The "Summer on the Beach" Programming Contest

USC Programming Contest , Fall 2005

09/24/2005

Sponsored by Electronic Arts and VSoE

Hopefully, all of you have returned (or arrived) to USC after a nice and relaxing summer break. As you are well aware, Southern California is known for its nice beaches: Malibu, Santa Monica, Venice, Redondo Beach, etc. Hopefully, some of you took advantage of their abundance. And hopefully, you did not think about computers and computer science too much while out in the sun.

Although, come to think of it, there are surprisingly many computational problems one encounters at the beach. Sandcastle planning, volleyball scorekeeping, etc., could all be simplified with some nice computer programs. In order to make your next trip to the beach that much more enjoyable, we are going to solve some of these problems right now.

When you submit a problem, you submit your source code file using PC^2. Make sure to follow the naming conventions for your source code, and the input file you read. Remember that all input must be read from the file of the given name, and all output written to `stdout`. You may not use any electronic devices besides the lab computer, and may not open any web browser for whatever reason.

One thing you should know about this contest is that the judging is done essentially by a `diff` of the files. That means that it is really important that your output follow the format we describe — if you have the wrong number of spaces or such, an otherwise correct solution may be judged incorrect. Consider yourselves warned!

Another warning that seems to be in place according to our experience: all of our numbering (input data sets, classes, etc.) always starts at 1, not at 0.

And a piece of advice: several problems ask you to print numbers rounded to two decimals. In C, you do this via `printf ("%.2f", r);` where `r` is the variable you want to print. And in Java, the syntax is `System.out.print ((new java.text.DecimalFormat("#.##")).format(r));`

[Including this page, this problem set should contain 9 pages. If yours doesn't, please contact one of the helpers immediately.]

Problem A: Volleyball Scores

File Name: `ball.cpp|ball.java`

Input File: `ball.in`

Description

Beach Volleyball is a very popular game to play at the beach.[1] You form teams of two or more players on each side, and try to hit a ball back and forth. In reality, you usually just hope that whatever random person is hit by the ball is shorter than you, or won't get too angry.

When you play Volleyball somewhat more seriously, you try to keep track of a *score*. You get points if the ball bounces in your opponents' court, or they hit it out of bounds, or touch it too often. Scorekeeping is a tedious chore — there is really no reason to do it by hand. So we will write a program which will figure out the score given the sequence of what happened.

Here's a refresher of volleyball rules. Only the team that is serving can score points, and they do so if they win the volley. If they lose the volley, then the other team doesn't get a point yet, but they do get to serve next. (If the serving team wins the volley, they continue to serve.) There are several ways to lose a volley:[2]

1. One of your team's players touches the ball twice in a row.
2. The ball is touched at least four times in a row by players of your team.
3. The ball touches the ground in your court.
4. The ball touches the ground out of bounds, and one of your team's players was the last to touch the ball.
5. One of your team's players touches the ball right after one of your players has served it.

So given a sequence of who touched the ball, our goal is to find out the current score (notice that "touching" the ball may include serving it). We also should output an error message if at any point in time, the wrong team served.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of a data set contains three numbers n_1, n_2, t with $2 \leq n_1, n_2 \leq 10$. n_1 and n_2 are the numbers of players on Team A and Team B (notice that these need not be equal, or even close), and $1 \leq t \leq 1000$ is the length of the sequence of "ball touches".

The second line is a sequence of t "ball touches", separated by one empty space, as follows:

- A number i with $1 \leq i \leq n_1 + n_2$ means that player i touched the ball (this may have been a serve). Assume that players $1 \ldots, n_1$ are on Team A, and $n_1 + 1, \ldots, n_2$ on team B.
- The characters '`A`' or '`B`' mean that the ball touched the ground in the court of Team A resp. Team B.
- The character '`X`' means that the ball touched the ground out of bounds.

The first "ball touch" is always a number, representing the person who served first.

We also promise you that the input will never contain impossible sequences (except wrong serves). For instance, you won't have the sequence "`1 X X`", where the ball bounced twice outside when really, some team should have been serving it for the third touch.

[1] Hence the name.

[2] We will ignore effects of the net here, as well as a few other details, including the fact that the person serving the ball must normally rotate.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the score of the two teams, separated by a single white space. If at any point during the sequence, a player from the wrong team served, output "`Wrong Serve`" instead of the score.

Sample Input/Output

Sample input `ball.in`

```
3
3 3 6
1 2 5 A 4 X
2 2 20
1 3 4 3 1 2 1 2 3 1 2 4 A 4 2 4 1 2 X 3
6 6 5
10 1 2 B 7
```

Corresponding output

```
Data Set 1:
0 1
Data Set 2:
0 2
Data Set 3:
Wrong Serve
```

Problem B: Sand Castles

File Name: `castles.cpp|castles.java`

Input File: `castles.in`

Description

It is pretty amazing what beautiful constructions one can build out of just sand and a bit of water. In fact, it is so amazing that A.-L. Barabasi has several publications in high-quality scientific journals about it.

We won't be quite as ambitious here. All we want is a few cylindric towers (conveniently formed by little beach buckets) on a nice plain patch of sand, with a little rectangular moat around it. But Ah!, there's a catch. You cannot just steal sand from a neighboring patch, or dispose of excess sand there.[3] That means that if you want a very deep moat relative to the castle, you will probably have to raise the "base level", whereas if your mote is shallow, and you have a lot of "buildings", then your base level will be lower than on neighboring patches. You are to find out just how many centimeters the base level is raised or lowered.

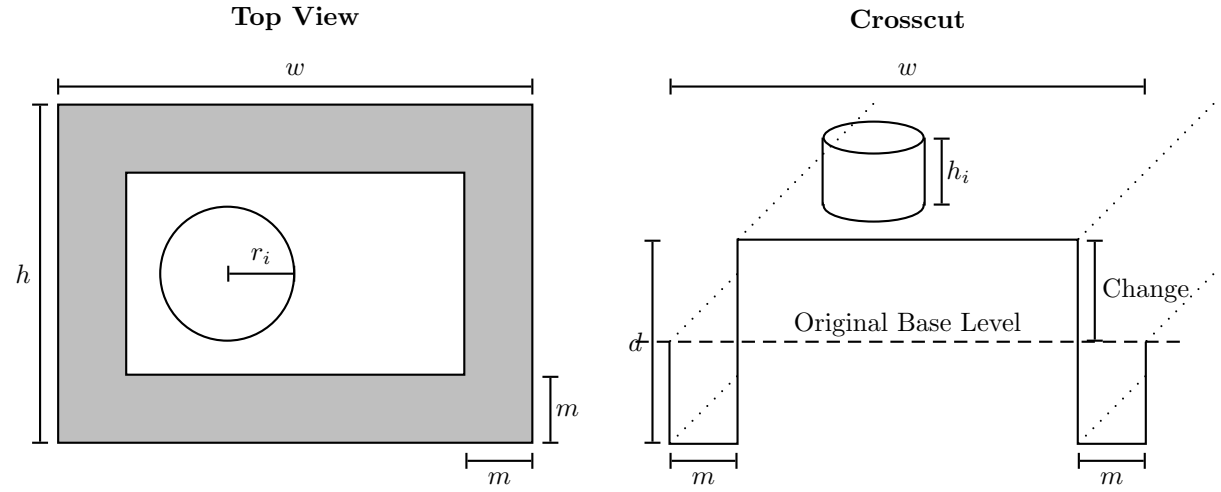
Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of each data set contains numbers w, h, m, d, b. w and h are the width and height of your patch in centimeters (real numbers). m is the width of your moat (thus, $m < \frac{1}{2}\min(h, w)$), and d is the depth of the moat, also in centimeters (also real numbers). Assume that the moat is always built at the perimeter of your patch. Finally, the integer number $0 \leq b \leq 100$ is the number of "buildings" you build.

This is followed by b lines, each containing two numbers h_i and r_i. Here, the number h_i is the height of the i^{th} building (or bucket you used), and r_i is the radius. You can assume that all buildings are cylindrical — for our purpose, it makes no difference if they are stacked on top of each other, or built next to each other. (Also, you need not worry about the case that the base area of a building is larger than the remaining area of your patch.)

Here are two pictures, viewed from the top and as a crosscut, illustrating the quantities.



Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the total change in the level of your patch, in centimeters, and rounded to two decimals. (If the level of your patch is lowered, then output it with a negative sign, of course.)

[3] You may think that this is a silly rule. In fact, it sometimes applies to building projects in the real world. One of the constraints when the Getty Center was built was that they were not allowed to add or remove soil from the patch of land. That would explain those weird round towers at the corners — they are added ornaments to hide excess soil.

Sample Input/Output

Sample input `castles.in`

```
3
100.0 80.0 20.0 10.0 0
70.0 100.0 0.0 0.0 1
30.0 20.0
100.0 100.0 10.0 5.0 3
30.0 20.0
4.0 3.0
4.0 3.0
```

Corresponding output

```
Data Set 1:
7.00
Data Set 2:
-5.39
Data Set 3:
-1.99
```

Problem C: Swimming with Sharks

File Name: `sharks.cpp|sharks.java`

Input File: `sharks.in`

Description

It appears that at one of the beaches in La Jolla, there is a fairly large number of leopard sharks right by the pier (those are only about 4 feet long, a lot smaller than most sharks). And apparently, some people consider it a very exciting experience to actually swim in the water with those sharks. We don't know about you, but all of the contest organizers seem to be unanimous in their agreement that this does not seem like the greatest idea. We also all agree that if we were in the water with a bunch of leopard sharks, our main goal would be to stay as far away from them as possible. Luckily, the computer could help us with that (in particular if it were water-proof).

We model the water as a $w \times h$ two-dimensional grid of integer coordinates. At time 1, you start at point $(1, 1)$. In each time step, you can swim one square to the right, left, up, or down, or stay where you are (so long as you don't leave the area, of course). You will be given the position at which sharks are at all given times, as well as a time horizon t to consider. You are to find a way to stay as far from all sharks as possible for t steps. More precisely, let d be the closest you ever came to any shark during any of the t time steps. (Distance between two points (x_1, y_1) and (x_2, y_2) is measured in the Euclidean norm $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$). Then, the goal is to find a "move plan" making d as large as possible, while ending up at position $(1, 1)$ again at time t.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of each data set contains integer numbers w, h, t, s. $1 \leq w, h \leq 10$ are the width and height of the water expanse we are considering. $1 \leq t \leq 100$ is the time for which you are in the water. $1 \leq s \leq 10000$ is the number of "shark sightings" in the file.

This is followed by s lines, each containing three integer numbers x_i, y_i, t_i (with $1 \leq x_i \leq w, 1 \leq y_i \leq h, 1 \leq t_i \leq t$). This means that at time t_i, there is a shark at position (x_i, y_i). These are not necessarily sorted by any parameter.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the closest you ever came in contact with a shark in the best possible solution, rounded to two decimals. (Notice that an output of `0` means that you cannot avoid occupying the same square as a shark at some point.)

Sample Input/Output

Sample input `sharks.in`

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

Corresponding output

```
Data Set 1:
0.00
Data Set 2:
1.41
```

Problem D: Sun Bathing

File Name: `sun.cpp|sun.java`

Input File: `sun.in`

Description

Based on empirical observations, it appears that despite the many exciting things to do at the beach, the favorite of most people is just to lie in the sand. As you are all aware, prolonged exposure to sun can cause painful sunburns, so the use of sunscreen is highly recommended. Another approach pursued by many people is to lie under a parasol, protecting them from rays. But if you know from when to when you will lie at the beach, where should you place the parasol to protect as much of your body as possible? [4]

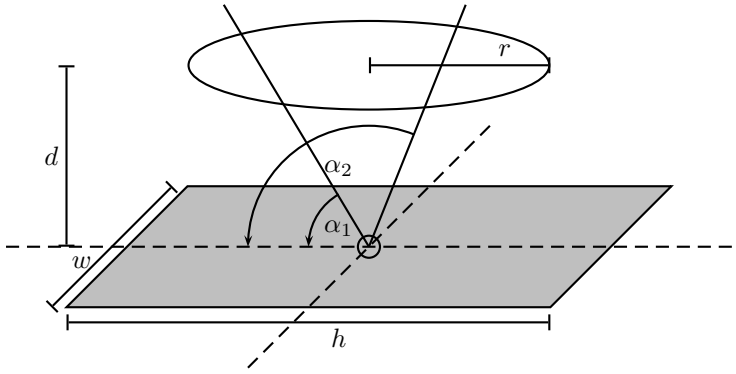
Here, we will answer a slightly easier question. Given the position of your body and the parasol, calculate which percentage of your body is exposed to the sun for at least part of the time. For simplicity, we will treat your body as rectangular, of size $h \times w$ centimeters. We treat the beach as a two-dimensional plane, with the origin at the center of your body. The parasol is assumed to be a circle of radius r, mounted horizontally at distance d above your body, with the center at the origin as well. You are sunbathing for a period from α_1 to α_2, where $1^\circ \leq \alpha_1 < \alpha_2 \leq 179^\circ$ are the angle that the sun makes with the beach at the beginning and end of your sunbath. (Assume that the sun is a point infinitely far away from the Earth, and travels straight from the left to the right.)

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

Each data set consists of a single line, containing (in this order) the real numbers $w, h, r, d, \alpha_1, \alpha_2$.

Here is a figure illustrating these quantities. Notice that the sun travels "from your feet to your head", and *not* "from your left arm to your right arm".



Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the percentage of your body that will be exposed to sun rays at some time, rounding to two decimals.

Sample Input/Output

Sample input `sun.in`

```
2
0.5 1.8 5.0 0.2 70.0 120.0
0.5 2.0 0.75 1.0 45.0 90.0
```

Corresponding output

```
Data Set 1:
0.00%
Data Set 2:
76.41%
```

[4] In a French student science competition, one winning entry was a "para-tournesol", a "sunflower parasol". Using a light sensor, the parasol moves as the sun does, protecting you as much as possible. We won't explore that avenue here.

Problem E: New Friends

File Name: `friends.cpp|friends.java`

Input File: `friends.in`

Description

During a nice day at the beach, you can also meet some nice new people who might become your friends.[5] If you want to stay in touch, you should make sure to get their address, e-mail, or something like that, in addition to their name. Afterwards, you may start to wonder how many different towns these people were actually from.

Of course, this is sometimes complicated by mistakes you make in writing down their information. If you write that one person is from "Pasadena", and another from "Passadena", you'll think they were from two towns. So we need a computer program to correct for these mistakes.

We'll have a list of names of towns that people are from. These are strings with uppercase and lowercase letters, as well as the character '`-`'. We will ignore case, i.e., "`SAN-DIEGO`" and "`san-diEgo`" denote the same town. In addition, when two strings differ by exactly one character (insertion, deletion, or overwrite), they are considered potentially the same city. For instance "`SanDIego`" and "`san-diego`" could be the same city, but "`san-diego`" and "`san-deigo`" could not. You are to compute the minimum number of cities that these people could be coming from.

Notice that people can only come from the same city if their city names differ at most in one character. It is not enough that there be some city name such that all of them differ in at most one character from that name. For instance, in the first example below, even though every city name differs from "`Tijuana`" by at most one character, the second and third person cannot be from the same city, so you need to assume they are from two cities.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of the data set contains a number n with $1 \leq n \leq 10$, the number of new friends you have met. This is followed by n lines, each containing one string of length between 1 and 100 characters, a place where one of your new friends originated.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the minimum number of cities that these people could be from.

Sample Input/Output

Sample input `friends.in`

```
2
3
Tijuana
tejuana
TI-Juana
5
Cancun
Can-Cun
can-can
canccun
CANKUN
```

Corresponding output

```
Data Set 1:
2
Data Set 2:
3
```

[5] Of course, you also meet a lot of people who you wouldn't want to be friends with. Ever.

Problem F: Beach Party

File Name: `party.cpp|party.java`

Input File: `party.in`

Description

For a rousing finale of a nice day at the beach, we are hanging around for a party, where various music groups set up stages, and perform, while people listen, dance, sing along out of tune, write their CS201 programming assignment, or do any number of other things. As there are many groups playing, everyone will simply flock to the music they like best. Unluckily, this sometimes means that you end up in a different place from most of your friends.

For instance, suppose that there are two stages, and on one of them, they perform Gregorian Chant, while the other one has Polkas.[6] You prefer Chant, but your best friend prefers Polkas. Now, if only instead of Polkas, they were playing Country (which your friend hates), he would come listen to Chant with you. So we may be wondering, given a choice of music styles, which ones, if assigned to stages, would result in the largest number of friends staying with you.

More formally, there are s stages, and $m \geq s$ music styles. Each stage has to get exactly one music style, and no style can be played on two stages. Each person, including you, has a preference order on styles, and will simply go to the stage with the most preferred music. The question is which assignment of music to stages will have the largest number of friends at the same stage as you.

Input

The first line contains a number $K \geq 1$, which is the number of input data sets in the file. This is followed by K data sets of the following form:

The first line of the data set contains three numbers s, m, n, the number of stages, music styles, and friends ($1 \leq s \leq 10, 1 \leq m \leq 20, 1 \leq n \leq 100$). This is followed by n lines, each describing one friend (the first one being yourself). Each friend description is a permutation of the music styles $\{1, \ldots, m\}$.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the maximum number of friends (including yourself) that will listen to the same music as you for the most favorable assignment of music to stages. (Notice that you do not need to output the actual assignment achieving the maximum.)

Sample Input/Output

Sample input `party.in`

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

Corresponding output

```
Data Set 1:
1
Data Set 2:
3
```

[6] Given these choices, we wouldn't blame you for leaving the beach.