# 2011 the 36th ACM Collegiate Programming Contest Asia Regional

Hsinchu Site

November 26, 2011

- **Problems:** There are 9 problems in this set.
- **Problem Input:** The Inputs to the problems are through the input files. The file names are given in the following table. Each input file may contain one or more test cases. Test cases may be separated by any delimiter as specified in the problem statements.
- **Problem Output:** All output should be directed to standard output (screen output).
- **Time Limit:** The judges will run the submitted program with certain time limit as shown on the problem pages.

| | Problem Name | Input File |
|---|---|---|
| Problem A | Hidden Terminal Problem | pa.in |
| Problem B | City Travel | pb.in |
| Problem C | Probability Computation | pc.in |
| Problem D | Register Allocation | pd.in |
| Problem E | Finding Bottleneck Shorstet Paths | pe.in |
| Problem F | Robot Arm Planning | pf.in |
| Problem G | Finding Feasible Paths | pg.in |
| Problem H | KGold | ph.in |
| Problem I | Airport | pi.in |

# Problem A
## Hidden Terminal Problem
Input File: *pa.in*
Time Limit: *1 second*

## Problem Description

Ad hoc networks are wireless networks with no fixed infrastructure. Each device in the network functions as a router that discovers and maintains routes for other nodes. Suppose that the communication range (radius) of each device is $R$ unit length. Device $A$ can communicate with device $B$ directly, when the distance between $A$ and $B$ is less than or equal to $R$. Specifically, device $A$ can communicate with device $B$ directly if $(x_1 - x_2)^2 + (y_1 - y_2)^2 \leq R^2$, where $(x_1, y_1)$ and $(x_2, y_2)$ are, respectively, the locations of $A$ and $B$.

The hidden terminal problem in ad hoc networks is caused by concurrent transmissions of two devices that cannot communicate with each other directly, but transmit to the same destination. For example, device $A$ cannot communicate with device $B$ directly (i.e. the distance between $A$ and $B$ is greater than $R$), but device $A$ can communicate with device $C$ directly (i.e. the distance between $A$ and $C$ is less than or equal to $R$) and device B can communicate with device $C$ directly. Such three devices are referred to here a hidden-terminal set. For example, there are four devices $A$, $B$, $C$, $D$, deployed in the plane at (0, 0), (0, 1), (1, 0), (1, 1), respectively. Given that the radius of each device is $R = 1$ unit length, $A$ can communicate with $B$ ($C$) directly, and similarly $D$ can communicate with $B$ ($C$) directly. However, $A$ cannot communicate with $D$ directly and $B$ cannot communicate with $C$ directly. The number of different hidden-terminal sets in the plane is four (i.e., $\{A, B, C\}$, $\{A, B, D\}$, $\{A, C, D\}$, and $\{B, C, D\}$).

Hidden-terminal sets in an ad hoc network seriously results in garbled messages and increases communication delay, thus degrading system performance. Given $N$ devices deployed in a plane, please compute the number of different hidden-terminal sets in the network.

## Input File Format

There are at most 10 test cases. The first line of each instance consists of an integer $N (3 \leq N \leq 100)$, where $N$ is the number of devices in the network. The second line of each instance consists of an integer $R (1 \leq R \leq 100)$, where $R$ is the communication range (radius) of each device. Each of the following $N$ lines consists of two integers $x$ and $y$ (separated by a space) $(-99 \leq x \leq 99, -99 \leq y \leq 99)$, which indicate the location ($x$-coordinate and $y$-coordinate) of a device in the plane. The last test case will be followed by a line containing $N = 0$.

## Output Format

The output for each instance should contain an integer denoting the number of different hidden-terminal sets in the network.

## Sample Input

```
4
1
0 0
0 1
1 0
```

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

## Sample Output

```
4
3
```

# Problem B
## City Travel
Input File: *pb.in*
Time Limit: *1 second*

## Problem Description

We have $n$ cities that are connected by $m$ bi-direction roads. Each city has a unique id from 1 to $n$, so we have cities $c_1, \ldots, c_n$. Let $r_{ij}$ be the road connecting two cities $c_i$ and $c_j$, then $r_{ij}$ has a length $l_{ij}$ and condition $d_{ij}$, where the length is a positive integer and the condition is an integer from 1 to $k$. Note that all roads are bidirectional so $l_{ij}$ is equal to $l_{ji}$ and $d_{ij}$ is the same as $d_{ji}$.

We want to drive a car from a starting city $c_s$ to a destination city $c_d$ along a shortest route. However, the condition of a road has a severe impact on the suspension system of our car. That is, after driving through a road of condition $c$, we *cannot* drive through another road of condition $c$ *immediately*, otherwise the suspension of our car will break down. For example, if road $r_{12}$ has condition 1 and $r_{23}$ also has condition 1, then we cannot drive from $c_1$ to $c_2$ to $c_3$. However, if road $r_{12}$ has condition 1, $r_{23}$ has condition 2, and $r_{34}$ also has condition 1, then we can drive from $c_1$ to $c_2$ to $c_3$ to $c_4$, since we did not drive through two roads of the same condition consecutively.

Now given the condition and length of all roads and $p$ pairs of starting and destination city pairs, please compute the shortest path for each pair of cities without going through two consecutive roads of the same condition.

## Technical Specifications

1. The number of cities $n$ is no more than 50.
2. The number of roads $m$ is no more than 500.
3. The number of conditions $k$ is no more than 50.
4. The number of source and destination city pairs $p$ is no more than 15.
5. The length of each $l_{ij}$ is at most $2^{15}$.

## Input File Format

The first line of the input file contains an integer, denoting the number of test cases to follow. The first line of each test case has the number of cities $n$, the number of roads $m$, the number of conditions $k$, and the number of city pairs $p$ to compute the distance. Each of the next $m$ lines has the information of a road, including the ids of the starting city $i$ and the destination city $j$, the distance $l_{ij}$ and the condition $d_{ij}$. To simplify the input presentation we also assume that $i < j$ so that a road will appear exactly once in the input. Each of the next $p$ lines has the source and destination city pairs to compute the distance.

## Output Format

For each test case, output the length of the shortest route from the starting city to the

destination city without going through two consecutive roads of the same condition for the $p$ city pairs. If there are no feasible routes between a pair of cities, please output "infinity".

## Sample Input

```
2
5 5 3 5
1 2 1 1
2 3 100 2
3 4 100 3
4 5 1 1
2 4 1 2
1 4
1 5
2 4
2 5
3 5
5 5 2 5
1 2 1 1
2 3 100 2
3 4 100 2
4 5 500 1
2 4 1 1
1 4
1 5
2 4
2 5
3 5
```

## Sample Output

```
2
3
1
2
101
infinity
infinity
1
infinity
600
```

# Problem C
## Probability Computation
Input File: *pc.in*
Time Limit: *1 second*

## Problem Description

A random binary number $X$ contains $n$ independent random binary digits (bits) denoted by $b_1, b_2, b_3, \cdots, b_n$, where $b_1$ is the most significant bit and $b_n$ is the least significant bit. That is, the value of $X$ is $b_1 2^{n-1} + b_2 2^{n-2} + b_3 2^{n-3} + \cdots + b_n 2^0$. For each $i$, the random bit $b_i$ is 1 with probability $p_i$ percents ($0 \le p_i \le 100$) and $b_i$ is 0 with probability $(100 - p_i)$ percents. Given the integer $n$ ($1 \le n \le 200$), the integers $p_1, p_2, p_3, \cdots, p_n$, and the integers $Q$ ($2 \le Q \le 99$) and $R$ ($0 \le R < Q$), your program should output the probability of the event that $X \ mod \ Q$ is equal to $R$, where *mod* is the modulus operation. In other words, your program should output the probability $Pr\{X \ mod \ Q = R\}$. The output probability must be rounded to 5 digits after the decimal point.

For example, consider a test case with $(n, p_1, p_2, p_3, p_4, Q, R) = (4, 0, 90, 100, 80, 5, 3)$. Your program should output 0.08000, since

$Pr\{X \ mod \ 5 = 3\}$
$= Pr\{X = 3\} + Pr\{X = 8\} + Pr\{X = 13\}$
$= Pr\{(b_1 b_2 b_3 b_4) = (0011)\} + Pr\{(b_1 b_2 b_3 b_4) = (1000)\} + Pr\{(b_1 b_2 b_3 b_4) = (1101)\}$
$= (100 - 0)\% \cdot (100 - 90)\% \cdot 100\% \cdot 80\% + 0\% \cdot (100 - 90)\% \cdot (100 - 100)\% \cdot (100 - 80)\% + 0\% \cdot 90\% \cdot (100 - 100)\% \cdot 80\%$
$= 0.08000$

**Note**: The above example is for explanation. The straightforward algorithm in the example may not meet our time constraint when input integers are much larger. You should develop another **more efficient algorithm**.

## Technical Specification

The ranges of input integers are: $1 \le n \le 200$, $0 \le p_i \le 100$ for each $i$, $2 \le Q \le 99$, and $0 \le R < Q$.

## Input File Format

The first line of the input file contains an integer indicating the number of test cases to follow. Then the input $(n, p_1, p_2, p_3, \cdots, p_n, Q, R)$ of each test case is given in a separated line. All integers are separated by one space.

## Output Format

For each test case, your program should output the probability of the event that $X \ mod \ Q$ is equal to $R$ in a separate line. The probability must be rounded to 5 digits after the decimal point.

## Sample Input

```
4
4 0 90 100 80 5 3
```

```
4 100 90 0 80 5 3
4 0 90 100 80 5 3
5 98 76 54 32 11 11 6
```

## Sample Output

```
0.08000
0.74000
0.08000
0.25224
```

# Problem D
## Register Allocation
Input File: *pd.in*
Time Limit: *3 seconds*

## Problem Description

A computer program stores the values of its variables in memory. For arithmetic computations, the values must be stored in easily accessed locations called *registers*. Registers are expensive such that we need to use them efficiently. If two variables are never used simultaneously, then we can allocate them to the same register. Suppose that there are $n$ variables used in a computer program. For convenience, we use $\mathsf{Var} = \{1, 2, \ldots, n\}$ to represent these $n$ variables. For each variable $i$, we know the *start time* $s_i$ and the *finish time* $f_i$ when it is used. A variable $i$ is *active* during the time interval $[s_i, f_i]$, where $s_i$ and $f_i$ are two positive integers with $s_i < f_i$. Two variables $i$ and $j$ can be stored in the same register if the corresponding time intervals are *disjoint*, that is, $[s_i, f_i] \cap [s_j, f_j] = \emptyset$. Note that even when $s_i < f_i = s_j < f_j$, the intervals $[s_i, f_i]$ and $[s_j, f_j]$ are not disjoint. Thus, such variables $i$ and $j$ cannot be placed in the same register. Given a set of $n$ variables and the corresponding start time and finish time, your task is to write a computer program to compute the minimum number of used registers.

For example, consider $\mathsf{Var} = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and the following table shows the start time and finish time for each variable. Note that variables $\{1, 3, 6\}$ can be stored in Register A; variables $\{2, 5, 7\}$ can be stored in Register B; and variables $\{4, 8\}$ can be stored in Register C. The minimum number of the required registers equals 3.

| Variable | $s_i$ | $f_i$ |
|---|---|---|
| 1 | 1 | 3 |
| 2 | 2 | 6 |
| 3 | 4 | 8 |
| 4 | 5 | 11 |
| 5 | 7 | 9 |
| 6 | 10 | 14 |
| 7 | 12 | 15 |
| 8 | 13 | 16 |

## Technical Specification

- $1 \leq n \leq 10000$.
- For each variable $i$, $1 \leq s_i \leq 10000$ and $2 \leq f_i \leq 30000$.

## Input File Format

The first line of the input file contains an integer, denoting the number of test cases to follow. For each test case, the set of registers $\mathsf{Var} = \{1, 2, \ldots, n\}$ is given with the following format: The first line of each test case contains a positive integer $n$. In the following $n$ lines, each line contains two positive integers separated by a single space. The first integer represents the start time and the second integer represents the finish time. The two positive integers

in the $i^{\text{th}}$ line of each test case represent $s_i$ and $f_i$ of variable $i$.

## Output Format

For each test case, output the minimum number of the required registers.

## Sample Input

```
2
8
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
6
1 2
2 3
3 4
4 5
5 6
6 7
```

## Sample Output

```
1
2
```

# Problem E
## Finding Bottleneck Shorstet Paths
Input File: *pe.in*
Time Limit: *1 second*

## Problem Description

A sensor network consists of a set of $n$ sensors $s_1, s_2, \ldots, s_n$. All the sensors are placed in a two dimensional plane and will never be moved again. Thus, each sensor $s_i$ has a fixed coordinates $(x_i, y_i)$.

A pair of sensors $s_i$ and $s_j$ can communicate by sending messages. Suppose that $s_i$ wants to send a message directly to $s_j$, a fixed amount of electrical power $p_{i,j}$ is required at $s_i$. In real world situation, the value of $p_{i,j}$ depends on many factors. For simplicity we assume that the value of $p_{i,j}$ depends only on the distance between the communicating sensors. In this problem, we assume that

$$p_{i,j} = (x_i - x_j)^2 + (y_i - y_j)^2.$$

Furthermore, we assume that only the sender needs to consume this amount of power in the communication.

Since the power stored in each sensor is a precious resource, sending message directly to the destination sensor may consume too much electrical power for a sensor. In this problem, we want to find an *optimal* path to send a message from $s_i$ to $s_j$ such that the maximum power required by the sensors on the path is minimized.

More formally, let $P$ be a *valid* path from $s_i$ to $s_j$. Let $k_1 = i, k_2, \ldots, k_r = j$ be the sequence of the indexes of the sensors along the path $P$. Define the weight of $P$ by

$$w(P) = \max_{1 \le i < r} \{p_{k_i, k_{i+1}}\}.$$

A path $P$ is an optimal path from $s_i$ to $s_j$ if its weight $w(P)$ is minimized among all paths from $s_i$ to $s_j$.

Given a sensor network $G$, the sender $s_i$ and the receiver $s_j$, write a program to compute an optimal path from $s_i$ to $s_j$ for sending a message.

## Input Format

An instance of the problem consists of

1. the number of sensors $n$,
2. the coordinates of the sensors $(x_i, y_i)$, $1 \le i \le n$, and
3. the source and the destination sensors $s_i$ and $t_i$.

These data are stored in $\lceil n/20 \rceil + 2$ lines in the input file.

1. The first line is the integer $n$.
2. The following $\lceil n/20 \rceil$ lines are the $n$ coordinates $(x_i, y_i)$, $1 \le i \le n$. Each line contains at most 20 coordinates. Each coordinates is written in two numbers $x_i$ and $y_i$, without the parentheses.

3. The last line of an instance contains two integer $i$ and $j$, indicating $s_i$ is the sender and $s_j$ is the receiver.

In this problem, we assume that $1 < n < 1000$, $x_i$ and $y_i$ are integers and $0 \leq x_i, y_i < 2^{15}$.

Note that the test data file may contain more than one instances. The last instance is followed by a line containing a single 0.

## Output Format

The output for each test case is an integer $w$ which is the maximum power required along the optimal path from $s_i$ to $s_j$.

## Sample Input

```
4
0 0  1 9  8 2  10 10
1 4
5
0 0  8 2  3 4  8 7  10 10
1 5
0
```

## Sample Output

```
68
29
```

# Problem F
## Robot Arm Planning
Input File: *pf.in*
Time Limit: *1 second*

## Problem Description

Your job is to design a robot arm as in Figure 1. However, the control of a robot arm can be complicated, so you want to experiment it on a 2D simplified version (as in Fig. 2) before a real one is built.

Figure 1: An robot arm.

A robot can have several joins (or "torques" in their formal name) and each join is controlled by a motor. The motor can only rotate exactly 45 degree clockwise or counter clockwise in one move. For example, to rotate 90 degree, you need two moves. To rotate a join in one move, the motor consumes a fixed amount of power. The goal of robot arm planning is to move the arm to a destination with minimum number of moves of all motors. In Fig. 2(a), a robot arm with 3 arms and 4 joins is shown. Let Join 0 be always fixed at coordinate (0,0). Each arm is 100 cm long and a join has radius of 10 cm. Given a rectangular area defined by $(x_1, y_1, x_2, y_2)$, where $(x_1, y_1)$ is the coordinate of bottom-left corner and $(x_2, y_2)$ is the coordinate of top-right corner, and given a robot arm with $N$ arms and $N + 1$ joins, please compute the minimum moves needed to move join $N$ (the top join) inside that area. For example, in Fig. 2(b), to move the join 3 inside the rectangular area $(100, 200, 200, 300)$ can be simply done by rotating join 1 by one clockwise move. Note that, the body of the top join must be completely enclosed by the rectangular area. In addition, please assume the arms will not block each other. For example, it is valid to have join 1 rotate 180 degree to overlap arm 0 and arm 1.

## Technical Specification

1. $2 \leq N \leq 10$.
2. $-N \times 100 \leq x_1, y_1, x_2, y_2 \leq N \times 100$, where $x_i$'s and $y_j$'s are integers.

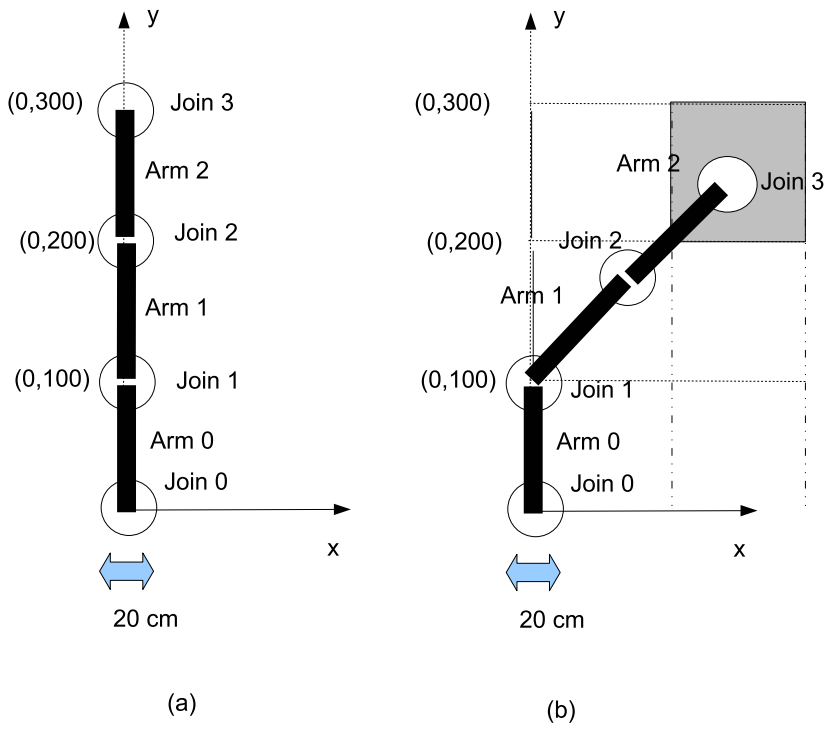



Figure 2: A 2D simplified robot arm.

## Input Format

The test data begins with a positive integer $M$, where $M$ is the number of test cases. Each test case begins with a positive integer $N$, which is the number of arms. The last part of a test case is the rectangle area specified by $x_1\ y_1\ x_2\ y_2$ where $(x_1, y_1)$ is the coordinate of bottom left corner and $(x_2, y_2)$ is the coordinate of top-right corner.

## Output Format

Please output the minimum number of moves for each test case. If a test case does not have feasibile solution, please output "-1".

## Sample Input

```
3
1
100 200 200 300
3
50 150 150 250
4
150 -50 250 50
```

## Sample Output

```
-1
2
3
```

# Problem G
## Finding Feasible Paths
Input File: *pg.in*
Time Limit: *3 seconds*

## Problem Description

You are all excellent programmers and should be good at tracing program running behaviors for debugging purposes. The following problem is to request you to verify if the program runs in the feasible ways.

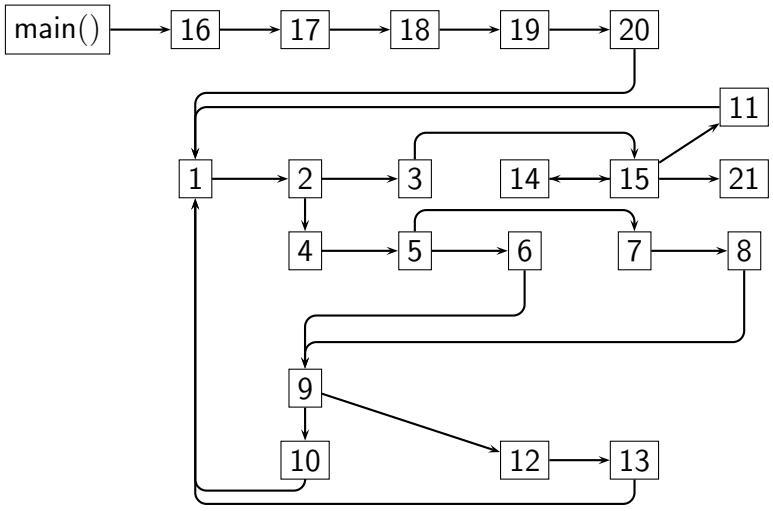
A program execution with function invocation can be modeled as a control flow graph. The control flow can be identified as an intra-function flow or inter-function flow. The inter-function flows will occur at function invocation and function return. For example, if we trace the following function simpleRecFunc(x,y,z) written in the syntax of the C programming language:

```
1: void simpleRecFunc(int a,int b, int *v) {
2:    if (a <=2) {
3:          *v = b + 1;
4:    } else {
5:        if (a>b)
6:          *v = *v+1 ;
7:        else
8:          *v = *v -1 ;
9:        if (a < b + 1) {
10:               simpleRecFunc(a-1, b, v);
11:               simpleRecFunc(a-2,*v,v);
12:       } else
13:               simpleRecFunc(a-3,*v,v);
14:    }
15: }
16: void main() { srandom(getpid());//set a random seed
17:   int x =random()%21;//obtain the x value between 0 and 20
18:   int y =random()%101;//obtain the y value between 0 and 100
19:   int z;
20:   simpleRecFunc(x,y,&z);
21: }
```

Each statement is annotated with a line number n, where n is a positive integer. After a function invocation, the return line number is n+1. An exception is that after a function invocation at line 11, the return line number is 14. The formal parameter v in the simpleRecFunc() function is always passed in a valid integer reference, containing an integer value.

For example, if at line 2, the condition ($a \leq 2$) is true, the run-time flow path is 2 3 15; if the above condition is false, and at line 5, the condition ($a > b$) is true, the run-time path is 2 4 5 6 9 12 13. We won't trace the control flow and step into the standard libary function of srandom(), getpid(), and random(), but only trace the flow into the function of simpleRecFunc(). According to the above program, we can depict a corresponding flow graph as follows:

Each node in the flow graph is labeled with the corresponding statement line number. The run-time path will be varied, due to different parameters of a, b, and v.




Given a path from node 16 to node 21, you are to determine if the path is feasible. For example, the path 16 17 18 19 20 1 2 3 15 21 is feasible. The path 16 17 18 19 20 1 2 4 5 7 8 9 12 13 1 2 3 15 21 is infeasible for we cannot find any possible integers or zero for a,b and *v to execute the above path. The path 16 17 18 19 20 1 2 4 5 7 8 9 10 1 2 3 15 21 is also infeasible because after the function invocation, the control must return to the next statement of the caller. The caller here being at line 10, the next statement line number is 11. Please note that there is no direct edge between 10 and 11 due to that we step into the function of simpleRecFunc() and the next node of 10 is 1 instead of 11. Assume the program is given enough memory to execute.

## Technical Specification

1. The number of path data is N , where $0 < N \leq 1024$.
2. The number of statement line numbers in an path is M, where $0 < M \leq 2^{16}$.
3. In the invocation of simpleRecFunc(x,y,z), $0 \leq x \leq 20$ and $0 \leq y \leq 100$

## Input File Format

The test data begins with a positive integer N, where N is the number of path data. The subsequence N lines are the input paths, specified by a sequence of statement line numbers, each separated with one or more spaces, from the starting statement line 16 to the end statement line 21. Each input path is read up to the end of the line, for example:

16 17 18 19 20 1 2 3 15 21

## Output Format

If the given path is feasible according to the above control flow graph, output feasible, otherwise output infeasible.

## Sample Input

```
4
16 17 18 19 20 1 2 3 15 21
16 17 18 19 20 1 2 4 5 7 8 9 12 13 1 2 3 15 21
```

```
16 17 18 19 20 1 2 4 5 7 8 9 10 1 2 3 15 21
16 17 18 19 20 1 2 4 5 6 9 12 13 1 2 3 15 14 15 21
```

## Sample Output

```
feasible
infeasible
infeasible
feasible
```

# Problem H
## KGold
Input File: *ph.in*
Time Limit: *5 seconds*

## Problem Description

Fortune magazine has an App that help determine the richest people in the world at any given time $T$. At the beginning of the year, the N richest people have their net worth estimated in KGold. Depending on their personal investment portfolio, their wealth increases at a constant rate in KGold/sec. Therefore, at time $T$, the wealth ranking may have changed.

Lately, it was found that there are some bugs in the App. For large time $T$, the ranking could be wrong. You are hired to enhance the App by adding a debug mode function. When in debug mode, the App is to output how the wealth ranking changes from time to time. First, whenever a person catches up (means equal) with and consequently overtakes another person in terms of overall wealth (KGold), a smiley face icon is flashed on the display screen. So, please count and display the number of times the smiley face is flashed by time $T$. Furthermore, information on who overtook whom should also be displayed, in order of the time of the overtake.

## Technical Specification

1. $1 <= N <= 250,000$.
2. Initial wealth of each person is between 0 and 1,000,000 KGold, inclusive.
3. The speed in which each person's wealth is incrased is between 0 and 100 KGold per second, inclusive.
4. $1,000,000 <= T <= 2,000,000$.

## Input File Format

The first line of the input specifies the number of test cases to follow. For each test case, the first line has two integers: $N$, the number of richest people at the beginning of the year; $T$, the time in the future in which the wealth of the $N$ people is peeked. The next $N$ lines each contains two integers $G_i$ and $S_i$ where $G_i$ is the wealth of person $i$ at the beginning of the year and $S_i$ is the speed in which person $i$'s wealth increases in KGold per second. Those $N$ lines are ordered in increasing order of $G_i$. In other words, no two people's beginning wealth will be the same and $G_1 < G_2 < ... < G_N$.

## Output Format

For each test case, first output a single integer $F$ on a line indicating the number of times the Smiley face is flashed on the display screen modulo 1,000. Next, display $F$ (or first 10,000 if $F > 10,000$) lines of output. Each line contains two integers $i$ and $j$, indicating that person $i$ overtook person $j$ in terms of overall wealth at some point on or before time $T$. The list must be in chronological order. In other words, the passing that took place first must be displayed before the passing that took place later in time. Furthermore, if two or more passings took place at the exact same time, the one with less wealth (among the two

people passing the other two perople) should be displayed first. For all test cases, no more than two people will have the same exact wealth (in KGold) at any given time.

## Sample Input

```
1
4 1000000
0 2
2 1
3 8
6 3
```

## Sample Output

```
2
3 4
1 2
```

# Problem I
## Airport
Input File: *pi.in*
Time Limit: *6 seconds*

## Problem Description

Wonder City is a beautiful place and has attracted a considerable amount of tourists in these years. The mayor is planning to build a new airport to accommodate tourists. The new airport will provide free shuttle buses to all hotels in Wonder City. There are several tourist centers. In order to further introduce the beauty of this city to tourists, each shuttle bus will first stop by a tourist center before arriving at its destination hotel.

With the existence of hotels and tourist centers, the mayor wants to find the best location for the new airport. In what follows, let $G = (V, E)$ be an undirected connected graph representing the map of Wonder City, where $V$ is the vertex set and $E$ is the edge set. In this problem, we consider each edge as a line segment in the Euclidean plane so that we can talk about points, not necessarily vertices, on the edges. We also use $G$ to denote the set of all points of the graph. Thus, the notation $x \in G$ means that $x$ is a point along any edge of $G$ which may or may not be a vertex of $G$. Each edge $e \in E$ has a nonnegative integral length. For any two points $a, b \in G$, let $d(a, b)$ be the distance of the shortest path between $a$ and $b$. For example, in Figure 3, $d(p, h_2) = 20.75$ and $d(c_a, c_b) = 29$. Let $C \subseteq V$ denote the set of tourist centers and let $H \subseteq V$ denote the set of hotels. The mayor's strategy for choosing the location of the airport is as follows.

(1) Any point of $G$ can be selected as the new airport $p$.

(2) The shuttle bus to each hotel $h \in H$ will stop by the tourist center $c \in C$ that minimizes the distance $d(p, c) + d(c, h)$, so that tourists can arrive at their hotels as soon as possible.

(3) For each $h \in H$, let $s(p, h) = \min\{d(p, c) + d(c, h) | c \in C\}$ and $t(h)$ be the expected number of tourists from the airport to the hotel $h$ every day. The new airport should minimize the *largest unsatisfactory factor* $f(p) = \max\{t(h) \times s(p, h) | h \in H\}$. (In this scenario a hotel $h$ is considered more important than another hotel $h'$ if $t(h) > t(h')$.)

Consider the example in Figure 3. Suppose the airport $p$ is built at location $h_3$. Then, $s(p, h_1) = \min\{d(p, c_a) + d(c_a, h_1), d(p, c_b) + d(c_b, h_1)\} = \min\{34, 52\} = 34$. Similarly, we have $s(p, h_2) = 28$ and $s(p, h_3) = 26$. Therefore, the largest unsatisfactory factor is $f(p) = \max\{30 \times 34, 50 \times 28, 20 \times 26\} = 1400$. In this example, the best location of $p$ is on the edge $(h_3, c_b)$, which is 4.75 units of distance away from $h_3$ and 8.25 units of distance away from $c_b$. The largest unsatisfactory factor of this location is $f(p) = 1162.5$.

Please help the mayor of Wonder City to determine the best location of the new airport. For simplicity, only the largest unsatisfactory factor of the new airport is required.

## Technical Specification

1. The number of hotels, $n = |H| : 2 \leq n \leq 200$.
2. The number of tourist centers, $k = |C| : 2 \leq k \leq 30$.
3. The number of edges, $m = |E| : 3 \leq m \leq 8000$.

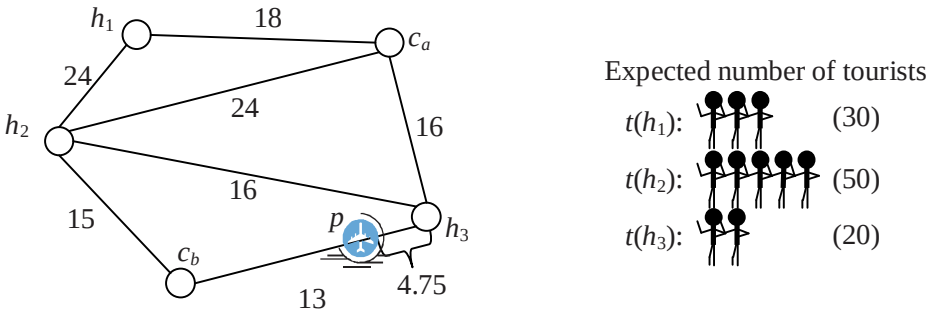



Figure 3: An example, in which $H = \{h_1, h_2, h_3\}$ and $C = \{c_a, c_b\}$.

4. The expected number of tourists for each hotel $h$, $t(h) : 1 \leq t(h) \leq 100$.

5. $C \bigcap H = \emptyset$ and the vertex set is $V = C \bigcup H$.

6. There is at most one undirected edge between any pair of vertices.

## Input File Format

There are at most 10 test cases. The first line of each test case consists of the three integers $n$, $k$, and $m$. Denote the $n$ hotels by $v_1, v_2, ..., v_n$; and denote the $k$ tourist centers by $v_{n+1}, v_{n+2}, ..., v_{n+k}$. Next, $m$ lines follow. Each line contains three integers $i$, $j$, $l$, where $1 \leq i \neq j \leq n + k$ and $0 \leq l \leq 1000000$, indicating that there is an edge of length $l$ connecting $v_i$ and $v_j$. Finally, there is a line containing the $n$ integers $t(v_1)$, $t(v_2)$, ..., $t(v_n)$. The last test case will be followed by a line consisting of $n = k = m = 0$.

## Output Format

For each test case, print a line containing the largest unsatisfactory factor of the new airport, rounded to three decimal places. Use the format of the sample output.

## Sample Input

```
3 2 7
1 2 24
2 4 24
5 2 15
5 3 13
4 3 16
4 1 18
2 3 16
30 50 20
3 2 7
1 2 10
2 4 20
5 2 15
5 3 15
4 3 15
4 1 20
2 3 20
```

```
30 50 20
0 0 0
```

## Sample Output

```
1162.500
750.000
```