

# The “100 Years of Engineering” Programming Contest

USC Programming Contest, Fall 2006

09/09/2006

Sponsored by VSoE, Electronic Arts, and LanguageWeaver

As most of you probably know, USC’s Viterbi School of Engineering is celebrating its 100<sup>th</sup> birthday this year.<sup>1</sup> In order to celebrate, the school already had several high-profile invited talks and other activities highlighting the school. In honor of the event, this semester’s contest will have a look around and see what kinds of problems various engineering departments are solving.

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<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, the problem set should contain 8 pages. If yours doesn’t, please contact one of the helpers immediately.]

---

<sup>1</sup>Also, USC is celebrating its 125<sup>th</sup> birthday.

# Problem A: Civil Engineering

**File Name:** `civil.cpp|civil.java`

**Input File:** `civil.in`

## Description

Civil Engineering is concerned with how to build things so that they don't collapse (whereas architects care more about how things look). Such "things" would include buildings in an earthquake-prone area like LA, or dams in a hurricane-prone area like Louisiana. Here, we will look at a simple problem related to building construction. Suppose that you want to put a building on top of an area, but have a limit on the total weight of your building. Evaluating whether a proposed construction exceeds that weight can be quite a chore. So we are asking you to write a program to take care of that chore for civil engineers.

The input will describe a collection of oblongs (i.e., rectangular 3-dimensional blocks) which are supposed to be assembled to give the building. Each oblong comes with information about its dimensions (height, width, depth), as well as the the material it is made from. In addition, you have, for each material, the density (weight per volume). You are supposed to find out the total weight of all oblongs.

## 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 numbers  $m, n$  between 1 and 10000.  $m$  is the number of materials in your database, and  $n$  is the number of oblongs in the building.

This is followed by  $m$  lines, each giving the density of one material (this is a non-negative integer number), in grams per cubic centimeter.

Next are  $n$  lines, each with four numbers  $h, w, d, i$ , where  $h, w, d$  are the height, width, and depth of the oblong (in centimeters), and  $i$  is the index of the material.  $h, w, d$  will be non-negative integers, and  $i$  an integer between 1 and  $m$ .

We will guarantee that for all inputs, the total weight will not exceed the maximum number you can store in a 32-bit integer.

## Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output the total weight of the oblongs in grams.

## Sample Input/Output

Sample input `civil.in`

```
1
3 4
2
4
10
30 50 49 3
10 50 49 1
25 50 2 3
25 2 49 3
```

Corresponding output

```
Data Set 1:
833500
```

# Problem B: Biomedical Engineering

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

**Input File:** biomed.in

## Description

Biomedical Engineering tries to use biological phenomena to engineer drugs or other substances at a very small scale. It is a very promising technology. In designing new drugs, an important aspect is to design them such that they will be able to attach to the right receptor cells in the body, where they cause or inhibit certain reactions. While the underlying biology can be very complex, we can look at the following very simplified view.

The docking site can be described as a string  $z$  with  $n$  letters, describing the chemical properties at the  $n$  consecutive locations. We want to assemble a matching string from given primitive components, so that it will attach at the site. In our simplified view, we have  $m$  strings  $y_i$ , where  $y_i$  consists of  $n_i$  letters. The goal is to select a sequence of these strings  $y_i$  such that their concatenation is exactly equal to  $z$ . We assume that we have an unlimited supply of the biological substances described by the  $y_i$ , so we can use arbitrarily many copies of the same  $y_i$  in our assembly.

## 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  $m$  of strings available for assembly, between 1 and 100. The second line is the string  $z$ , which will be between 1 and 1000 characters long, consisting only of uppercase and lowercase letters.

This is followed by  $m$  lines, each giving one string  $y_i$ . Each string  $y_i$  will be between 1 and 1000 characters long, consisting only of uppercase and lowercase letters.

## 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 segments which can be used to assemble  $z$ . If the assembly is impossible, then output "Impossible" instead.

## Sample Input/Output

Sample input biomed.in

```
2
4
acgaagaacga
acg
gaa
ac
cga
3
aaaaaaat
aaaaa
at
aaa
```

Corresponding output

```
Data Set 1:
4
Data Set 2:
Impossible
```

# Problem C: Electrical Engineering

File Name: ee.cpp|ee.java

Input File: ee.in

## Description

Electrical Engineers like to think about the constructions of electrical and electronic devices, and the communication infrastructure between them. Lately, a lot of focus has been on computing devices and their connections. In particular, a lot of electrical engineers care about wireless networks. Here, we are going to look at a simplified version of the problem of providing wireless coverage to an area.

The area will be described as a polygon, by a sequence of corners. It is surrounded by walls. In addition, we are given the location of one or more wireless routers. Our simplified model of wireless connectivity is as follows<sup>2</sup>. If there is a wall in the straight line from a router to a location, then the location cannot get any signal at all from that router, and the signal strength is 0. If there is no wall, then the strength of the signal is  $1/d^2$ , where  $d$  is the distance between the router and the location. If a location gets a signal from multiple routers, we only count the strongest signal. You are to find the signal strength for several locations.

## 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  $n, r, p$ , the number of corners of the polygon, the number of routers, and the number of points at which you are to evaluate signal strengths. All numbers will be between 1 and 100.

This is followed by  $n$  lines, each describing a corner, as a pair  $x, y$  of floating point numbers. Next are  $r$  lines, each describing a location of a router as a pair  $x, y$  of floating point numbers. Finally,  $p$  lines describe the point locations, as pairs  $x, y$  of floating point numbers.

We will make sure that (1) no routers or points lie exactly on a building wall, (2) no router and point are in the same location, and (3) no line from a router to a point just touches a wall (either it really intersects it, or completely avoids it).

## Output

For each data set, first output “Data Set  $x$ :” on a line by itself, where  $x$  is its number. Then, for each of the  $p$  points, on a line by itself, output the maximum signal strength at that point, rounded to two decimals.

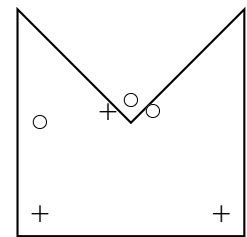
## Sample Input/Output

Sample input ee.in

```
1
5 3 3
0 0
1 0
1 1
0.5 0.5
0 1
0.1 0.1
0.9 0.1
0.4 0.55
0.1 0.5
0.6 0.55
0.5 0.6
```

Corresponding output

```
Data Set 1:
10.81
3.42
0.00
```



The example input  
+: router, o: location

<sup>2</sup>It ignores reflection of signals from walls, and penetration of walls.

# Problem D: Aerospace & Mechanical Engineering

File Name: ame.cpp|ame.java

Input File: ame.in

## Description

Aerospace Engineers and Mechanical Engineers like to build machines that do something exciting in the physical world. Aerospace engineers like building airplanes, rockets, and other stuff that flies. Not all of it is rocket science, though. Some questions are well within the grasp of programming contestants. For instance, once you have built a rocket launch it, you want to know how high it can fly before it lands.

Here is our model: the rocket consists of  $n$  stages, which are essentially fuel tanks. A stage lasts for a certain amount of time, and after it is empty, it is discarded. After the last fuel tank is discarded, the rocket cannot accelerate any more. As the rocket sheds its stages, it becomes lighter and lighter. We assume that the rocket itself weighs  $M > 0$  kilograms (kg), i.e., after shedding all its stages, the leftover weight is  $M$ .

In addition to  $M$ , you will be given, for each of the rocket's stages  $i$ , its weight  $m_i \geq 0$ , the duration for which it lasts  $t_i \geq 0$  (in seconds s), and the amount of thrust (force) it produces during that time  $F_i \geq 0$  (in  $\frac{\text{kg}\cdot\text{m}}{\text{s}^2}$ ), where m stands for meters. We will pretend that the rocket fuel itself does not weigh anything. We assume that the rocket is shot straight up, and ignore all effects of wind, friction, leaving the Earth's gravity field, etc. We also assume that the initial speed of the rocket is 0 m/s, until the first stage starts burning.

The important physical formulas you will need are: (1)  $a = F/m$ , i.e., the acceleration of an object of mass  $m$  under force  $F$  is  $a = F/m$ . (2) In the Earth's gravity field, any object accelerates toward the Earth at a constant acceleration of  $g = 9.81\frac{\text{m}}{\text{s}^2}$ . (3) If an object starts out with velocity  $v$ , and accelerates at acceleration  $a$ , then in  $t$  seconds, it travels distance  $vt + \frac{1}{2}at^2$ , and its new speed after those  $t$  seconds is  $v + at$ .

Assume that fuel tanks are used (and discarded) in the order  $1, 2, \dots, n$ . You are to compute the height at which the rocket was right when the last stage fell off. Our inputs will always be such that the rocket will lift off, and not crash into the Earth before the last stage 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 each data set contains the number of stages  $n$  (between 1 and 30), and the weight  $M$  of the rocket itself.  $M$  is a floating point number strictly greater than 0.

This is followed by  $n$  lines, each consisting of three non-negative floating point numbers  $m_i, t_i, F_i$ , describing a stage. So the initial weight of the rocket is  $M + \sum_i m_i$ .

## Output

For each data set, first output "Data Set x:" on a line by itself, where x is its number. Then, output the height at which the rocket was when the last stage fell off, rounded to two decimals.

## Sample Input/Output

Sample input ame.in

```
1
3 1000.0
325.0 120.0 18000.0
75.0 60.0 5100.0
200.0 120.0 15000.0
```

Corresponding output

```
Data Set 1:
8550.0
```

# Problem E: Industrial & Systems Engineering

File Name: ise.cpp|ise.java

Input File: ise.in

## Description

A lot of what industrial engineering does is plan and improve industrial processes to be more efficient. As such, they solve a lot of optimization problems in product planning, supply management, etc. Some of these are fairly complex, but a natural and clean version of one of the central topics can be understood quite easily.

In the “Warehouse Location Planning Problem”, you have several stores which need a supply of wares. In order to provide them with the wares, you can build several warehouses. For each warehouse, there is a price to build it (depending on where it is located). On the other hand, if a store is far from its assigned warehouse, it is expensive to ship all the goods to it. So the warehouse locations need to trade off between building costs and shipping costs. The goal is then to build warehouses to minimize the cost.

We assume that if the locations of a warehouse and a store are given, the shipping cost is exactly equal to their Euclidian distance. We also assume that any warehouse, once built, can supply arbitrarily many stores. Notice that you must always build at least one warehouse. The total cost is then the sum of the building costs, plus the sum of shipping costs for all stores from their assigned warehouse.

## 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 numbers  $n, m$ , the number of stores, and the number of possible warehouse locations under consideration. The number of stores will be between 1 and 100, and the number of warehouse locations between 1 and 20.

This is followed by  $n$  lines, each describing a store, by giving its  $x$  and  $y$  coordinates (floating point numbers).

Next are  $m$  lines, each describing a warehouse location and price as floating point numbers  $x, y, p$ . Here,  $p$  will be non-negative.

## Output

For each data set, first output “Data Set  $x$ :” on a line by itself, where  $x$  is its number. Then, output the minimum total cost at which all stores can be supplied, rounded to two decimals.

## Sample Input/Output

Sample input ise.in

```
1
4 4
0.1 0.1
0.0 0.9
1.0 0.05
1.1 -0.1
-0.1 -0.1 0.8
0 1.1 0.5
0.7 0 0.3
0.5 0 0.3
```

Corresponding output

```
Data Set 1:
2.32
```

# Problem F: Computer Science

**File Name:** `csci.cpp|csci.java`

**Input File:** `csci.in`

## Description

Computer Science studies ways of solving real-world problems using computers and related technology. This encompasses a lot of issues. One of the great success stories of recent years in computer science is organizing and searching information. Just think about the way everyone's life has changed as a result of web search. Here, we are going to explore a very simple version of a web search engine.

You will be given several documents, consisting of words and hyperlinks (in a simpler format than HTML, though). You will also be given queries (one word each), and are to find out the most relevant search results. A page  $P$  is relevant either because (1) the search term appears in the page, or (2) the search term appears in a page pointing to  $P$ , close to the hyperlink<sup>3</sup>. Specifically, if  $q$  is the query term, then  $P$  gets one point for each occurrence of  $q$  in  $P$ . In addition, if  $L$  is a hyperlink from  $P'$  to  $P$ , then  $P$  gets  $\max(4 - d, 0)$  points for each occurrence of  $q$  at distance  $d$  words of  $L$  (that is, the score is  $4 - d$  if  $d < 4$ , and 0 otherwise). If  $q$  appears in the hyperlink, then it is worth 4 points, as  $d = 0$ . Notice that the same occurrence in  $P'$  can count multiple times if it is close to multiple hyperlinks to  $P$ .

## 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 numbers  $m$  and  $n$ , the number of queries, and the number of web pages. Both numbers will be between 1 and 100.

Next are  $m$  queries. Each query will be a single word of no more than 20 letters, on a line by itself. All words will consist only of lowercase letters.

This is followed by the description of  $n$  web pages. The description of web page  $i$  begins with the number  $\ell_i$  of lines that page  $i$  consists of (between 1 and 100). This is followed by  $\ell_i$  lines of text, each at most 255 characters. Text is a sequence of words, separated by spaces. Each word consists only of lowercase letters, and is at most 20 characters long. Some words can be marked as hyperlinks. This is done by enclosing them in square brackets, with the number of the page that the link goes to. For example, `[usc,3]` would describe a hyperlink to page 3 displaying the word `usc`. The number will always be between 1 and  $n$ , i.e., a legal page. Also, we will never have self-links.

## Output

For each data set, first output "Data Set  $x$ :" on a line by itself, where  $x$  is its number. Then, for each of the queries, output the page with the highest score on a line by itself. If there are multiple pages with the same highest score, output all of them on one line (separated by spaces), sorted by increasing page number.

---

<sup>3</sup>The latter is actually more important. Often, what people linking to you say about you is a better description than what you yourself say. One reason is that web pages can otherwise use spamming to attract additional visits. Another is that web pages often don't contain all relevant terms.

## Sample Input/Output

Sample input csci.in

```
2
3 3
usc
great
sucks
2
i am a student at [usc,2] a great school
i also think that [ucla,3] sucks
3
we are usc the university of southern california
we are located in the same town as [ucla,3]
we have many excellent [students,1]
2
we are ucla
we are a great great school
1 2
usc
1
empty page
2
[link,1] usc usc [usc,1] text text text text text
usc usc usc usc usc usc usc usc usc usc usc usc
```

Corresponding output

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