The Cornell Weather Programming Contest

Cornell ACSU Programming Contest, Fall 2001

October 13, 2001

Sponsored by Trilogy

As those of you who have been at Cornell for a while already know, and the newcomers among you will soon find out, the weather in Ithaca tends to pose some — shall we say — challenges to people living here. It's definitely not true that the four seasons are rain, rain, rain and rain; after all, we have those great cold winter days, and on those humid 90-degree summer day when everyone is dying for a nice quick rain shower, the rain seems to completely forget about its duties and its reputation.

As a service to the community, we thought that we'd let you write programs that will solve some of the more common problems associated with the Ithaca weather. Maybe, they will come in handy at some point.

While there is an underlying theme again for this year, there will again *not* be a common input format. As usual, problems need not be solved in the same order that they appear in this problem set — the order has nothing to do with the difficulty level.

As a change over previous years, you will have the choice between reading your input from a file and writing output to a file, or reading input from `stdin` and writing output to `stdout`. When you submit your program, just select the option that matches your I/O style. Note, however, that you cannot mix, e.g. you cannot read from a file and write to `stdout`.

The names of your program files, as well as those of your input and output files (if you decide to use file I/O) must match exactly the ones prescribed in the problem set. If they don't, your submission is considered incorrect.

[Including this page, this problem set should contain 4 double-sided pages. If yours doesn't, please contact one of the helpers immediately.]

Problem A: Raindrops keep falling on my head

File Name: RAINDROP.CPP|RAINDROP.JAVA

Input File: RAINDROP.IN

Output File: RAINDROP.OUT

Description

At 2:30 am in the CSUG-Lab, your code for the 412/413 homework finally works. You're about to head home when you notice that it's raining. Of course, your umbrella is lying happily at home.

Being the ingenious computer expert that you are, you of course immediately see the solution. If only you knew in advance the exact time and location at which the raindrops will be falling, you could time your walk so as to avoid as many raindrops as possible. However, there's the additional constraint that you do want to get home eventually.

For simplicity, we shall assume that your walk home is a straight line. Time is divided into discrete steps, and in each step, you can either move 1 unit forward, 1 unit backward, or stay in the same place. If you and a raindrop are in the same place at the same time — well, you get hit. At time 0, you are at position 0 (Upson Hall), and by time T, you will have to have reached position H (your home). You can never get hit by raindrops at positions 0 or H.

Input

The first line will contain the number K of input data sets, followed by K data sets of the following form:

The first line of a data set contains three numbers, T, H and D ($1 \leq H \leq T \leq 200$, $0 \leq D \leq 5000$). T is the amount of time you have to get home, H is the position of your home, and D is the total number of recorded raindrops.

The following D lines each contain two numbers t and p, which are the time and position at which the raindrop hits the ground (or you). They will be sorted by non-decreasing times t, and no two raindrops will hit the same position at the same time.

Output

The output file should contain one line for each input. Each line contains first the number of the data set, and then the minimum number of raindrops that will hit you on your way home (subject to the moving and time constraints stated above).

Sample Input/Output

Sample input RAINDROP.IN

```
1
7 5 12
1 2
1 3
2 1
2 3
3 2
3 3
3 4
4 3
4 4
5 4
6 3
6 4
```

Corresponding output RAINDROP.OUT

```
Data Set 1: Hit by 1 raindrops
```

Problem B: Wind Chill

File Name: `WIND.CPP|WIND.JAVA`

Input File: `WIND.IN`

Output File: `WIND.OUT`

Description

If you simply looked at the thermometer over an entire year, you could get the impression that the weather in Ithaca is rather temperate. Yet those hot humid summer days, and those cold winter days with icy wind certainly don't feel temperate. That is of course because the temperature feels very different when it is humid or there is wind.

Scientists have determined (experimentally) formulas that describe what a temperature feels like to us given wind speed or relative humidity. For the rest of this problem, T will denote the temperature in degrees Fahrenheit, v the wind speed in miles per hour, and H the relative humidity — contrary to custom, we will not write it in percent, but as a value $0 \leq H \leq 1$, where 1 corresponds to 100%.

WC denotes the temperature in degrees Fahrenheit that it will feel like to you on a day with wind speed v. The wind chill formula is

$$\mathrm{WC} = 91.4 + (0.474677 - 0.020425 \cdot v + 0.303107 \cdot \sqrt{v}) \cdot (T - 91.4).$$

The heat index formula tells you how hot it feels on a humid day, and is a little more complicated.

$$\begin{aligned} \mathrm{HI} = & -42.379 + 2.04901523 \cdot T + 1014.333127 \cdot H - 22.475541 \cdot T \cdot H \\ & -0.00683783 \cdot T^2 - 548.1717 \cdot H^2 + 0.122874 \cdot T^2 \cdot H + 8.5282 \cdot T \cdot H^2 - 0.0199 \cdot T^2 \cdot H^2. \end{aligned}$$

Since we want the output in both Celsius and Fahrenheit, here is the conversion (C denotes the temperature in degrees Celsius).

$$C = (T - 32) \cdot \frac{5}{9}.$$

Input

The first line will contain the number K of input data sets. This is followed by the K data sets, each consisting of a single line with three floating point numbers T, v and H. T is the base temperature in degrees Fahrenheit, v the wind speed, and H the relative humidity.

Output

For each data set, output its number, followed by what the temperature feels like on that day, both in degrees Fahrenheit and Celsius (and rounded to two decimals each). If the day is cold (base temperature below 60 Fahrenheit), use the wind chill formula, otherwise (base temperature 60 Fahrenheit or above) use the Heat Index formula.

Sample Input/Output

Sample input `WIND.IN`

```
3
30.0 45.0 0.3
85.0 10.0 0.8
100.0 30.0 0.1
```

Corresponding output `WIND.OUT`

```
Data Set 1: It feels like -6.16 degrees (-21.20 Celsius)
Data Set 2: It feels like 96.81 degrees (36.01 Celsius)
Data Set 3: It feels like 94.75 degrees (34.86 Celsius)
```

Problem C: The glass is half full

File Name: `VASE.CPP|VASE.JAVA`

Input File: `VASE.IN`

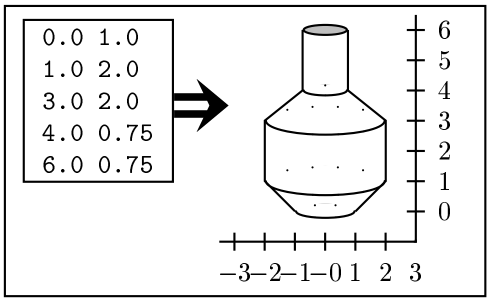
Output File: `VASE.OUT`

Description

At last year's Salvation Army Christmas Raffle, you won this beautiful geometrically shaped flower vase, and after months and months of hinting, someone finally gave you some flowers to put in it. Now, all you need to do is provide some water, but with Cornell's current tuition, you have to save on the water bill, so you decide to turn the natural weather hazards to your advantage, and place the vase outside to fill it.

You will be given a list of intervals, and for each interval the speed of rain flow (in cubic meters per square meter and second). That is, if the rain flows at $0.1\frac{m^3}{m^2 s}$, and you keep a bucket with an open surface of $0.002m^2$ outside for one minute, you would collect $0.012m^3$ of water (that's 12 liters). Also, you will be given the height up to which you want to fill your vase (in meters), and are supposed to calculate the time it will take to fill the vase to that point.

Alas, as we said in the first paragraph, the vase is geometrically shaped. It's not just a cylinder, but a "sequence" of cone pieces. That means that it is always circular, but the radius of the circle varies. You will be given a list of (height, width) pairs — between those pairs, the surface forms a straight line. An illustration probably explains this much better than many words. As a reminder, the volume of a cylinder of height h and radius r is $\pi r^2 h$, and that of a cone of height h and bottom radius r is $\frac{1}{3}\pi r^2 h$.



Input

The first line will contain the number K of input data sets. This is followed by K data sets, each of the following form:

The first line contains three numbers, D, I and f. D and I are integers ($2 \leq D \leq 100, 1 \leq I \leq 100$), while f is a floating point number. D is the number of (height, width) pairs describing the vase. I is the number of rain speed intervals. f is the height up to which you want to fill the vase.

The next D lines each contain two floating point numbers h and w. $w > 0$ is the radius of the vase at height h. These lines will be sorted by increasing height h, and the first height will always be 0. The last height will always be at least as large as the filling height f.

The following I lines each contain two floating point numbers t and v. Here, $v \geq 0$ is the water speed starting at time t (and until the time specified in the following line, or ever after, if it is the last line). These lines will be sorted by increasing times t. The first time will always be 0.

Output

For each data set, output its number, and the earliest time (in seconds) at which the vase is filled up to height f, rounded to two decimals. If the vase will never be filled to height f, output "never".

Sample Input/Output

Sample input VASE.IN

```
2
5 4 4.0
0.0 1.0
1.0 2.0
3.0 2.0
4.0 0.75
5.0 0.75
0.0 2.2
3.0 1.1
5.0 0.5
6.0 1.0
5 4 4.0
0.0 1.0
1.0 2.0
3.0 2.0
4.0 0.75
5.0 0.75
0.0 0.2
3.0 0.1
5.0 0.02
6.0 0
```

Corresponding output VASE.OUT

```
Data Set 1: Vase is filled to height 4.00 at time 18.66
Data Set 2: Vase is never filled to height 4.00
```

Problem D: Paper Boats

File Name: BOATS.CPP|BOATS.JAVA

Input File: BOATS.IN

Output File: BOATS.OUT

Description

Once again, it's pouring. Everybody is retreating to their cozy fireplaces and studying for the wine-tasting class, only some few Engineers are out on the Quad[1] and making paper boats to swim on the little rivulets that have formed. They soon notice that, depending on where they start their boats, they sometimes run into each other. Being engineers, they get curious and want to know more. So they go ahead and draw a precise map of the water system.

The map contains several reference points, which include all the points at which two or more streams meet. Also, for each stream, the map includes the time the boats need to travel along that stream). Since the water is running downhill, the boats always go the same way along any given stream, and can never go around in circles. Streams may join or split at reference points. If they split, it's impossible to predict which of several turns a boat will be taking. What they want from you is a program that, given a map and the starting points of two boats, will tell them if it's possible for the two boats to meet, and when is the earliest time this could happen.

Input

The first line is the number K of input data sets, followed by the K data sets, each of the following form:

The first line contains four numbers, N, M, P and Q. N is the number of reference points ($1 \leq N \leq 100$), and M is the total number of streams ($0 \leq M \leq 1000$). P and Q are the starting points of the two boats (and hence satisfy $1 \leq P, Q \leq N$).

This line is followed by M lines, each describing one stream, by giving you the starting point s, the ending point t, and the (integer) length L. Of course, s and t will satisfy $1 \leq s, t \leq N$ (and $s \neq t$), and L will satisfy $1 \leq L \leq 20$. This entry means that a boat can travel from s to t (but not from t to s) in L time units. There will never be two entries for the same pair (s, t).

Output

For each data set, output its number, and the earliest time at which the two boats could meet, or "never", if the boats can never meet.

Sample Input/Output

Sample input BOATS.IN

```
2
4 5 1 2
1 2 2
1 3 4
2 4 10
2 3 3
3 4 5
4 5 1 2
1 2 2
1 3 4
2 4 11
2 3 3
3 4 5
```

Corresponding output BOATS.OUT

```
Data Set 1: Boats can meet at time 10
Data Set 2: Boats can never meet
```

[1] Notice to all freshmen and first-year students: There used to be a so-called "Engineering Quad" with trees and grass where the huge construction site is now.

Problem E: Snowball Fight

File Name: `SNOWBALL.CPP|SNOWBALL.JAVA`

Input File: `SNOWBALL.IN`

Output File: `SNOWBALL.OUT`

Description

It's snowing! This admittedly happens rarely in Ithaca[2], but when it does, it offers students the great opportunity to go out, have snowball fights, and display their complete lack of aim. As you all know, a real snowball fight has to have two teams fighting against each other, not people randomly throwing snow around. But how do we form the teams? We want to mix up the various years and genders as much as possible, yet at the same time maintain an even skill level.

More specifically, each participant is characterized by three parameters: their gender, their year, and their skill. Each team should have half of the participants of each gender and each year (if there is an odd number for some category, one team may have one more participant from that category than the other), and subject to this constraint, the cumulative skills should be as close as possible. This might get fairly complicated, so we'd better write a computer program for it.

Input

The first line is the number K of input data sets, followed by the K data sets, each of the following form:

The first line contains the number, N of participants ($1 \leq N \leq 20$). This line is followed by N lines, each containing first a letter g, which is either '`f`' or '`m`', then a number y with $1 \leq y \leq 10$, and then a number s with $0 \leq s \leq 1000$. Here, g is the gender, y the year, and s the skill level of a participant.

Output

For each data set, output its number, and the smallest difference in skill levels between the two teams which is attainable subject to the constraints of splitting up all years and all genders as evenly as possible. The skill level of a team is the sum of the skill levels of all its members.

Sample Input/Output

Sample input `SNOWBALL.IN`

```
2
4
m 3 10
m 2 0
f 1 10
f 2 20
6
m 3 5
m 2 9
m 1 7
f 3 9
f 2 7
f 1 5
```

Corresponding output `SNOWBALL.OUT`

```
Data Set 1: Least skill difference is 20
Data Set 2: Least skill difference is 4
```

[2]Please note the clever use of *irony* here.

Problem F: The Four Seasons

File Name: SEASONS.CPP|SEASONS.JAVA

Input File: SEASONS.IN

Output File: SEASONS.OUT

Description

As you might have noticed during the previous problems, it does rain quite a bit in Ithaca. But there are also nice times. Actually, there's nice stuff in almost every season — it just depends on what you like to do. To help your friends decide when they should best come to visit you, you are going to write an **Ithaca Season Personality Test**.

There are 15 categories, representing weather, hobbies, events, etc. For each of the categories, the test subject can enter on a scale from -3 to 3 how much they like this category (or care about it). Each season will have its own multipliers for each category (given in Figure 1), and by adding up the weighted preferences, we can determine how much the subject likes each season, and make a visit recommendation.

#	Category	Spring	Summer	Fall	Winter
1	Rain	2	1	3	-2
2	Sunshine	1	3	1	-1
3	Snow	2	-2	-1	3
4	Gorge-dipping	0	3	1	0
5	Skiing/Sledding	1	0	0	3
6	Book sale	2	0	3	0
7	Foliage	0	1	3	0
8	Flowers	3	2	1	0
9	Hockey	2	1	1	3
10	Action	3	-2	3	-1
11	Programming Contest	2	0	3	1
12	Classes	3	1	3	1
13	Barbecues	1	3	1	0
14	Hikes	3	2	3	2
15	Concerts	3	2	3	1

Figure 1: Categories and multipliers

Input

The first line is the number K of input data sets, followed by K data sets, each consisting of 15 integer numbers between -3 and 3 in one line.

Output

For each data set, output its number, followed by the recommended seasons. The recommended season is the one achieving the highest positive score. If no season achieves a positive score, recommend that the person not come to Ithaca at all. If several seasons achieve the highest score, output them all, in the same order as they are listed in the table (i.e. Spring, Summer, Fall, Winter).

Sample Input/Output

Sample input SEASONS.IN

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

Corresponding output SEASONS.OUT

```
Data Set 1: Come to Ithaca in winter!
Data Set 2: Don't come to Ithaca!
Data Set 3: Come to Ithaca in spring or fall!
```