

# The USC Programming Contest

USC Programming Contest, Spring 2005

04/16/2005

Sponsored by Electronic Arts and VSoE

This is the first year that USC has a real programming contest, and we're really excited about it. To honor the occasion, we themed the contest to introduce you to various exciting aspects of USC. You will quickly realize that wherever we look, interesting programming challenges arise. And guess what! You get to solve them!

When you submit a problem, you submit your source code file using PC<sup>2</sup>. 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: School Colors

File Name: colors.cpp|colors.java

Input File: colors.in

## Description

You all know what our school colors are, right?<sup>1</sup> They may not be quite the prettiest combination, but at least you can read one written on top of the other quite well. Much better than, say, Cardinal and Red, or Green and Turquoise. One would imagine that universities do make some effort to select contrasting colors. But did they get the most contrasting ones?

Here, we're going to write a program to find the most contrasting pair of colors among a given list. Each color is specified by a triple  $(R, G, B)$ , denoting the red, green, and blue values. Each of these values will be an integer between 0 and 255. The triple  $(0, 0, 0)$  is the color Black, and  $(255, 255, 255)$  is White. When we want to know the contrast between two colors, for our purposes here, we just treat it as the Euclidean distance between the corresponding triples, i.e., for triples  $(R, G, B)$  and  $(R', G', B')$ , the contrast is  $\sqrt{(R - R')^2 + (G - G')^2 + (B - B')^2}$ .

## 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  $n$  with  $2 \leq n \leq 200$ , the number of colors in the data set. This is followed by  $n$  lines, each containing three numbers  $R_i, G_i, B_i$ . These are the Red, Green, and Blue values of the  $i^{\text{th}}$  color in the list.

## Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output the pair of colors having largest possible contrast. If there are multiple pairs achieving largest contrast, output all of them, sorted by increasing first index, breaking ties by increasing second index.

## Sample Input/Output

Sample input colors.in

```
2
3
0 0 0
100 100 100
0 200 0
4
0 0 0
0 255 255
255 0 0
255 255 255
```

Corresponding output

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

---

<sup>1</sup>Hint: Cardinal and Gold

# Problem B: Lunch

**File Name:** lunch.cpp|lunch.java

**Input File:** lunch.in

## Description

USC has a bunch of quite yummy lunch options.<sup>2</sup> Sometimes, when you go for lunch, you want to meet your friends, and for that, there are some great meeting spots on campus, such as Tommy Trojan. But you have to carefully plan where you meet, depending on which restaurant you go to, to minimize the total distance you walk.

More precisely, you will be given a map of campus, with certain spots marked as meeting spots, and others as restaurants. In addition, you are given the starting points for yourself and the friends you are having lunch with. If you have selected a meeting point and a restaurant, this means that you and all your friends will be walking from your respective locations to the meeting point, then to the restaurant, and back to your respective starting locations. Naturally, you want to select the pair of meeting point and restaurant which minimizes the sum of walking distances for you and all your friends.

## 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  $h$  and  $w$ , the height and width of the campus map ( $1 \leq h, w \leq 30$ ). This is followed by  $h$  lines of  $w$  characters each. Each character is one of the following:

- **X** means that the square is inaccessible (building or fence or such).
- **.** means that the square is accessible.
- **R** means that the square contains a restaurant. You *cannot* walk through a restaurant. But you can enter it from (or leave it to) any unobstructed adjacent square.
- **M** means that the square is a meeting point. You *can* walk through meeting points.
- **S** means that the square is a starting point for you or one of your friends. You *can* walk through starting points.

Notice that you can only move up, down, left, or right on this map, not diagonally.

## Output

For each data set, first output “Data Set  $x$ :” on a line by itself, where  $x$  is its number. Then, output the total length traveled by you and all your friends together for the best pair of meeting point and restaurant. (You do not need to output the sequence.)

If there is no meeting point, or no restaurant, accessible from all starting points, then output “Impossible” instead of the total length.

## Sample Input/Output

See next page.

---

<sup>2</sup>If you haven't done so, check out Parkside Restaurant and Galen Athletic Center.

Sample input lunch.in

```
2
5 5
..X..
S.X.S
.MX.M
..X..
..R..
7 7
S.....
M...X..
...MX..
.XX.X..
..X.X..
.SX.X..
.R..XRS
```

Corresponding output

```
Data Set 1:
Impossible
Data Set 2:
62
```

# Problem C: Orange Bowl

**File Name:** football.cpp|football.java

**Input File:** football.in

## Description

It's late in the Orange Bowl Football game, and USC is trailing by 4 points, needing desperately to score another touchdown.<sup>3</sup> So coach Pete Carroll consults his newest weapon: a play strategy evaluator written at the USC programming contest.

While football is a somewhat more complicated game, we can simplify it as follows: USC is currently  $n$  yards away from the end zone ( $1 \leq n \leq 100$ ). The coach needs to decide on a sequence of plays to get the ball to the end zone as safely as possible. For each play, the coach can choose among  $m$  different plays ( $1 \leq m \leq 1000$ ). Each play is characterized by two numbers: its yard gain  $g$ , and its success probability  $p$ . Here  $g$  is an integer between 1 and 100, and  $p$  is a real number between 0 and 1. This means that the play succeeds with probability  $p$ , and if it does, it moves USC  $g$  yards closer to the end zone. If it fails, the ball is turned over, and USC loses the game.

The goal is to select a sequence of plays (which can have repetitions), such that the total yard gain is at least  $n$ , and the total success probability is maximized. We assume that the plays all succeed independently, i.e., the success probability of a sequence is the product of the individual probabilities.

## 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 the numbers  $n$  and  $m$ . This is followed by  $m$  lines, each containing the two numbers  $g_i$  and  $p_i$  for one play  $i$ .

## Output

For each data set, first output "Data Set  $x$ :" on a line by itself, where  $x$  is its number. Then, output on one line the overall success probability of the sequence of plays most likely to get USC to the end zone, rounded to two decimals. (You do not need to output the actual sequence.)

## Sample Input/Output

Sample input `football.in`

```
2
3 1
1 0.7
5 3
1 0.94
2 0.9
3 0.8
```

Corresponding output

```
Data Set 1:
0.34
Data Set 2:
0.76
```

---

<sup>3</sup>Yeah, right! Like USC is going to be trailing in football games any time soon.

# Problem D: Course Load

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

Input File: `courses.in`

## Description

USC offers a large number of fantastic courses that you may wish to take during your time here. There are classes on algorithms, stage make-up, chemical dependence and substance use, civil right litigation, the universe, yoga, leadership, and many more. But when you try to decide which classes to take, you always run into two problems:

- The most exciting classes always overlap with each other.
- There are only so many hours in the day to work and study.

If you want to be a bit more systematic about it, you could solve this as an optimization problem. For each class you are considering, you could assign a *utility*  $u$ , the benefit you would derive from taking this class (this benefit could be fun, or job skills, or whatever else). In addition, each class has a *workload*  $w$ , how many units of work per week are required to pass. (Notice that the workload may not correlate with the number of times the class meets per week.) Finally, you have the schedule for each class. You want to select a set of classes that don't overlap in their meeting times, and have a total workload not exceeding your work capacity  $C$ , maximizing your utility.

## 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 three numbers  $n$ ,  $m$  and  $C$ .  $1 \leq n \leq 20$  is the number of classes you are considering.  $1 \leq m \leq 100$  is the number of class meeting slots in the week (a meeting slot could for instance be "Monday from 3:30-4:50"). The number  $C$  with  $1 \leq C \leq 100$  is your capacity for class work.

This is followed by  $n$  lines, each describing one class. The first number on the line for class  $i$  is the (integer) utility  $u_i \geq 0$  of the class, the second one is the workload  $w_i \geq 0$ . The third number  $m_i$  is the number of times the class meets per week. This is followed (on the same line) by  $m_i$  numbers between 1 and  $m$ , each being one meeting time that the class occupies.

## Output

For each data set, first output "Data Set  $x$ :" on a line by itself, where  $x$  is its number. Then, output the maximum utility that can be achieved by any non-overlapping set of classes of total workload at most  $C$ . (You do not need to output *which* set of classes achieves this utility.)

## Sample Input/Output

Sample input `courses.in`

```
2
3 5 5
5 4 2 1 4
3 2 3 2 3 5
1 1 1 4
3 5 5
1 1 3 1 3 5
1 1 2 1 2
1 1 2 4 5
```

Corresponding output

```
Data Set 1:
5
Data Set 2:
2
```

# Problem E: Lecture Screens

File Name: screens.cpp|screens.java

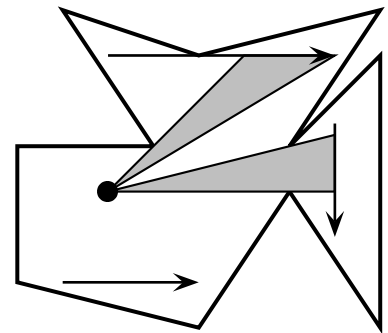
Input File: screens.in

## Description

Most of you have probably had lectures or seen talks in one of the classrooms (like OHE studios) which have multiple displays of the same screen, which is used for PowerPoint or other presentations. The reason is obvious: often enough, a pillar, or the head of the person in front of you, partially or completely obstructs the view of one screen, and you have a better chance of seeing everything if there are multiple copies. But just how much of the content does one see, depending on the seating position?

Here's how we can model this. We'll ignore people's heads, and just focus on architectural obstruction. The lecture hall is given to you as a closed and simple polygon (i.e., walls don't cross). In addition, you are given multiple positions of screens as lines in 2D, and your own position within the polygon. For each screen, you may see some or all of it from your position. Now, if you take the union of the contents you see, we want to calculate just how much content you get. For instance, if you see the left half of one screen and the right half of another, you see 100%. But if you see the left half of both, you only see 50% of the content. Also, if you see a screen from the back (meaning that its right side is to your left), then you can't see any of the contents.

In the picture on the right, the thick lines are the walls of the lecture hall. The arrows depict the screens, and point from the left to the right side. The gray shaded areas denote your view cone, and where they meet the arrow is the screen area you can see. Notice that the screen below the person is oriented the wrong way, so it is not visible. In the example on the right, you can see 90% (all but the leftmost 10%) of the screen. The input data corresponding to this example is given below.



## 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 four numbers  $n, m, x, y$ .  $3 \leq n \leq 100$  is the number of corner points of the polygon.  $1 \leq m \leq 20$  is the number of screens in the room, and  $(x, y)$  is your seating position ( $x$  and  $y$  are real numbers).

This is followed by  $n$  lines, each containing two real numbers  $(x_i, y_i)$ . These are the positions of the  $i^{\text{th}}$  corner point of the classroom. That means that there are walls from point  $(x_i, y_i)$  to  $(x_{i+1}, y_{i+1})$  for all  $i$ , as well as a wall from  $(x_n, y_n)$  to  $(x_1, y_1)$ .

Finally, there are  $m$  lines, each containing four real numbers  $(\ell_j^x, \ell_j^y), (r_j^x, r_j^y)$ , which are the  $x$ - and  $y$ -coordinates of the left and right endpoints of the  $j^{\text{th}}$  screen.

## 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 content that you can see, rounded to two decimals.

## Sample Input/Output

See next page.

Sample input screens.in

```
1
11 3 2 5
0 6
3 6
1 9
4 8
8 9
6 6
8 8
8 2
6 5
4 2
0 3
2 8 7 8
7 6.5 7 4
1 3 4 3
```

Corresponding output

```
Data Set 1:
90.00%
```

# Problem F: Campus Buildings

File Name: building.cpp|building.java

Input File: building.in

## Description

Most campuses have a lot of named buildings, and as the names tend to be rather long, the class schedules have abbreviations. At USC, it seems that the abbreviations have become used so widely that most people don't even remember the full name of the building. SAL, PHE, OHE. To a newcomer, this can be quite confusing. So perhaps, we ought to write a little program that finds out which building could be meant by abbreviations.

Here's how we'll do it: you will be given a list of building names, and a building abbreviation, such as SAL or FRSSC. The abbreviation matches the building name if all of its letters appear, in this order, in the building name (no letter can be matched twice). So, for instance, SAL matches "SALvatori", or "Student Aerospace Laboratory", or "univerSity of southern cALifornia". It does not match "angeles", as the letters are in the wrong order. For the comparison, we will ignore case, so 'S' and 's' are the same letter.

## 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 the number  $n$  of buildings,  $1 \leq n \leq 100$ . This is followed by  $n$  lines, each containing the name of a building, consisting of only uppercase and lowercase letters and white space. Finally, the building code will follow on a line by itself.

## Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output all of the building names that match the building code, each on its own line, and in the order in which they appeared in the initial list.

## Sample Input/Output

Sample input building.in

```
2
3
studENT aeRoSpace laBoratory
salVatori
angeles
SAL
2
SaLvatori
Science LibrarieS
SSL
```

Corresponding output

```
Data Set 1:
studENT aeRoSpace laBoratory
salVatori
Data Set 2:
```