The Family Contest

USC Programming Contest, Fall 2018

September 30, 2018

Sponsored by VSoE, Electronic Arts, Google, Northrop Grumman

Family plays a very important part in most of our lives. They raise us, and inspire many of our interests, characteristics, fears, etc. They can be our biggest supporters and biggest antagonists; they can understand us better than anyone, and be the most exasperating people to interact with. In this contest, we will explore a number of questions related to family, get-togethers, and their dynamics.

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.]

Problem A: Teenage Mutant

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

Input File: `mutant.in`

Description

In common parlance, a "mutant" is an individual with a very unusual deformity, or — for some superheros — power. To biologists, it simply describes an accidental change in genetic information from a parent to a child. Telling mutation from inheritance is complicated a bit by the fact that some genetic traits are recessive, and may not be expressed for a generation, but then be visible later. Thus, to see whether a trait is really a mutation, you might have to go some generations back.

That's what you'll do here. You will get expressed traits for yourself and a number of your ancestors, described as strings of length n. A trait is a "mutant trait" if your value in that position is different from that of *all* your ancestors. You are to compute what fraction of your traits are mutant traits, i.e., how much of a mutant you are.

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 n, k, with $1 \leq n \leq 100, 1 \leq k \leq 100$. This is followed by $n + 1$ lines of exactly k uppercase letters each. The first of those $n + 1$ lines is a description of your traits, and the remaining n lines each give you the string for one of your ancestors.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number.

Then, output the fraction of traits in which you are a mutant. Do *not reduce* the fraction; for example, if you are a mutant in two out of ten traits, write "2/10", not "1/5" or "0.2".

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `mutant.in`

```
3
1 5
ABABA
AABBB
4 5
AAAAA
ABBBB
CACCC
DDADD
EEEAE
3 4
ABCD
DCBA
ADBC
CABD
```

Corresponding output

```
Data Set 1:
3/5

Data Set 2:
1/5

Data Set 3:
2/4
```

Problem B: Fairness

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

Input File: `fairness.in`

Description

Anyone who has grown up with siblings knows that having siblings is a perfect way to develop an *extremely* acute sense of fairness. Almost every commodity — time with parents, time at the computer, access to the "best" chair at the dinner table — must be shared absolutely evenly, or else it is clear that your parents simply like your sibling much better than you. But probably the most frequent cause of arguments is favorite foods, in particular pizza slices. Things can sometimes get a little easier for the parents if the children have vastly different food preferences: you can safely give more pepperoni to the child who likes meat and more bell peppers to the child who prefers veggies. But sometimes, there is just no simple solution, for example, if all the toppings are in the same corner of the pizza pie.

We will consider a rectangular pizza of size $X \times Y$. You will be given the coordinates at which the toppings sit: each will be a triple (x_i, y_i, t_i), where t_i denotes the type of topping at this location. For each of the T types of topping, you will be told how much each of the n children likes this type of topping. Your goal is to cut the pizza into n *equal-shaped rectangles* (other cuts are too complicated) and assign one rectangle to each child. In fact, the children are particularly fussy, so you cannot even rotate any of the rectanglar pieces: they must all have the same height and width. You cannot throw away any pizza — clearly, it's bad to waste food. Your goal is to make sure that each child likes his/her rectangle at least as much as he/she would like any of their siblings' rectangles. It this is impossible, your program should say so.

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 integers and two doubles $1 \leq T \leq 100, 1 \leq n \leq 100, 1 \leq m \leq 1000$ (the number of types of toppings, the number of children, and the number of toppings on this particular pizza), and $1.0 \leq X, Y \leq 100.0$ (the dimensions of the pizza).

This is followed by n lines, each containing T integers $p_{j,t} \in [0, 10000]$, describing how much child j likes topping t.

Finally, there are m lines, each giving one topping on the pizza. Each such line has two doubles x_i, y_i with $0.0 \leq x_i \leq X$ and $0.0 \leq y_i \leq Y$, giving the location of the i^{th} topping; and one integer $t_i \in \{1, \ldots, T\}$, the type of topping. We promise that no topping will ever be in (or extremely close to) a location that you may need to cut through.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, on one line, output the sum, over all chilcren, of the extent to which they like their own slice of pizza. If there are multiple ways of dividing the pizza such that for each of them, each child likes his/her own slice best, output the maximum sum possible over all such divisions. If it is impossible to divide the pizza such that each child prefers his/her own slice, output "Impossible" instead. Each data set should be followed by a blank line.

Sample Input/Output

Example I/O on the next page.

Sample input `fairness.in`

```
2
4 4 4 1.0 1.0
1 0 0 1
3 3 1 0
2 4 4 0
1 1 2 2
0.2 0.2 1
0.2 0.8 2
0.8 0.2 3
0.8 0.8 4
5 3 5 5.0 1.0
2 3 0 1 4
1 2 5 3 3
1 0 5 3 1
0.1 0.65 1
0.3 0.9  1
0.5 0.55 2
0.7 0.42 3
0.9 0.62 2
```

Corresponding output

```
Data Set 1:
10

Data Set 2:
Impossible
```

Problem C: Craziness

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

Input File: `craziness.in`

Description

When you ask anyone about their family after a major gathering (e.g., for any kind of holidays), they will typically tell you how crazy their family is.[1] Somehow, years of knowing each other — and probably genetic similarity — mean that family members know exactly how to egg each other on or exasperate each other. Somehow, when put in the same room, a group of normal quiet people suddenly become insanely wild. Here, you will calculate which family members to invite to maximize the total craziness.

For each pair of family members i, j, you will be given a (positive or negative) number $c_{i,j}$, which is how much craziness this pair will contribute if both are invited; negative numbers mean that this pair will actually calm things down. The goal is to find the non-empty (i.e., at least one person must be invited) subset to invite to achieve maximum total craziness, which is the sum of crazinesses over all pairs of invited family members, plus the individual crazinesses of all guests.

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 the number $2 \leq n \leq 20$ of relatives to choose from. This is followed by n lines, each with n doubles $-1000.0 \leq c_{i,j} \leq 1000.0$. $c_{i,j}$ is the craziness added when you invite both i and j. We promise you that $c_{i,j} = c_{j,i}$ for all i and j. $c_{i,i}$ is the craziness that person i himself/herself contributes.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the maximum total craziness you can get by inviting a non-empty subset of these relatives, rounded to two decimals.

Sample Input/Output

Sample input `craziness.in`

```
1
5
1     -2.1 1.5  -10.3 4.7
-2.1  0    -8.1 2.3   5.0
1.5   -8.1 1.5  1.0   0.5
-10.3 2.3  1.0  -1    15.4
4.7   5.0  0.5  15.4  -2
```

Corresponding output

```
Data Set 1:
19.7
```

[1] If you asked salmon why they swim up rivers for weeks, they'd probably tell you: "I can't miss that party — my salmon family is crazy, man!!!"

Problem D: Siblings

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

Input File: `siblings.in`

Description

You probably know who your siblings are. After all, you usually grow up in the same environment as your siblings. But everyone has heard stories of siblings separated at a young age, who then found each other after many years. Finding siblings would of course be much easier if one were given complete information on the parents of every person in the world. Then, all it would take is a program to go through and find people who share a parent. Since there is some ambiguity with respect to half-siblings (sharing one parent, but not both), we will here focus only on women, and declare two women siblings (sisters) if they have the same mother. You will be given, for each woman, who her mother is. The goal is to compute how many pairs of sisters there are.

Input

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

The first line is the number $0 \leq n \leq 10,000,000$ of women in the data set. This is followed by n integers, distributed over one or more lines. The i^{th} integer gives you the index of the mother of i, which is some number $j < i$. If the mother is listed as number 0, this means we do not know i's mother; as far as your program is concerned, this means that i has no mother.

Output

For each data set, output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the number of pairs of sisters in the data set.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `siblings.in`

```
2
4
0 1 2 3
8
0 1 2 1 2
1 1 0
```

Corresponding output

```
Data Set 1:
0

Data Set 2:
7
```

Problem E: Family Portrait

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

Input File: `portrait.in`

Description

Earlier, we talked about crazy family gatherings. One of the craziest moments is usually when you try to take a family portrait. How are you getting the 2-year old to look at the camera? And why is there always at least one person who has to sneeze the exact moment you try to take a picture? Not to mention the difficulty in deciding who should stand where! Here, you will write a program to help place people in a picture. We are only considering pictures in which everyone stands in one line. The photographer has already decided on the order in which the women will stand, and on the order in which the men will stand. But the question is how to interleave the genders. There are two goals: spread out the genders evenly, and keep height differences between neighboring people small. Obviously, these two goals might be in conflict, so you have some computing to do.

You will be given for each of the w women the height h_i, in the order they are supposed to stand, and similarly for the m men. Sometimes, there will be more men, sometimes more women. Let's say that $w \geq m$ for this example. Then, the men define $m+1$ intervals, and on average, there should be $\frac{w}{m+1}$ women in each of these intervals. (Similarly with genders reversed if $m \geq w$.) Of course, $\frac{w}{m+1}$ might not be an integer, in which case you can choose which of the $m+1$ intervals contain $\lceil \frac{w}{m+1} \rceil$ women, and which contain $\lfloor \frac{w}{m+1} \rfloor$; this is entirely up to you.

Subject to this equi-spacing constraint, you should minimize the "height deviation", measured as follows. Suppose that the person standing in position j (from left to right) is p_j, and thus has height h_{p_j}. Then, the total height deviation is $\sum_{j=1}^{n-1}(h_{p_{j+1}} - h_{p_j})^2$.

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 contain two integers $1 \leq w, m \leq 500$. This is followed by one line with w integers between 0 and 1000, the heights of the w women, in the order they are supposed to stand. Next is a line with m integers between 0 and 1000, the heights of the m men, in the order they are supposed to stand.

Output

For each data set, output "`Data Set x:`" on a line by itself, where `x` is its number. Then output the minimum total height deviation you can achieve subject to the equi-spacing constraint.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `portrait.in`

```
1
3 6
150 165 180
152 155 157 159 163 170
```

Corresponding output

```
Data Set 1:
516
```

Problem F: Junior

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

Input File: `junior.in`

Description

In many cultures, children share their last name with at least one of their parents. In addition, in quite a few families, children will also be given a middle name to honor an ancestor, often a grandparent or parent. And in some cases, parents go even further, and name a child after their own first name. Most often, this is firstborn sons named after their father, and in that case, the word "junior" is often appended to the name to distinguish them. In this problem, you will use these naming conventions to determine how many families seem to be contained in a data set.

You will be given a list of names. Each name is a string containing only lowercase letters and whitespace (which separates parts of the name). Names consist of a first name (always), a middle name (optional), a last name (always), and one of the words "junior" or "iii" (optional). No one will ever have a first, middle, or last name of "junior" or "iii". You can infer that person A is the child of person B if (1) person A was born after person B, and (2) one of the following happens:

- Person A has the exact same name as person B, but with "junior" appended.
- Person A has the same name as person B, but with "junior" replaced with "iii". (That is, B's name ends in "junior", and A's in "iii", but apart from that replacement, their names are exactly the same.)
- A and B have the same last name, and the middle name of A is the first name of B, and A does not have any "junior" or "iii" appended to his name.

The number of families in your data set is the number of people for whom you cannot find a potential parent in the data set. (Notice that this means that if you have two "fitzgerald kennedy" and one "john fitzgerald kennedy", it doesn't matter that you cannot figure out which family he belongs with.)

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 single integer $1 \leq n \leq 1000$, the number of people in your data set. This is followed by n lines, each containing a string of length at most 80 characters, consisting only of lower-case letters and whitespace. The names will be given in increasing order of when the people were born.

Output

For each data set, first output "`Data Set x:`" on a line by itself, where `x` is its number. Then, output the number of families represented in the data set, i.e., the number of people whose parent is not known to be in the data set.

Each data set should be followed by a blank line.

Sample Input/Output

Sample input `junior.in`

```
1
10
thomas       jefferson
george  washington
george  washington    junior
richard   dwight  eisenhower
george washington     iii
thomas jefferson  iii
john fitzgerald   kennedy
robert   fitzgerald kennedy
dwight   david eisenhower
gerald  thomas jefferson
```

Corresponding output

```
Data Set 1:
7
```