The Olympic Contest

USC Programming Contest, Fall 2017

October 01, 2017

Sponsored by VSoE, Electronic Arts, Facebook, Google, Northrop Grumman

As you may have heard, Los Angeles just won the bid to host the 2028 Olympic Summer Games. This will be the third Olympics that LA has hosted, after 1932 and 1984. The Coliseum (which some of you may be familiar with) was used in all of them — in fact, it was built for the 1932 Olympics. As part of celebrating this event[1], we will explore in this contest some of the computational problems that are raised by hosting such a huge event. In particular, as you will see, almost every sport has some computational problems of its own that you will help them solve.

You can solve or submit the problems in any order you want. 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` (the screen). 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, people, etc.) always starts at 1, not at 0.

And a piece of advice: you will need to print floating point numbers rounded to two decimals. Here is how you do that:

C: `printf ("%.2f", r);`.

C++: `cout.precision (2); cout << fixed << r;`.

Java: `System.out.print ((new java.text.DecimalFormat("#.##")).format(r));`

[Including this page, the problem set should contain 8 pages. If yours doesn't, please contact one of the helpers immediately.]

[1] Given the costs and corruption involved in hosting Olympic Games, this may well not be quite the blessing it seems like initially.

Problem A: Opening Ceremony

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

Input File: `ceremony.in`

Description

Olympic games always start with a beautiful opening ceremony. In addition to all kinds of dance numbers, the opening ceremony features the athletes of all participating countries walking around the main stadium, waving at the crowds, and generally acting like USC students at their graduation. And much like graduation, there are quite a few who would actually prefer to skip this ceremony, and work on their CS homeworks — sorry, we meant "train for their competitions." That's why each country always marches as a bloc — this way, it is easier for the organizers to tell which country has the largest number of delinquents[2]. Imagine how much harder this would be if the athletes were just marching in any order. You'd have to look at their jerseys and keep a careful count for each country. Well, that's exactly what you will do!

You will be given for each country how many athletes it has registered. Then, you will be given a sequence of country affiliations for all athletes that marched in the opening ceremony. You are to output the maximum number of delinquents of any country.

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 two integers c, n. $1 \leq c \leq 1000$ is the number of countries participating in the Olympics, and $0 \leq n \leq 10000$ is the number of athletes at the opening ceremony.

This is followed by a line with c integers $m_1, m_2, \ldots, m_c$; here, $m_i \geq 0$ is the number of athletes registered for country i. Next comes a line with n integers $b_j \in \{1, 2, \ldots, c\}$. Here, b_j is the country affiliation of the j^{th} athlete in the opening ceremony. We guarantee that there will never be more than m_i athletes from country i, for any i.

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 delinquent athletes for any country.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `ceremony.in`

```
2
3 8
5 5 2
1 2 1 2 1 2 1 2
5 6
10 10 10 10 10
1 2 4 5 3 1
```

Corresponding output

```
Data Set 1:
2

Data Set 2:
9
```

[2] And you always thought it was because each country wants to present itself as a unified group of friends?!

Problem B: Rock Climbing

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

Input File: `climbing.in`

Description

As of 2020, rock climbing will for the first time be an Olympic sport. In rock climbing you have a wall (natural or man made) with little holes and protrusions that you can use to put your fingers or toes. By combining good technique and careful planning with strength, your goal is to reach a particular spot, typically the top of the wall. Often, the planning aspect is quite complex, since you need to make sure that a limb is available to put somewhere useful for the next step. It's no accident that rock climbing is very popular with math and CS folks — after solving this problem, you will know why.

We model the rock climbing planning problem as follows: you will be given a number of 2-D coordinate locations. Those are places that you could use to put your toes or hold on with your fingers. In each step, you can move exactly one of your limbs to a new location, but there are constraints on how far away your limbs can be from each other. Specifically, your arms cannot be more than 2 units apart, and neither can your feet. Your left arm and left foot cannot be more than 3 units apart, and neither can your right arm and right foot. And your left arm and right foot cannot be more than 4 units apart, and neither can your right arm and left foot. You can never have two limbs in the same location. Finally, at any given time, both of your feet must be no higher than both of your hands, meaning that their y-coordinates are no bigger than the hands' y-coordinates.

You will always start with your left foot in location 1, right foot in location 2, left hand in location 3, and right hand in location 4. (We will ensure that this is always legal.) You have climbed the wall when one of your limbs reaches location n. Your goal is to do so in as few moves as possible.

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 one integer $4 \leq n \leq 30$ (the number of places to put your limbs). This is followed by a line with $2n$ doubles x_i, y_i; here, x_i, y_i are the coordinates of location i.

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 moves until one of your limbs touches location n for the first time. If there is no way to get to location n, output "Impossible" instead.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `climbing.in`

```
2
10
0 0  1 0  0 1.4  1 1.5  1 2.5  3 1  2.5 2  2.5 5  1 3.1  3 3
7
0 0 1 0 0 1.5 1 1.5 1.1 1.5 3 0.5 3 3
```

Corresponding output

```
Data Set 1:
4

Data Set 2:
Impossible
```

Problem C: Volleyball Scores

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

Input File: `volley.in`

Description

Volleyball has been an Olympic sport since 1964. Teams of 6 players (in beach volleyball: 2 players) hit a ball back and forth, but each team can only touch the ball up to three times until they need to hit it to the other side. There are additional rules, such as the same player not being allowed to touch the ball twice in a row, etc., which make keeping track of the score quite tedious. This is where you come in and write a program to solve this problem for them.

Here's a more precise refresher of volleyball rules. The two teams each consist of 6 players: Team A's players are numbered 1–6 in this problem, and Team B's players are 7–12. The two teams take turns serving. The other team wins the point when one of the following things happen to your team:[3]

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 before that.
5. One of your team's players touches the ball right after one of your players has served it.

Given a sequence of who touched the ball, your goal is to find out the current score (notice that "touching" the ball may include serving it). You 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 a number t with $1 \leq t \leq 1000$. The second line is a sequence of t "ball touches", separated by one empty space, as follows:

- A number i with $1 \leq i \leq 12$ means that player i touched the ball (this may have been a serve).
- 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. The point may be ongoing at the end of the sequence. The input will never contain impossible sequences (except wrong serves).

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 `volley.in`

```
2
21
1 X 9 A 3 7 7 7 3 4 5 B 2 8 11 10 3 4 9 B 12
7
1 X 9 A 8 X 7
```

Corresponding output

```
Data Set 1:
3 2

Data Set 2:
Wrong Serve
```

[3] 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.

Problem D: Single Elimination

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

Input File: `single.in`

Description

Single Elimination is a popular tournament format used, for example, for all major tennis tournaments, March Madness in Basketball, the College Football Playoff, and of course many Olympic events. It works most easily when the number of players is a power of 2. In each round, the remaining players are paired up and play matches. All losers are out, and the winners go on to the next round. After $\log_2(n)$ rounds, the one remaining player is the winner.

When the best player can beat every other player, he/she will always emerge as the winner. Things get interesting if every player has one or more other players they have trouble with. If a player is "unlucky" and runs into such a player early, they may be eliminated, which could benefit another "lucky" player who would have lost to them later. Thus, being able to "seed" the tournament (which means determining who plays whom in each round) can be quite a useful tool in getting your favorite player to win the tournament. In this problem, you will figure out which players (out of a field of 16) have a chance to win the tournament if the seeding is designed perfectly for them.

You will be given, for every pair of players, who will win if those two player play each other. (We assume that this will always be the result.) The question is then how to design the matches for the first round, then for the winners in the second round, etc., so that a particular player wins. Your final output will be the list of all players who could possibly win.

Input

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

There are 16 lines, each containing 16 numbers $a_{i,j} \in \{0, 1\}$. If $a_{i,j} = 1$, then i beats j when they play, whereas $a_{i,j} = 0$ means that j beats i. We guarantee that exactly one of $a_{i,j}$ and $a_{j,i}$ is equal to 1, and the other is 0. Obviously, the value of $a_{i,i}$ doesn't matter, as i will not play himself/herself; we include it to make the input easier to parse.

Output

For each data set, output "`Data Set x:`" on a line by itself, where `x` is its number. Then, list, one per line, all players who can win the tournament if the seeding is designed perfectly for them. The players should be listed in increasing order.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `single.in`

```
1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1
0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Corresponding output

```
Data Set 1:
1 2 3 4 5 7
```

Problem E: Triple Jump

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

Input File: `jump.in`

Description

Triple Jump is kind of a weird discipline. Like most Track&Field disciplines, it most likely came out of skills that were useful in fighting battles (think running fast, jumping far or high, hurdling things, or throwing spears or hammers). In Triple Jump, you get a runup, and then jump three times in a row. Your distance is the distance from the start of the first jump to the "end" of the third jump. Like with regular long jump, you must start the first jump from a designated "take off board;" if you over- or understep it, your jump does not count. Also, you must be careful not to fall backwards after you land, since the distance is counted to the least far imprint left in the sand by a part of your body. (So if you land on your feet, but then tip backwards on your butt, the distance from the start of the jump to where your butt hit the ground is what counts.)

In this problem, you will be given a sequence of locations where your body parts (feet, etc.) touched the ground. The take off board extends from location 30.0 to 30.2 (inclusive). The locations in the sequence could increase or decrease (because you turned back, or fell backwards). The triple jump starts the first time any body part is inside the $[30.0, 30.2]$ interval. The length is measured from 30.0 (not the actual location) to the smallest location that is at least 3 steps later than the start of the actual jump. This distance could be negative. If you never hit the interval, or did not take 3 steps after touching it, the distance is counted as 0.

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 of the data set contains an integer $1 \leq n \leq 1000$, the number of steps you took. This is followed by a line with n doubles $0.0 \leq x_i \leq 100.0$; x_i is the place where you touched the track/sandbox on the i^{th} step.

Output

For each data set, output "`Data Set x:`" on a line by itself, where `x` is its number. Then output the distance of the triple jump, rounded to two decimals.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `jump.in`

```
4
11
0.0 20.5 15.7 28.3 30.1 39.7 42 48.2 50.7 46.561 52
8
1.5 19.2 28.3 30.01 38.5 45.1 51.2 60.9
6
10.7 28.3 30.3 38.1 45.2 53.87
7
15.0 30.1 38.2 45.6 56.3 40.8 27.2
```

Corresponding output

```
Data Set 1:
16.56

Data Set 2:
21.20

Data Set 3:
0

Data Set 4:
-2.80
```

Problem F: Basketball

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

Input File: `basket.in`

Description

Basketball is the other sport we're supposed to be excited about at USC (besides women's soccer, of course). Somehow, it never seems to quite catch on. So in case you aren't sure how basketball works: each team has 5 players, who pass the ball around and try to get it in the opposing team's basket, which is suspended about 50ft high in the air. There are some rules about how long you are allowed to be in various places, what are the exact ways in which you are allowed to punch the other players, why you can or cannot hold on to the ball in various situations, and various other arbitrary things.

A very important part of basketball is rebounds. No, not that kind! When your team throws the ball in the general direction of the other team's basket, it doesn't always go in. Instead, it will often bounce off the rim, and in a pretty random direction. You'd then rather your team catch the ball and get to take another shot, than the other team catch the ball and break away towards your basket and take a shot. It's thus a very important decision where to place your players to catch those rebounds. On the one hand, you want to place your players so they are close to the places where the ball will likely land. But you also want them to be in places where they are likely to score when they get the ball. And you want them to be in places where they can prevent the other team from scoring if they catch the ball first. Optimizing this is quite difficult, and indeed, professional sports teams do use sophisticated algorithms to determine rebound strategies[4].

The basketball field is a rectangle with corners $(0.0, 0.0)$ and $(94.0, 50.0)$ (measured in feet). Your team's basket is at location $(0.0, 25.0)$, and the other team's basket at $(94.0, 25.0)$. You will be given the locations of the other team's 5 players, and the candidate locations where you can in principle place your own 5 players. You will also be given a list of locations where the ball may bounce, and for each location, the probability that the ball goes there. For each location, the player closest to that location will get the rebound. The player will run in a beeline to that location to pick up the ball, then run in a beeline to the other team's basket. Simultaneously, all of the other team's players will run in a beeline to their own basket to defend it. All players run at the same speed of 20 feet per second. Let t be the number of seconds by which the person with the ball outruns the fastest defender, i.e., how much earlier he/she gets to the basket. (If the fastest defender outruns the person with the ball, t can be negative.) If $t \geq 0$, then the probability of scoring is $1 - 2^{-(t+1)}$. If $t < 0$, then the probability of scoring is 2^{t-1}. In keeping with basic basketball rules, we will treat each scored basket as 2 points.

Your goal is to find the locations for your 5 players that maximize the expected number of points that your team will get from the rebound. (Points for the other team count as negative points.)

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 two integers n, m. $5 \leq n \leq 15$ is the number of candidate locations where you can place your players, and $1 \leq m \leq 100$ is the number of locations where the ball could rebound.

This is followed by a line with ten doubles $x_1^-, y_1^-, x_2^-, y_2^-, \ldots, x_5^-, y_5^-$; these are the locations of the opposing players. Next is a line with $2n$ doubles x_j^+, y_j^+, which give you the candidate locations for your players. Finally, there is line with $3m$ doubles, consisting of m triples $x_k^\circ, y_k^\circ, p_k$. The first two doubles of a triple are the coordinates of the k^{th} location where the ball may bounce, while $p_k \in [0, 1]$ is the probability that the ball bounces to location k. All x coorcinates (players/candidates/rebound) are between 0 and 90, and all y locations between 0 and 50. We will guarantee that $\sum_k p_k = 1$. Also, we will ensure that for each (x_k°, y_k°), there are never two player/candidate locations whose distance to it are the same (or within less than 0.001). So you don't have to worry about how to break ties about who gets to the rebound first.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, on a line by itself, output the maximum expected number of points your team can get by placing its five players perfectly, rounded to two decimals.

[4] One of the leading sports analytics teams doing this kind of computation grew out of USC's own ISI.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `basket.in`

```
1
8 7
52 25  77 35  85 10  88.5 30  80 49.9
70 30  93 25  90 40  90 10  85 12  75 45  66 1  80 25
55 30 0.1 61 28 0.3 89 19 0.3 86 13 0.01 94 26 0.01 10 2 0.18 82 17 0.1
```

Corresponding output

```
Data Set 1:
0.11
```