IDI Open Programming Contest April 17th, 2010

Problem Set

A Guarding the Border
B Beehive Epidemic
C Mobile Gaming
D Balancing Weights (Easy)
E Ambulance Antics
F Nurikabe
G Cookie Monster (Easy)
H Typing Monkey
I The Diligent Cryptographer
J Combat Odds

Jury and Problem Writers

Eirik Reksten, Steria
Ruben Spaans, IDI/NTNU
Tor Gunnar Høst Houeland, IDI/NTNU
Rune Fevang, Opera Software

Tips

- Tear the problem set apart and share the problems among you.
- Problems are not ordered by difficulty.
- Try solving the easy problems first. Two problems in this set are tagged with "(Easy)" to help point you in the right direction.
- If your solution fails on a problem, you can print your program and debug it on paper while you let someone else work on a different problem on the computer.
- If you need help, contact the judges.

Rules

- Each team consists of one to three contestants.
- One computer is used per team.
- You may not cooperate with persons not on your team.
- You may print your programs on paper to debug them.
- What you may bring to the contest:
  - Any written material (Books, manuals, handwritten notes, printed notes, etc).
  - Pens, pencils, blank paper, stapler and other useful non-electronic office equipment.
  - NO material in electronic form (CDs, USB pen and so on).
  - NO electronic devices (PDAs and so on).
- The only electronic content you may consult during the content is that specified by the organiser (see the web-page). You may not copy source code from web pages, etc.
- Your programs should read from standard in and write to standard out. Writing to standard error will result in a failed submission. C programs should return 0 from `main()`.
- Your program may use at most 100MB of memory.
- Your programs may not:
  - access the network,
  - read or write files on the system,
  - talk to other processes,
  - fork,
  - or similar stuff.
  - If you try, your program will hang or crash. If it hangs, it will take a couple of minutes before others will be able to run their programs. And please do not crack somebody who uses their spare time trying to give you something valuable.
- Show common sense and good sportsmanship.

Problem A

Guarding the Border

As newly appointed Chief of Security, you have decided that it's time to upgrade the border defense. There have been rumors of some neighboring countries developing nuclear weapons, so a few extra archer towers will be needed. In order to spot the miscreants when they approach, you would like to minimize the maximal distance between adjacent towers.

The border is modeled as a cyclic straight line from 0 to L, with N (old) towers placed along it. The country is an inland country, so the first and last towers are neighboring as well (consider points 0 and L to be the same). You have enough finances to place up to M new towers along the border. Find the lowest possible maximal distance between adjacent towers after the placement.

Input specifications

The first line of input contains a single number T, the number of test cases. Then follow T lines, each describing a test case. Each test case starts with three integers N, M and L. N is the amount of towers already present at the border, M is the maximum amount of new towers you are allowed to place and L is the length of the border. Then follow N floating-point numbers t_i describing the locations of the current towers.

Output specifications

For each test case, output one line containing a single number, the lowest possible maximal distance between adjacent towers after the placement.

Notes and Constraints

- $0 < T \leq 100$
- $0 \leq N \leq 20000$
- $0 < M \leq 20000$
- $0 < L \leq 10000000$
- $0 \leq t_i < L$
- No two towers are located at the exact same location.
- An absolute or relative error of up to 10^{-7} compared to the correct answer will be accepted for the distances.

Sample input

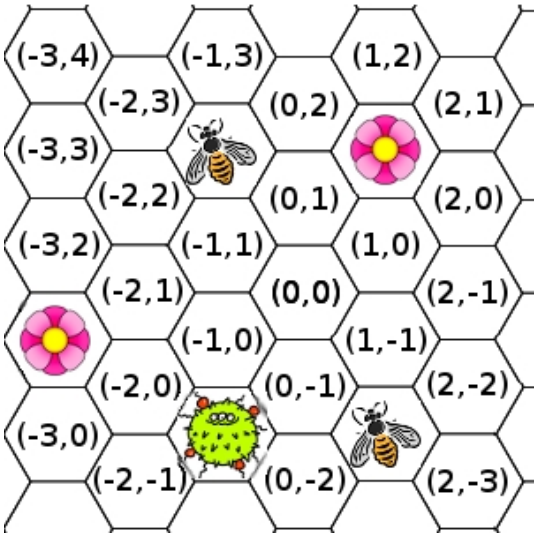
```
2
0 3 15
2 1 1000 667.4 333.8
```

Output for sample input

```
5
333.6
```

Problem B

Beehive Epidemic



In the last couple of months the wretched bee flu has been on the rampage. To battle extinction, the bees of Inner Dreaded Illnessia have set up a system of safe zones in their hives. Sadly, they can't create such zones everywhere (they need somewhere to place the honey as well), so there is always chaos when the bacteria appear in the hive. You need to help the bees, and figure out how many of them can be saved if they organize themselves optimally.

The beehive is a hexagonal grid (see illustration), and a bee can move from the zone where it's located to any neighboring one once every second *or it can choose to remain in the same cell (clarification)*. At the same time, the bacteria will spread from every cell containing it to all its six neighbors once every two seconds. Safe zones stay in the same location (of course). If a bee reaches an empty safe zone before the bacteria does, it can withstand infection by staying there until the danger passes. Due to the functioning of these things, only one bee can stay safe at each of these zones at the same time.

Input specifications

The first line of input contains a single number T, the number of test cases to follow. Each test case consists of four lines. The first line contains the numbers N, S and B, the number of bees, safe zones and bacteria, respectively. Then follow three lines describing the initial locations of each of these. The first line describes the locations of the N bees, the second the locations of the S safe zones, while the third describes the locations of the B bacteria. Each line is formatted $x_1y_1x_2y_2...x_ky_k$, where k represents the amount of locations (N, S or B).

Output specifications

For each test case, output one line containing a single number, the maximum amount of bees that can survive the epidemic.

Notes and Constraints

- $0 < T \leq 150$
- $0 < N, S \leq 50$
- $0 < B \leq 1500$
- $-100 \leq x_i, y_i \leq 100$
- The bees make up to two moves before the bacteria spread for the first time (and then two more moves before the spread continues, and so on).
- Though only one bee can stay safe at a single safe zone, there is no problem for bees to stay in the same zone (even safe zones).
- The earliest time a bee can be safe at a safe zone is after their first move. That is, they are not hiding "just in case".
- The beehive is very large. For the purposes of this problem, assume that the bees won't be able to make it outside (without growing tired and having to take a break until the bacteria catches up).

Sample input

```
1
2 2 1
-1 2 1 -2
-3 1 1 1
-1 -1
```

Output for sample input

```
2
```

Problem C

Mobile Gaming

Online gaming using mobile phones has exploded in popularity, and you've thrown yourself on the wave, now creating a kind of cop-and-robber type of game. Some players control the cops, while others control the robbers. The objective for the cop players is to catch the robbers, while the robbers will try to escape. During development, you've recently encountered a problem. There is a significant lag in the update rate on the mobile phones. You might get updates on player positions as seldom as once every second. Since scoring is based on when the cops caught the robbers, you need to calculate this exact time.

Cops and robbers are modeled as rectangles in the game world, and a cop has caught a robber if their rectangles overlap. You are given the sizes of the rectangles, as well as their positions in time 0 and 1. Your program should calculate whether the rectangles have overlapped at some point in this time interval and, if so, at what time this happened *for the first time (clarification)*.

Input specifications

The first line of input contains a single number T, the number of test cases to follow. Each test case consists of two lines, containing 6 numbers each. The first line contains the integers W_1, H_1, $Xstart_1$, $Ystart_1$, $Xend_1$ and $Yend_1$. W and H describe the width and height of the policeman rectangle, respectively. $Xstart_1$ and $Ystart_1$ describe the policeman's starting point (the upper left corner), while $Xend_1$ and $Yend_1$ describe his end point (his location at time 1). The second line contains six integers W_2, H_2, $Xstart_2$, $Ystart_2$, $Xend_2$ and $Yend_2$, the corresponding numbers for the robber.

Output specifications

For each test case, output one line containing a single number, the *earliest (clarification)* instant in time (a real number between 0 and 1, inclusive) in which a collision occurs. If there is no collision, output `No Collision` instead.

Notes and Constraints

- $0 < T \leq 100$
- $0 < W_i, H_i \leq 100$
- $0 \leq Xstart_i, Ystart_i, Xend_i, Yend_i \leq 10000$
- The coordinate system has increasing x-values from left to right, and increasing y-values from top to bottom.
- A collision occurs if the two rectangles overlap or if their sides/corners touch.
- Assume that both the cop and robber moved in a straight line.
- An absolute or relative error of up to 10^{-7} compared to the correct answer will be accepted.

Sample input

```
3
2 2 0 0 0 8
2 2 2 10 2 2
5 5 10 10 0 0
1 4 0 0 10 0
4 4 3 9 10 18
3 3 8 14 15 23
```

Output for sample input

```
0.5
0.6
No Collision
```

Problem D

Balancing Weights (Easy)

Ever since you started studying, your whole family have been expecting you to know the answers to a whole lot of difficult questions. What is wrong with my computer? What is the name of Prince Harry's new girlfriend? Have you seen my new pants? Your grandfather has just found a new problem for you, and you are yet again under the pressure of finding the answers to one of life's most fundamental questions.

You are given a 20 meter long lever balanced exactly on the middle by a massless support. A number of weights are applied to the lever. You need to figure out which side will drop, if any. Being such a brilliant mind, you immediately notice that each of the weights will contribute to the total torque applied on the lever, and that this will determine the answer. The torque from a single weight is determined by

$$\tau = m \times d \tag{1}$$

where τ is the total torque applied, m is the mass of the weight and d is its distance from the center. The lever's angular acceleration can then be found by the equation

$$\alpha = \tau / I \tag{2}$$

where α is the angular acceleration and I is the lever's moment of inertia. The moment of inertia is given by the function

$$I = \int r^2 dm \tag{3}$$

where r is the perpendicular distance to the axis of rotation.

Input specifications

The first line of input contains a single number T, the number of test cases to follow. Each test case starts with a line containing N, the number of weights in the test case. This is followed by a line containing N numbers, W_1 W_2 ... W_N the locations of the N weights.

Output specifications

For each test case, output one line containing `Left` if the weight tips to the left, `Right` if the weight tips to the right or `Equilibrium` if the weight does not tip to any of the sides.

Notes and Constraints

- $0 < T \leq 100$
- $0 < N \leq 100$
- $-1000 \leq W_i \leq 1000$
- A negative W_i means that the weight is located to the left of the center, while a positive one means that it is located to the right.
- The weight of the lever is exactly 2000 grams, uniformly distributed. Each weight weighs 100 grams.
- Weights are modeled as single point masses.

Sample input

```
3
3
-2 0 2
1
4
4
4 -2 0 -3
```

Output for sample input

```
Equilibrium
Right
Left
```

Problem E

Ambulance Antics

Your friend, the madman Tommy Vercetti, is in trouble. He has hijacked an ambulance, and has to transport patients that are scattered around Vice City, to the local hospital. He wants to accomplish this task as soon as possible, so he can carry on doing other missions (mostly ones involving brutal violence). Therefore, you must write a computer program which calculates the shortest time it is possible to accomplish his task.

The ambulance has room for a maximum of three patients. The ambulance needs to return to the hospital to drop off patients. Vice City consists of streets and intersections. One intersection is the location of the hospital, and each of the other intersections contain a patient to be picked up. Each street is bidirectional, and has a cost associated with it, which is the number of minutes it takes to drive from one end to the other. Loading and unloading the ambulance is done instantly. At the beginning of each scenario, the ambulance is at the hospital.

Input specifications

The first line of input contains a single number T, the number of test cases that follow. The first line of each test case contains two integers N and M, the number of intersections containing patients, and the number of bidirectional streets. The following M lines contain three integers a_i, b_i, c_i. Each of these lines represent a bidirectional street between intersections a_i and b_i, and c_i is the number of minutes needed to drive from a_i to b_i (or the opposite direction). Vice City has $N + 1$ intersections, numbered from 0 to N (inclusive). The hospital is located at intersection N, and there are patients located at intersections $0, 1, \ldots, N - 1$.

Output specifications

For each test case, output the minimal number of minutes it takes to deliver all the patients to the hospital.

Notes and Constraints

- $0 < T \leq 100$
- $1 \leq N \leq 20$
- $M > 0$
- $0 \leq a_i, b_i \leq N$
- $0 < c_i \leq 100000$
- There always exists a path between any pair of intersections.
- There are never two or more streets between two intersections.

Sample input

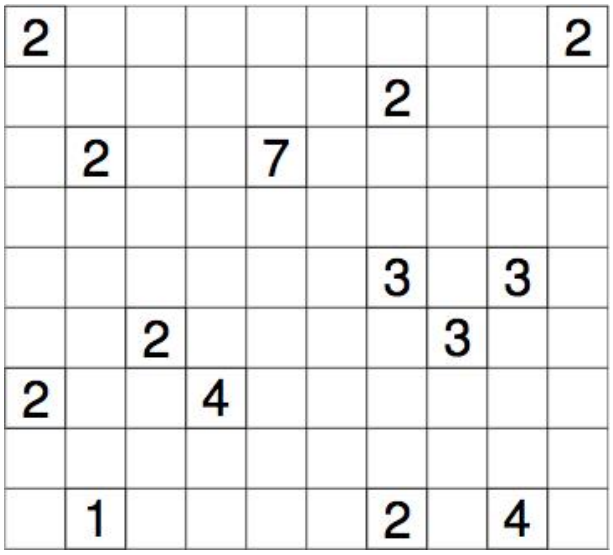
```
1
2 2
0 1 10
1 2 10
```

Output for sample input

```
40
```

Problem F

Nurikabe



Nurikabe is a binary determination puzzle originating from Japan. Given a grid where some cells contain numbers, the objective of the puzzle is to mark each blank cell as either island (white) or water (black), while obeying the following constraints:

- Each island has exactly one numbered cell, containing a number between 1 and 9. The number of white cells (including the numbered cell) in this island is equal to this number. Two cells are connected if they share a side. Two cells belong to the same island if there exists a path going through connected island cells.
- All water cells (black) are connected. Water cells are connected in the same manner as island cells.
- Within a 2×2 block there must be at least one cell belonging to an island.

In this problem, you are asked to verify that Nurikabe puzzles are solved correctly.

Input specifications

The first line of input contains a single number T, the number of test cases that follow. The first line of each test case contains integers N and M, the size of a puzzle in rows and columns. The next N lines contain the rows of the puzzle. Each line contains characters from the set `123456789.#` where `.` and any digit represent an island cell and `#` represents a water cell.

Output specifications

For each test case, output `YES` if the board is filled in correctly according to the rules, and `NO` otherwise.

Notes and Constraints

- $0 < T \leq 100$
- $1 \leq N, M \leq 50$
- Recent surveys indicate that more than seven billion chocolate chip cookies are eaten annually.

Sample input

```
2
9 10
2.#...##.2
###..#2###
#2#.7#.#.#
#.######.#
##.#..3#3#
.#2####3##
2##4.#..#.
##..#####.
#1###.2#4.
2 2
#1
1#
```

Output for sample input

```
YES
NO
```

Problem G

Cookie Monster (Easy)

Everyone needs hobbies, and Christian is no exception. He is out of this world fond of cookies. Where there are cookies, you'll find Christian. Sadly, he's not too good at keeping track of his supplies. He needs to eat at least one cookie every day, and as long as there are enough left, he will eat C of them. Given that he has N cookies left, for how many days will he eat at least one cookie?

Input specifications

The first line of input contains a single number T, the number of test cases to follow. Each test case consists of one line containing two numbers, N and C, the total number of cookies he has and the number he eats every day.

Output specifications

For each test case, output one line containing a single number, the amount of days where Christian still has cookies to eat.

Notes and Constraints

- $0 < T \leq 100$
- $0 < N \leq 1000000000$
- $0 < C \leq 5000$

Sample input

```
2
6 2
10 3
```

Output for sample input

```
3
4
```

Problem H

Typing monkey

You have gotten hold of a monkey which is able to use a typewriter, and it is even capable of typing letters from a specific probability distribution.

You believe that this monkey will eventually be able to type Shakespeare's complete works. Your friend Eirik, however, believes that the monkey is more capable of producing another novel in the Harry Potter saga.

To settle this uncertainty, you must write a program which calculates the probability of producing a certain piece of literature before another piece, here represented by single words. A word is produced if is occurs as a substring somewhere in the stream of letters typed by the monkey. The monkey is only able to type lowercase letters from the English alphabet.

Input specifications

The first line of input contains a single number T, the number of test cases that follow. Each test case consists of two lines. The first line contains 26 floating point values $p_a, p_b, \ldots, p_z$, the probabilities of the monkey typing each letter, each separated by one space. The second line contains two strings P and Q, each separated by one space. Both strings consist of lower case letters from the English alphabet. The two strings will never be equal, and the probability of the monkey being able to produce the strings is always greater than zero. There are no test cases where the monkey can produce both strings at the same time.

Output specifications

For each test case, output the probability of the monkey producing the word P before the word Q.

Notes and Constraints

- $0 < T \leq 100$
- $0 \leq p_\alpha \leq 1$
- $\sum p_\alpha = 1$
- $0 < |P|, |Q| \leq 16$
- An absolute or relative error of up to 10^{-7} compared to the correct answer will be accepted.
- The monkey's keypresses are independent events.

Sample input

```
1
0.1 0 0 0 0.1 0 0 0.1 0 0 0 0.1 0.1 0 0.1 0.1 0 0.1 0 0.2 0 0 0 0 0 0
hamlet potter
```

Output for sample input

```
0.33333333333333
```

Problem I

The Diligent Cryptographer

Halvor is in charge of the Single Sign-On (SSO) login system for Identity Directories, Inc. He has been a passionate supporter of their technology for years, telling anyone who will listen how it makes user authentication simpler and more secure with the encrypted login back-end provided by Trustworthy Enterprises (TE). Last week Halvor got a newsletter from TE, where they introduced their new and highly innovative Open Trust Protection (OTP) system, which was recently implemented and has been used for new accounts and users that changed their password in the last month.

Previously, a user's cryptographic key consisted of a permutation of the first letters of the alphabet, repeated many times so it could be used for long messages. In the new improved system the cryptographic key instead consists of random letters generated by a lava lamp-based sub-contractor.

As an example the string `BCAEDBCAEDBCAED` was a possible key in the old system since this is a repetition of `BCAED`, which is a permutation of the letters from `A` to `E`. The strings `BCDBCD` and `BABBABBABBAB` would not be possible, since the letter `A` is missing from the repeated permutation `BCD`, and `BAB` is not a permutation of `AB` since there are two `B`'s.

Halvor decides to change the keys for the users that have not already been automatically moved to the new system. Luckily he has read and write access to all the keys for his users, and has contracted you to write a program to determine which users need to be updated. To avoid any privacy concerns, you are only given a list of the user's names, last login times and up to the first 1000 letters of their key.

Thus for the old system the end of the key substring you receive might be cut off in the middle of a repetition, but the first letter is guaranteed to be the start of a permutation. For the new system the entire string will be random, including the letters you receive.

Input specifications

The first line of input contains a single number T, the number of test cases to follow. Each test case consists of one line containing a string K, which is the first part of a user's cryptographic key.

Output specifications

For each test case, output a line containing the line "old" if K is definitely from the old system, "new" if K is definitely from the new system, or "unknown" if this cannot be determined from the provided key substring.

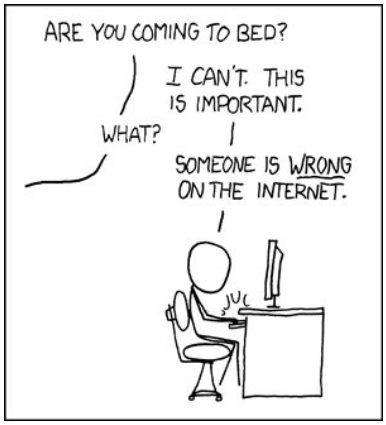
Notes and Constraints

- $1 \leq T \leq 1000$
- $1 \leq |K| \leq 1000$
- *All letters in the input string are uppercase (clarification)*
- The entropy can be written as $H(X) = -\sum_{i=1}^{n} p(x_i) \log_b(x_i)$, where p denotes the probability mass function of X.

Sample input	Output for sample input
4	
ABCD	unknown
BB	new
HELP	unknown
IAMTRAPPEDINACRYPTOGRAPHICKEYFACTORY	new

Problem J

Combat Odds



On internet forums for games, you'll always have a bunch of guys complaining about how they're losing. It's always something wrong with the game. Lack of realism and cheating AI is among the top choices. Sadly, it's mostly just lacking skills. To the rest of us, this becomes an everlasting quest to show them how wrong they are!

The latest in the never ending series is a guy claiming that the computer cheats in his newest game. Even though a 70% probability of winning is reported for a battle, he still loses! He is quickly shot down by our army of know-it-alls, but another guy rushes to his support. Even though you can lose a single battle at these odds, he has observed 5 such losses in a row! The probability of that is $(1 - 0.7)^5 = 0.00243$. Too low to be possible, he claims! Being such an honorable know-it-all, you explain how that might be true if those 5 battles were the only events overall, but throughout a whole game there are a lot more battles than that. The probability of observing such a streak is much larger! To back up your statement, you decide to calculate such probabilities.

A battle has two outcomes, win or lose. The probability of winning is given by p. Given that you simulate N battles, what is the probability of witnessing a losing streak of at least L battles?

Input specifications

The first line of input contains a single number T, the number of test cases to follow. Each test case consists of one line containing the three numbers N, L and p, separated by whitespace.

Output specifications

For each test case, output a line containing a single floating-point number, the probability of witnessing a losing streak of at least L battles.

Notes and Constraints

- $0 < T \leq 200$
- $0 \leq p \leq 1.0$
- $0 < N \leq 2000$
- $0 < L \leq N$
- The game in question is Civilization IV. You can assume that this game has a perfect random number generator, and that all battle outcomes are independent.
- An absolute or relative error of up to 10^{-7} compared to the correct answer will be accepted.

Sample input

```
2
5 5 0.7
10 5 0.7
```

Output for sample input

```
0.00243
0.010935
```