Programming Design, Spring 2015
Lab Exam 3

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Submission and grading. In this exam, there are three problems. You need to write a C++ program for each problem. 100% of your grades will be based on the correctness of your outputs. The online grading system will input in total 65 sets of testing data and then check your outputs. These 65 sets count for 130 points, i.e., 2 points for each set. Your official grades will be 100 if you get more than that. Before the due time of the exam, you may upload your programs multiple times. Only the last one you upload will be graded. Unlike what happens with your homework, you will not see the grading results during the exam. For each problem, you will only see the results for the online samples.

To submit your work, please upload the three .cpp files, one for each problem, to the online grading system at https://pdogs.ntu.im/judge/. Just a reminder: Talking to any other person online, directly or indirectly, is definitely considered cheating.

Problem 1

(40 points) Given a set of pairs of n values $\{(x_i, y_i)\}_{i=1,...,n}$, the (sample) covariance is defined as

$$\sigma_{xy} = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{n-1},$$

where $\mu_x = \frac{\sum_{i=1}^{n} x_i}{n}$ and $\mu_y = \frac{\sum_{i=1}^{n} y_i}{n}$ are the means of x_is and y_is, respectively. The (sample) correlation coefficient is then defined as

$$r = \frac{\sigma_{xy}}{\sigma_x \sigma_y},$$

where

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu_x)^2}{n-1}} \quad \text{and} \quad \sigma_y = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \mu_y)^2}{n-1}}$$

are the standard deviations of x_is and y_is, respectively. For example, suppose that we have $n = 5$ pairs of values $(0, 0)$, $(1, 5)$, $(2, 6)$, $(4, 4)$, and $(5, 5)$, we have $\mu_x = 2.4$, $\mu_y = 4$, $\sigma_x \approx 2.0736$, $\sigma_y \approx 2.3452$, $\sigma_{xy} = 2.5$, and $r \approx 0.5141$. It can be shown that $-1 \leq r \leq 1$.

In the field of Statistics, two sets of data are said to be positively correlated, negatively correlated, or not correlated if the correlation coefficient r is positive, negative, or zero. They are strongly correlated if $|r| \geq 0.7$, moderately correlated if $0.4 \leq |r| < 0.7$, or weakly correlated if $0 < |r| < 0.4$. In the previous example, the two sets of data have a moderately positive correlation. In this problem, you will need to categorize given sets of data based on their correlations.

Input/output formats

The input consists of 20 files. At the beginning of each file, there is a positive integer n as the number of pairs of values. In the second line, there are n values x_1, x_2, ..., x_n. In the third line, there are n values y_1, y_2, ..., y_n. Two consecutive values are separated by a white space. At the end of each line, there is a newline character. Your task is to calculate the correlation coefficient and then print out (1) the maximum number in x_is and y_is and (2) the degree of correlation. The second part should be printed out based on the following table:

strongly	moderately	weakly	zero	weakly	moderately	strongly
negative				positive		
−3	−2	−1	0	1	2	3

As an example, below is an input file:

```
5
0 1 2 4 5
0 5 6 4 5
```

Your program should read each input file and then print out two values

```
6 2
```

in a single line, where `6` means that the maximum number is 6 and `2` means a moderately positive correlation. Please note that the category should be determined without rounding up or down r. For example, if $r = 0.6999$, it belongs to category 2, not 3.

Problem 2

(50 points) A card game (that does not exist in the real world) runs in the following way. Each player gets 5 cards, one with an integer between 1 and 10 on it. These cards then generate credits to the player if the following situations happen:

- A *straight* is a hand that contains five cards in sequence, such as 1, 2, 3, 4, 5 or 3, 4, 5, 6, 7. Note that 10 and 1 are not considered as consecutive. Therefore, 7, 8, 9, 10, 1 or 8, 9, 10, 1, 2 do not count as straights, and there are in total 6 possible straights.
- A *five of a kind* is a hand that contains five cards of the same rank, such as 1, 1, 1, 1, 1 or 6, 6, 6, 6, 6.
- A *four of a kind* is a hand that contains four cards of the same rank and the last one different, such as 1, 1, 1, 1, 4 or 6, 6, 6, 6, 4.
- An *all odds* is a hand that contains only 1, 3, 5, 7, and 9.

A straight, a five of a kind, a four of a kind, and an all odds gives the player 100, 80, 60, and 40 credits, respectively. Then each card gives the player a number of credits equal to its rank. Below are some examples:

- If one gets 1, 1, 1, 1, and 1, she gets 80 for a five of a kind, 40 for all odds, and 5 for ranks on the cards. In total she gets 125.
- If one gets 8, 8, 8, 8, and 8, she gets 80 for a five of a kind and 40 for ranks on the cards. In total she gets 120.
- If one gets 5, 6, 7, 8, and 9, she gets 100 for a straight and 35 for ranks on the cards. In total she gets 135.
- If one gets 7, 8, 9, 10, and 1, she gets 35 for ranks on the cards. In total she gets 35.

Given a set of five cards on hand, you will need to find the number of credits earned by this set. Given multiple sets of cards, you will need to find the number of credits earned by the winner, i.e., the one with the most credits.

Input/output formats

The input consists of 25 files. In each file, there are $n + 1$ lines of integers:

- In the first line, there is a positive integer n. It is given that $1 \leq n \leq 100$. There is a newline character at the end of this line.
- In each of the ith line, $i = 2, ..., n + 1$, there are 5 integers $x_{i,1}$, $x_{i,2}$, ..., and $x_{i,5}$. They are the ranks of the cards got by the $(i - 1)$th player. It is given that $1 \leq x_i \leq 10$. These five values are separated by four white spaces. There is a newline character at the end of each line.

Your program should read each input file and then print out two integers: The credits earned by the winner(s) and the number of winner(s), separated by a white space. For example, suppose we have

```
3
1 1 1 1 1
3 4 5 6 7
7 8 9 10 1
```

as a file of input, the output should be

```
125 2
```

in a single file. The winners are players 1 and 2, each getting 125 credits.

Problem 3

(40 points) In a typical train schedule, there are multiple trains running among multiple stations. Each train passes through multiple stations. For each train, the scheduled arrival time at each station that it passes is recorded. To make this problem easier, let's assume that the arrival time and departure time at each station are the same. Moreover, let's assume that all stations lie on a single chain (like Taiwan High Speed Rail) rather than a network (like Taipei Metro Rapid Transit). As an example, suppose there are stations A, B, C, D, E, F, and G (from north to south), a train schedule may look like

```
Train Station Time
1     A       1215
1     B       1235
1     D       1305
1     F       1320
1     G       1330
2     G       1220
2     F       1240
2     E       1305
3     B       1240
3     G       1315
```

where the column `Train` records train ID, `Station` records station ID, and `Time` records the arrival time of that train at that station (where `1215` means 12:15, etc.). Note that a train may skip some stations. In the above example, train 1 stops at stations A, B, D, F, and G while train 3 only stops at stations B and G. In other words, train 3 passes (at least) stations D and F.

In this problem, you will be given a traveler's information, including the origin station, destination station, and preferred departure time horizon within two time points. Your task is to count the number of direct trains that meet the criterion, and among these trains, find the one whose departure time at the origin station is closest to the starting time point of the preferred time horizon. To make this problem easier, do not consider connecting multiple trains; consider direct trains only.

As an example, suppose one wants to travel from station B to station G and depart from station B within 12:30 and 13:00, you should tell the traveler that there are 2 feasible trains (trains 1 and 3), and train 1's departure time at station B is the closest to 12:30. If the traveler changes the time horizon to 12:40 to 13:10, you will respond that only 1 train is available, which is train 3. Finally, if the time horizon is shifted to 12:50 to 13:20, the answer would be that no train is available.

Input/output formats

The input consists of 20 files. In each file, there are $n + 2$ lines of information:

- In the first line, there is a string `Train Station Time`. The three words are separated by two white spaces. There is a newline character at the end of this line.
- In each of the ith line, $i = 2, 3, ..., n + 1$, there are three values. First is an integer as the train ID, second is a capital English letter as the station ID, and last is a string of four digits of numbers as the departure time. There are at most 100 trains and 26 stations. All trains run in the same day (i.e., no train departs from a station today but arrive at another station tomorrow). No train passes one station twice. A train either goes from north to south or from south to north; no train goes back and forth. Two consecutive values are separated by a white space. There is a newline character at the end of each line.
- In the last line, there are four values. First is a capital English letter as the origin station, second is a capital English letter as the destination station, third is a string of four digits as the starting time point of the preferred horizon, and last is a string of four digits as the ending time point of the preferred horizon. Two consecutive values are separated by a white space. There is a newline character at the end of each line.

Your program should read each input file and output (1) the number of direct trains that departs from the origin station within the preferred time horizon and goes to the destination station, and (2) among all these trains, the train ID of the one whose departure time at the origin station is closest to the starting time point of the preferred time horizon. If there is a tie in the second part, find the one with the smaller train ID. As an example, suppose you are given the example schedule as above with

```
B G 1230 1300
```

as the last line, the output should be

```
2 1
```

in a single file. Here `2` is the number of trains that meet the criterion and `1` is the ID of the winning train. If no train meets the criterion, print out a single integer `0`.