street
5,8,10,MLK Blvd.
6,7,11,Mills Ave.
7,8,12,Mills Ave.
```

Sample Output:

```
Smith Street 10
University Blvd. 14
```

UCF Local Contest — August 31, 2013

Come Minion!

filename: `minion`

Welcome treasure hunter! I am the great and powerful interplanetary ninja and you are my minion. We must hurry to save this planet from the evil, "Dialog Not Found". We must travel across the land so that I may prevent "Dialog Not Found" from their evil plans. Also just so you know, minion, I am unable to triumph over a great many trials. Some of these include being looked at, handshakes, and even stairs.

The Problem:

Given a map of locations, routes between locations, and the trials which exist along routes, help the interplanetary ninja reach their target. You must avoid any of the trials that the ninja is unable to triumph over. Then tell the interplanetary ninja if they are able to reach their target and save the world.

The Input:

The first input line contains a positive integer, *m*, indicating the number of maps to check. Each map will start with an integer on a new line, *t* ($0 \le t \le 50$), that describes the number of trials that the ninja is unable to accomplish. On the next *t* lines these trials are listed, one per line. The following line will contain two integers, *n* ($2 \le n \le 30$) and *e* ($0 \le e \le 500$). *n* indicates the number of locations on the map (numbered 0 through *n*-1) and *e* indicates the total number of routes between locations on the map. Assume the ninja's starting location number is 0 and the ninja's target location is *n*-1.

The following *e* lines each describe a route between a pair of locations, and the trial on that route. These lines will consist of two integers L_a and L_b and a string Q. The ninja can travel between location L_a and location $L_{b,}$ or between L_b and $L_{a,}$ as long as he does not have the trial (Q) on that route. For example, if ninja has the trial "xyz" and the route also has the trial "xyz", then ninja cannot travel on that route. Assume that there is at most one route between any two locations and exactly one trial for a route.

All locations are numbered from 0 to *n*-1, inclusive. All trials are named using only lowercase letters, 1 to 20 in length. If a trial is not on the "unable to accomplish" list of ninja, then the ninja will be able to accomplish it. Successive values on a line are separated by exactly one space. There are no leading or trailing spaces on any line.

The Output:

For each map, output a line that contains only a 1 if the ninja can reach his target and a 0 if the ninja is unable to reach his target. Each answer must be on a separate line.

(Sample Input/Output on the next page)

Sample Input:

```
2
3
stairs
talking
staring
4 5
0 3 talking
0 1 abc
0 2 xyz
1 3 stairs
2 3 staring
3
fire
water
people
4 5
0 1 abc
0 2 water
1 2 fire
1 3 xyz
2 3 abc
```

Sample Output:

```
0
1
```

UCF Local Contest — August 31, 2013

Jumping Frog

filename: `jump`

A frog is located at the coordinate (x_1,y_1). He wants to go to the coordinate (x_2,y_2). He will perform one or more jumps to reach his destination. The rule of the jumping is as follows: Suppose the frog is located at the coordinate (x,y); then he can jump to the following four squares:

1. (x+y,y)
2. (x-y,y)
3. (x,y+x)
4. (x,y-x)

The Problem:

Given the coordinates (x_1,y_1) and (x_2,y_2), you need to determine if it is possible for the frog to travel from (x_1,y_1) to (x_2,y_2) through a series of jumps as described.

The Input:

The first input line contains an integer, *n* $(1 \le n \le 100)$, indicating the number of test cases. Each test case consists of four integers (between -1,000,000,000 to +1,000,000,000 inclusive) separated by a single space denoting x_1, y_1, x_2 and y_2, respectively.

The Output:

For each test case, output 1 if the frog can travel from (x_1,y_1) to (x_2,y_2) through a series of jumps as described or 0 otherwise.

Sample Input:

```
3
-6 8 17 25
13 17 -16 11
0 0 5 6
```

Sample Output:

```
0
1
0
```

UCF Local Contest — August 31, 2013

Gold Rush

filename: `goldrush`

After learning about the gold rush in school, your friends have decided to have a gold mining tournament. This one-of-a-kind tournament will be the largest of its kind!

For the tournament, everyone will all line up in a row. Every pair of consecutive people will then face off in a "round" of the tournament. Assuming there are 7 people in the tournament, the first round would look something like this when pairing off: (1, 2) (3, 4) (5, 6) (7).

Notice that when there are an odd number of people in a round, the last person automatically advances to the next round. The remaining players in that round play a best-of-k format where k is an odd number of games. The first person to win ceiling(k/2) games advances to the next round, the other player is eliminated from the tournament, and no more games are played between those players.

For example, let's say players 2, 3, 5, and 7 advance from the first round above. The following round then pairs off in order and the match ups look like this: (2, 3) (5, 7).

The rounds continue in this fashion until a single player remains, the tournament winner!

You would like to know, for a given tournament layout, how many ways the tournament records can play out. Two tournament records are considered different if one of the following conditions is met:

1. The number of games played by a player in the tournament is different.
2. The outcome of the i-th game played by a player is different.

The Problem:

Given the number of players in a tournament and its format (as in best-of-k), determine the number of possible win-loss records.

The Input:

The input begins with a single positive integer, t, representing the number of tournaments to evaluate. Each of the next t lines contains two integers, n ($1 \le n \le 10^{17}$) and k ($1 \le k \le 10^{17}$), representing the number of players and the best-of-k format, respectively.

The Output:

For each tournament, output the number of possible win-loss records. As this number may be quite large, output the result modulo 1,000,003.

(Sample Input/Output on the next page)

Sample Input:

```
2
2 7
3 1
```

Sample Output:

```
70
4
```

UCF Local Contest — August 31, 2013

Buying in Bulk

filename: `bulk`

To encourage customers to shop more, some stores charge lower prices if you buy multiples of an item. For example, if you buy one, it may cost you $5 but if you buy two, it will cost you $8 instead of $10.

The Problem:

Let's assume a store provides discounts as follows:

1. No discount if you buy only one.
2. $2 discount for each additional item if you buy more than one.

Given the number of items a customer has purchased and the price for one item, you are to compute the total cost for the customer.

The Input:

The first input line contains a positive integer, *n*, indicating the number of customers to check. The customers are on the following *n* input lines, one customer per line. Each line provides two integers; the first integer c ($1 \le c \le 100$) is the number of items purchased by the customer, and the second integer p ($3 \le p \le 50$) is the price for one item.

The Output:

For each customer, print two lines of output. The first line will contain the two input values separated by a single space. The second output line will contain the total cost for the customer. There should be no leading or trailing spaces on any output line.

Sample Input:

```
2
1 5
3 10
```

Sample Output:

```
1 5
5
3 10
26
```

UCF Local Contest — August 31, 2013

Knights Airways

filename: `airways`

"This is Knights Airways Flight 472 requesting permission to take off…over…kshh." Knights Airways was recently created to take UCF Knights to their vacation destinations at a cost every college student can afford. One day, while trying to get to his favorite travel spot, Knights Airways owner Uppin Diare realized there was a problem. His flight from Orlando arrived into Tampa on time, but his connecting flight from Tampa to Miami had already taken off. Not wanting other travelers to experience this, Mr. Diare wants flights to be ordered in such a way that no passenger would miss their connections. Mr. Diare has asked you to determine a flight ordering that would prevent people from experiencing the annoyances of a missed connection. Note that this flight ordering will dictate from which cities passengers can fly to which other cities. For example, if the flight ordering puts the flight from City A to City B before the flight from City B to City C, then passengers can fly from A to B, B to C, and A to C.

The Problem:

Given a list of flights, determine an ordering that would allow all passengers flying into a city to arrive before the departure of the flights leaving the same city. You may assume that no traveler, once departing a city, can make any number of connecting flights and return back to their origin.

The Input:

The first input line contains a positive integer, *n*, indicating the number of flight schedules to order. The description of each flight schedule is as follows: Each flight schedule will start with a line containing a single integer, f $(1 \leq f \leq 10{,}000)$, which is the number of flights for the given schedule. The following *f* lines will contain the origin city, destination city, and flight number for each flight, each separated by a single space. There will be no leading or trailing spaces on any input line. All cities will consist of up to 25 uppercase letters. Multiple flights from the same origin city to the same destination city may exist, but will have different flight numbers. All flight numbers for a given flight schedule will be unique positive integers less than 100,000 with no leading zeros.

The Output:

For each flight schedule, on a single line, output the ordered flight numbers separated by a space. If there are multiple ways of ordering the flights, choose the ordering that comes first numerically based on flight numbers. You are guaranteed that each flight schedule will produce one unique answer. Note that all the flights in the input will be in the output. In particular, when there are multiple flights from the same origin city to the same destination city, every one of these flights will be listed in the output.

(Sample Input/Output on the next page)

Sample Input:

```
2
7
TAMPA MIAMI 912
ORLANDO TAMPA 450
MIAMI ATLANTA 321
ATLANTA NEWYORK 942
ORLANDO ATLANTA 952
CHICAGO MIAMI 568
MIAMI ATLANTA 64
3
ATLANTA TAMPA 351
MIAMI NEWYORK 135
ORLANDO CHICAGO 531
```

Sample Output:

```
450 568 912 64 321 952 942
135 351 531
```

LIS Number

filename: `lis`

Let A be a sequence of integers. The *LIS Number* of A is the smallest positive integer L such that A can be obtained by concatenating L strictly increasing sequences. For example, the *LIS Number* of A = {1, 4, 4, 2, 6, 3} is 4, since we can obtain A as {1, 4} + {4} + {2, 6} + {3}, and there is no way to create A by concatenating 3 (or fewer) strictly increasing sequences. The *LIS Number* of a strictly increasing sequence is 1.

The Problem:

You are given a sequence of length N and an integer K. You want to transform the given sequence into a sequence with *LIS Number* K. The only operation you are allowed to do is to delete 0 or more numbers from the original sequence. Count how many ways you can do that. Two ways are different if the set of removed numbers (their indices/positions) are different.

The Input:

The first input line contains a positive integer, *t*, indicating the number of test cases. First line of each test case consists of two integers *N* $(1 \leq N \leq 50{,}000)$ and *K* $(1 \leq K \leq 10)$. The second line contains *N* integers of the sequence (separated by a single space). These integers will be between 0 and 100000, inclusive.

The Output:

For each test case, output the number of ways you can transform the given sequence of length N into a sequence with LIS Number K. Since the number of ways can be too large, output the result modulo 1,000,000,007.

(Sample Input/Output on the next page)

Sample Input:

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

Sample Output:

```
31
5
10
0
```

UCF Local Contest — August 31, 2013

Blackout

filename: `blackout`

Percy is very afraid of ghosts. He decided that because ghosts are so scary he would like to add a generator to his house so that, when the power goes out in his neighborhood, he will not be stuck in the dark with scary ghosts regardless of where he is in his house. In fact, Percy is so paranoid he would like to install such generators throughout the houses in his neighborhood.

In each house, there are a number of lights. Each light has an efficiency value, p. When the generator generates current, c, the light will illuminate an area of radius $p*c$. Given this, for each house it is possible to determine what the minimum current required to light up the entire house is.

While this is okay for some people, Percy is extra paranoid and also would like to know that if k lights went out within a house, what is the minimum amount of current needed to still keep the whole house lit (regardless of which k lights went out).

The Problem:

Given the size of a house and the number of bulbs contained within it, determine how much current, c, is needed to illuminate the entire house if any k bulbs went out.

The Input:

The first line of the input contains a single positive integer, n, representing the number of houses. Each house begins on a new line that contains four integers: b ($1 \le b \le 100$), k ($0 \le k < b$), w and l ($2 \le w \le 20000$; $2 \le l \le 20000$), representing the total number of bulbs, the number of bulbs that can go out, and the width and length of the house, respectively. Each of the next b lines will contain three integers each. These represent the (x,y) locations ($0 < x < w$; $0 < y < l$) of the bulbs within the house and the power value, p ($0 < p \le 20000$), for that bulb.

The house is rectangular and axis-aligned from (0, 0) to (w, l).

The Output:

For each house, output a number to two decimal places representing the minimum current needed such that, regardless of which k bulbs burn out, the whole house will be lit. Your output is considered correct if it is within ±0.01 of the judge's output.

(Sample Input/Output on the next page)

Sample Input:

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

Sample Output:

```
1.41
1.45
```

UCF Local Contest — August 31, 2013

Are We Stopping Again?

filename: `stopping`

Going on a road trip is an adventure for Dr. Orooji and his family. Obviously he has to stop to refuel the car, but he also stops whenever his kids want to eat. Dr. O needs to figure out the number of stops before going on the trip so he is mentally prepared.

The Problem:

Find the total number of stops for Dr. O's trip, given:

1. Total miles to be traveled.
2. How often he stops for gas (in miles).
3. How often he stops for food (in miles).

Assume that the car's gas tank is full at the beginning of the trip and the kids are full as well. If the destination happens to be the time to refuel (or eat), do not count it as a stop. Also, if a particular mileage happens to be both refueling time and eating time, count it as one stop and not two stops.

Note that if a particular mileage happens to be refueling time only, kids won't eat at that stop. Similarly, if a particular mileage happens to be eating time only, the car is not refueled at that stop.

The Input:

The first input line contains a positive integer, *t*, indicating the number of trips to check. The trips are on the following *t* input lines, one trip per line. Each trip provides three integers (each between 1 and 1000, inclusive); these are the three values specified in order above.

The Output:

For each trip, output the three input values. Then, on the next output line, print the number of stops for the trip.

(Sample Input/Output on the next page)

Sample Input:

```
3
100 30 40
10 5 1
20 3 4
```

Sample Output:

```
100 30 40
5
10 5 1
9
20 3 4
9
```

UCF Local Contest — August 31, 2013

Sub Matrix Sum

filename: `sum`

You have written many programs to search mazes so matrix search shouldn't be any different, or will it?

The Problem:

An integer matrix with R rows and C columns has $\binom{R}{2}\binom{C}{2}$ sub matrices. We want to select a sub matrix with sum (the sum of all integers in it) greater than or equal to a given integer S. We want the size of the sub matrix to be the least possible. The size of a sub matrix is defined as the number of elements in that sub matrix (i.e., number of rows * number of columns in that sub matrix).

The Input:

The first input line contains a positive integer, *t*, indicating the number of test cases. The first line of each test case consists of three integers *R*, *C* ($1 \le R \le 100{,}000$; $1 \le C \le 100{,}000$; $1 \le R*C \le 100{,}000$) and *S*. Next R lines contain the description of the matrix. Each of these R lines contains C integers separated by a single space. All integers (other than R and C) are between -10^9 and $+10^9$, inclusive.

The Output:

For each test case, output the size of the minimum sub matrix whose sum is greater or equal to the given S. If there is no such sub matrix, output -1.

(Sample Input/Output on the next page)

Sample Input:

```
3
3 3 26
1 2 3
4 5 6
7 8 9
3 3 0
-1 -2 -3
-4 -5 -6
-7 -8 -9
2 2 1
-1 -2
0 2
```

Sample Output:

```
4
-1
1
```