

The “Midterm Elections” Programming Contest

USC Programming Contest, Spring 2006

04/22/2006

Sponsored by Language Weaver and VSoE

As most of you are aware¹, this year is midterm election year: many seats in the senate and house are up for reelection. If you have an opinion one way or the other — and we hope that you do — this is a good time to make it heard, and do your share participating in the democratic system. Staying informed and abreast of the political events of your time is as much part of being an educated citizen as anything you are learning here at USC.

And, wouldn't you know it, politics and elections lead to all kinds of intricate computational problems. From fundraising to gerrymandering to campaign speech design to vote counting, etc., interesting computational challenges abound. Being the talented programmers that you are, you will get to solve some of them. And who knows, if you do well, you may already have your job lined up when you graduate from USC.

When you submit a problem, you submit your source code file using PC². 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, persons, 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.]

¹and if you are not, then the many robocalls and spam mailings to come will certainly make sure you will be by the time

Problem A: Fundraising

File Name: money.cpp|money.java

Input File: money.in

Description

In order to even start running a campaign, you need some amount of money. How else are you going to pay for annoying robo-calls, slanderous TV ads against your opponent, or all those suits you will have to wear on the campaign trail? There are many different ways of raising funds, including old-fashioned donations, ads on the Internet, expensive fund-raising dinners, and others.

As campaign donations could be construed² as buying influence over a candidate, campaign finance laws place limits on the total amount of donations that any person can make to a candidate or party. According to the recent 2002 campaign finance law, any individual can contribute at most \$2100 to any one candidate, and at most \$40000 total. However, when these donations are divided over many transactions, it may be hard to figure out just how much someone donated.

You are to write a program that takes a sequence of individual donations, and finds out whether there were any violations, and if so, which donors violated the limits.

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 c, d, t with $1 \leq c \leq 100, 1 \leq d \leq 1000, 1 \leq t \leq 100000$. Here, c is the number of political candidates (numbered $1, \dots, c$), d the number of donors (numbered $1, \dots, d$), and t the number of transactions.

This is followed by t lines, each consisting of three integers d_i, c_i, m_i . This line means that transaction i , donor d_i gave m_i dollars to candidate c_i . Notice that the same donor-candidate pair can appear multiple times.

Output

For each data set, first output “Data Set x :” on a line by itself, where x is its number. Then, output either the line “No violations” on a line by itself, or “Violators:” on a line by itself, followed by all the donors who violated one of the limits, each on a line by himself. The donors are to be sorted by increasing number.

Sample Input/Output

Sample input money.in

```
2
3 2 4
1 1 1500
1 2 500
2 1 100
2 3 2000
3 2 4
1 1 1500
1 2 500
2 3 2500
1 2 1700
```

Corresponding output

```
Data Set 1:
No violations
Data Set 2:
Violators:
1
2
```

²unfortunately, frequently rightly so

Problem B: Gerrymandering

File Name: gerry.cpp|gerry.java

Input File: gerry.in

Description

Gerrymandering is the technique of assigning electoral districts so as to favor one party over another. Originally, voting districts were supposed to be reviewed on a once-a-decade basis so as to ensure roughly equal representation among voters. Nowadays, the party in power often seems to consider redistricting a useful tool for ensuring that it gets more seats in future elections. Of course, this tends to completely contradict the original intent, in that many voters are in fact disenfranchised by this redistricting. But hey, one does what one can for winning in an election, right?

Of course, in order to gerrymander well, one needs a computer program to produce the best voting districts. The fact that these districts don't tend to look all that "natural" won't matter. In our case, we assume that there are a number of voting precincts, which are to be divided into two districts. The program is to find out if a particular party can make sure to win both districts, one district, or no district. Of course, each voting district will have to contain at least one precinct, but the two district don't need to contain the same (or even nearly the same) number of precincts.

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 one number n , between 1 and 100, the number of precincts. This is followed by n lines, each describing one precinct i by giving two numbers p_i and q_i , the number of voters likely to vote for the two parties P and Q . These numbers are between 0 and 1000.

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 districts (0, 1, or 2) that party P can win with optimal gerrymandering.

Sample Input/Output

Sample input gerry.in

```
2
3
100 90
95 92
81 90
3
100 92
95 90
81 90
```

Corresponding output

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

Problem C: Stump Speech

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

Input File: `speech.in`

Description

It used to be that a candidate would simply state to potential voter what he thought, and what (s)he was going to do and vote for³. Nowadays, each statement and speech is carefully prepared and focus-group tested, to make sure that it does not elicit any negative reactions, or create any unwanted associations. Focus groups are literally sat down with levers in their hands, and while listening to the speech, are asked to trace with the lever how positive they feel about the candidate. Based on the feedback, the speech is then altered.

Wouldn't it be much easier if we could save ourselves the trouble of getting an actual focus group, and instead had a computer program that would tell us how people would react to a speech? Here, we are going to write a program that gives a first rudimentary approximation to this functionality. You will be given a list of key words and phrases, and the reactions they elicit. Based on those, you are to assess the quality of the speech.

More specifically, each phrase has a positive or negative score attached with it, for instance, "taxes" could have a score of -2, "increase taxes" a score of -4, "education" a score of 3, and "slash education" a score of -6. You would then count the number of times each of these phrases occurs in the text, and add the corresponding scores, to obtain the score of the speech.

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 a number n , the number of words or phrases eliciting emotions.

This is followed by n pairs of lines. The first line of each pair contains a key phrase of at most 80 characters, consisting entirely of lower-case letters and white space. The second line of each pair contains the score, a positive or negative integer.

This is followed by one more line of at most 10000 characters, containing the candidate's speech. This string consists only of lowercase letters, white space, and the punctuation signs ':' and ','. (In the example below, there are line breaks, in order to display the input properly. There won't be any line breaks as part of the text in the actual test files.)

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 proposed text on a line by itself. The score is the sum over all key phrases that occur in the text (if they occur multiple times, they are scored multiple times). Only exact matches count — if as much as the amount of white space is different, we assume that it is not the key phrase.

³perhaps fibbing a little — even the good old times were not *that* good.

Sample Input/Output

Sample input speech.in

```
1
4
taxes
-3
reduce taxes
5
education
3
slash education
-5
if you elect me, i promise to reduce    taxes and improve
education. my opponent, on the other hand, would slash education
and increase taxes.
```

Corresponding output

```
Data Set 1:
-5
```

Problem D: Campaign Stops

File Name: stops.cpp|stops.java

Input File: stops.in

Description

One of the important parts of running an election year campaign is to go lots of places, give your carefully crafted stump speech, and convince a lot of potential voters to vote for you. Unfortunately, there are only so many hours in the day, so you cannot go everywhere. So you need to plan your campaign trips carefully, trading off between travel time, time spent at the stops, and the number of voters you are going to sway. Better have a computer do the planning for you.

We assume that for each potential campaign stop, you are given the number of voters you expect to sway by appearing there, and the number of hours you would have to spend at the location. In addition, for each pair of stops, you are given the travel time from one to the other. If you also know the number of hours you have available, you can now optimize to sway as many voters 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 each data set contains two number n and H . Here, $1 \leq n \leq 10$ is the number of candidate campaign stops, and $1.0 \leq H \leq 24.0$ is the number of hours you have available (this may be a fractional number).

This is followed first by n lines, each describing a campaign stop, by giving two numbers, an integer $v_i \geq 0$, and a fractional number $h_i \geq 0$. v_i is the number of voters you could sway at that stop, and h_i the number of hours you'd have to spend there.

Finally, this is followed by n more lines, each containing n numbers, where the j^{th} number of line i is a fractional number, the number of hours it takes to travel from city i to city j . (Thus, the i^{th} number of line i is 0, but it is not necessarily the case that the travel time from city i to city j is the same as from city j to city 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 voters that can be swayed during the available time H . Assume that the candidate's tour always starts at city 1, and has to return there, although he may not have to campaign there.

Sample Input/Output

Sample input stops.in

```
1
4 13.5
100 3.5
100 1.0
300 2.0
140 5.0
0.0 1.0 4.0 1.5
1.0 0.0 5.0 0.5
5.0 5.0 0.0 5.5
2.0 0.7 6.0 0.0
```

Corresponding output

```
Data Set 1:
340
```

Problem E: Butterfly Ballots

File Name: ballots.cpp|ballots.java

Input File: ballots.in

Description

To “properly” improve one’s chances for winning an election, it is helpful to have allies on the election board. For instance, it can be quite useful to have the ballot designed in such a way that voters who are not paying close attention may actually vote for the wrong candidate. One tried and true way to achieve that is the so-called “Butterfly Ballot”. In a butterfly ballot, all the candidates’ names are on one side (say, the left), while all the boxes in which to mark a vote are on the other side. Now, if the boxes are a little bit shifted, it can be hard to tell which box corresponds to which candidate. For instance, look at the following ballot:

Which school do you like best?	
Berkeley	<input type="checkbox"/>
Stanford	<input type="checkbox"/>
USC	<input type="checkbox"/>
UCLA	<input type="checkbox"/>
CalTech	<input type="checkbox"/>
UCSD	<input type="checkbox"/>

You’ll have to admit that if one isn’t careful with this ballot, one might easily end up making a mark next to UCLA instead of USC. Of course, similar things could happen to political candidates.

You are to write a program to design a ballot that will make your candidate win, if possible. The assumption is that all candidates must be on the ballot, in some order from top to bottom that you can determine. The boxes will be on the other side, so that the first box is above the first candidate, the second box between the first and second candidate, and so forth. The assumption is then that among the voters who intend to vote for candidate i , half will actually vote for i , and half will accidentally vote for candidate $i + 1$. Of course, among the voters who intend to vote for the last candidate on the ballot, all will vote correctly. You are to decide if your candidate can be made to win the election. (If your candidate is tied for first place, we also consider that a win.)

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 $1 \leq n \leq 100$, the number of candidate on the ballot. Candidate 1 is the one you are trying to make win. This is followed by n lines, each containing a number v_i , the number of voters who intend to vote for candidate i . All the v_i will be even numbers, so you don’t need to worry about what happens about division by 2.

Output

For each data set, first output “Data Set x :" on a line by itself, where x is its number. If it is possible for candidate 1 to win, then output “Possible” on a line by itself, otherwise output “Impossible”.

Sample Input/Output

Sample input ballots.in

```
2
5
10
54
8
94
48
5
10
54
8
94
52
```

Corresponding output

```
Data Set 1:
Possible
Data Set 2:
Impossible
```

Problem F: And The Winner Is

File Name: winner.cpp|winner.java

Input File: winner.in

Description

Finally, we must count the votes for the candidates, and determine the winners of the elections. But we must be careful, as people sometimes fill out ballots incorrectly, and such ballots must be discarded. You are to write software that correctly determines the winners of the elections.

Why multiple elections? Usually, on one ballot, there are many different votes you make, for senator, congressperson, district positions, and many others. Here, we assume that if the vote for any one of those categories is incorrect, then the entire ballot is discarded.

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 n, r, v , the number of candidates (between 1 and 100), the number of races (between 1 and 9), and the number of voters (between 1 and 10000). This is followed by n lines, describing the candidates. Each of these lines first contains a number between 1 and 9, which describes the race that this candidate is running in. This is followed by a single space, and then the name of the candidate. For each race, there will be at least one candidate in it.

The next v lines each describe a voter's vote. Each line is a sequence of exactly n characters, each of which is either `x` or `#`. An `x` in position j means that voter i voted for candidate j , and an `#` in position j means that voter i did not vote for candidate j . If a voter voted for more than one candidate in the same race, the entire ballot is discarded. On the other hand, if a voter did not vote for any candidate in a particular race, that's ok.

Output

For each data set, first output "Data Set x :" on a line by itself, where x is its number. Then, output all the candidates who were in first place (or tied for first place) in their race, in the order in which they appeared in the input file.

Sample Input/Output

Sample input winner.in

```
1
5 2 5
1 Jay Bulworth
1 Hugh Waldron
2 USC
2 UCLA
2 CalTech
xx###
x#x##
####x
xx#x#
x####
```

Corresponding output

```
Data Set 1:
Jay Bulworth
USC
CalTech
```