2004 APICS Programming Contest

The 2004 Atlantic Canada programming Contest was held at University New Brunswick St John Friday Oct 15 2004. There were 22 teams of three students from 10 universities in the four Atlantic provinces of Canada.

Each team of 3 students was given 6 problems to try to solve in 5 hours using one computer following the rules for the ACM programming contests.

a. Optimization - Todd Wareham
b. Spam Filter - Martin van Bommel
c. Minimum Spanning Tree - Arthur Sedgwick
d. Bipartite matching - Patricia Evans
e. Pay up - Arthur Sedgwick
f. Set Intersection - Jalal Almhana, Arthur Sedgwick

The participating teams came from the following universities:

```
Acadia University, Wolfville N.S.
Dalhousie University, Halifax N.S.
Memorial University of Newfoundland, St. John's.
Mount Allison University, Sackville N.B.
St Marys University, Halifax  N.S.
St Francis Xavier University, Antigonish N.S.
University of Prince Edward Island, Charlottetown.
University of New Brunswick, Fredericton Campus.
University of New Brunswick, St John Campus.
Universite de Moncton, Moncton N.B.
```

The top 3 teams from this competition advanced to the next level of competition in Rochester N.Y.

Problem A: Corporate Cashier Follies

Consider the following problem that has recently come up at the McCrafty Fast Food Corporation: An efficiency consultant on loan from ENRON has analyzed McCrafty restaurant operations worldwide and noted the following three universal factors when cashiers give customers change from a purchase:

1. Cashiers like to give back as few coins as possible;
2. Customers are willing to accept a lower amount than their actual change if it keeps a cashier happy; and
3. The relative strengths of preferences (1) and (2) vary from season to season and from country to country, *e.g.*, some cashiers are lazier than others and some customers are stingier than others.

Given a set $C = \{c_1, c_2, \ldots, c_n\}$ of n coin-values and the amount amt owed a customer, the consultant has proposed the function

$$f(ret, i_a, i_b) = \begin{cases} (i_a \times minCoins(C, ret)) + (i_b \times (amt - ret)) & \text{if minCoins(C, ret)} > 0 \\ (i_b \times amt) & \text{otherwise} \end{cases}$$

where ret is the amount returned by the cashier, i_a and i_b are cashier and customer irritation factors, respectively, and $minCoins(C, x)$ is the minimum number of coins with values from the set C whose values sum to x (note that $minCoins(C, x) = 0$ if there is no set of coins with values from C whose values sum to x). For example, given $C = \{2, 5, 25\}$ and $amt = 49$,

- $f(45, 1, 1) = (1 \times 5) + (1 \times (49 - 45)) = 9$;
- $f(47, 1, 1) = (1 \times 6) + (1 \times (49 - 47)) = 8$;
- $f(45, 3, 1) = (3 \times 5) + (1 \times (49 - 45)) = 19$; and
- $f(47, 3, 1) = (3 \times 6) + (1 \times (49 - 47)) = 20$.

The consultant conjectures that if the cashier always returns change equal to the value of ret that minimizes $f()$, cashier and customer satisfaction will be optimized and the McCrafty Corporation will also have access to a previously underutilized source of revenue.

Given C, amt, i_a, and i_b, compute a value ret that minimizes function $f()$ specified above. The input will consist of a $(3 \times i)$-line file, $i \geq 1$, such that in each group of 3 lines, line 1 gives the number n of coins in C, line 2 contains the n values of the coins in C sorted in ascending order, and line 3 gives the values of amt, i_a, and i_b, respectively. You may assume that all input files are formatted correctly.

Sample input (available as file "A.in"):

```
3
2 5 25
9 1 1
3
2 5 25
9 2 1
3
2 5 25
9 5 1
3
2 5 25
9 2 2
```

Sample output (available as file "A.out"):

```
Amount returned on 9 cents is 9 cents [3 coin(s) / F-val = 3]
Amount returned on 9 cents is 5 cents [1 coin(s) / F-val = 6]
Amount returned on 9 cents is 0 cents [0 coin(s) / F-val = 9]
Amount returned on 9 cents is 9 cents [3 coin(s) / F-val = 6]
```

Problem B Filter it OUt

Part of the task of most spam filters is text recognition; that is, searching for key words and phrases in the body of e-mail messages. This task would be very simple, except spammers keep coming up with new methods to trick the filters. In response, filters have become more sophisticated, but new methods can lead to more false positives - identifying mail as spam that is not.

Consider the key word "viagra". Analysis of spam indicates that this word appears

1. in plain text, sometimes in mixed upper- lower-case, but entirely on one line,
2. in text with the "i" replaced by another character or one or more HTML tags but not just missing,
3. in text with spaces or HTML tags separating any of the letters.

For this problem an HTML tag is "<...>" on one line with "..." meaning one or more characters excluding ">".
Note that only the "i" can be replaced, all other letters must appear.
You can assume that all HTML is properly formatted; that is, the appearance of a "<" to start a tag always has a corresponding ">". You may also end up with false hits, as indicated by the last line in the example, where "V A Grant" is counted.

Input: The first line consists of a number n of lines to follow. The rest of the input is the lines of e-mail, including plain text and HTML, with a maximum of 80 characters per line.

Output: For each line, a number indicating the number of occurrences of variations of the word "viagra" on the line.

Sample Input:

```
8
Try our amazing new drug Viagra.  Order viagra from our online pharmacy.
Our Vlagra is cheaper that others.
<H>Order <font size=+2>V</font>i<font size=-1>a</font>gr<font size=+1>a</font> from us.
You just can't go wrong with <strong>V</strong><img src="eye.jpg"><strong>agra</strong>.
<A HREF="via">Grain and coal are the major cargo of CN Rail.
A vagrant would rarely use viagra, even though viagra might help.
Send your order to V.A. Grant and Associates.
V A Grant himself would never use v!agra.
```

Sample Output:

```
2
1
1
1
0
2
0
2
```

Our contest software this year requires all data sets to be in one file (B.in for this problem). You should have a copy of B.in in the directory with your program. However your program should take input from System.in/stdin/cin. This will be connected to B.in when you submit.

Terror on in-line skates

If you take up inline skating you may find going downhill rather terrifying. When you move to a new town it is helpful to calculate routes that avoid steep hills. For this problem we consider towns that are laid out on a rectangular grid with streets running north-south and east-west. Sample input for two towns is shown below (5 lines).

```
acabab
bcdefg
dcbabb

a
```

The first town has 3 e-w streets and 6 n-s streets. There are 18 intersections. The elevation of each intersection is the numerical value of the character used to label it. For example, intersection g is 5 units higher than the intersections labelled b immediately north and south of g (indicating steep hills skaters would try to avoid).

The steepness of a hill between two adjacent intersections labelled x and y is defined as |x-y|.

When you move to a new town you want to plan routes which allow you to get from any intersection to any other avoiding steep hills where possible. In the sample output shown below (7 lines) there is a ' ' or '*' between each pair of intersections. A '*' indicates that you will not use the direct route between those two intersections. There should be as many '*'s as possible and the total steepness avoided (at the *s) should be as large as possible.

```
a*c a b a b
    * * * *
b c d e f g
*   * * * *
d c b a b b

a
```

From this we see that to get from g to b immediately south of it we would travel west to c and then turn south and then go east. This route avoids any hills with a steepness greater than 1.

The second town has only one intersection so the output has no ' ' or '*' for it.

The input used to test your program will consist of one or more town descriptions separated by blank lines. Each town description has one n >= 1 lines each containing the same number of characters m >= 1 followed by a '\n'. m will be at most 26. The output for that town will have 2n-1 lines each with 2m-1 characters and a '\n'. On the "in-between" lines the ' ' or '*' are separated by ' 's.

The output has the same as the input but with a ' ' or '*' between each pair of adjacent intersections in each town.

Note that there may be more than one correct output for a town.

The sample input shown above is available for testing as file C.in,
however, your program should read from stdin/System.in/cin.
File C.in needs to be copied to the directory with your solution.
Use it for testing using input redirection and when you submit your
program the system will run your program with System.in/stdin/cin
connected to the judge's version of the file.

(Postscript: The judges had difficulty with non-unique solutions and
request that the total steepness avoided and number of *s
be part of the output.)

Problem D: Sampling the Bon Accord

A thirsty programmer has stumbled across a place that sells a variety of fine refreshments. Being inclined to experiment, she would like to sample as many different kinds as she can, but will only try one per night. Planning is made difficult by the fact that the change in varieties is not regular, but is instead dictated by what is available.

The place has T spots for refreshment. When a spot becomes vacant, which only happens when its current type runs out, it is filled with the next available type from a queue. Types can be repeated.

After some research, our programmer has come up with some additional information. First, each type of refreshment lasts exactly for a positive integer number of days, which will vary. Most importantly, she has discovered the list of the queue of types, in the order in which they will be used, together with how many days each will last. Note that a type may not always last for the same number of days.

She is willing to visit this place on as many days as are required, but will only sample once on each day. She needs your help (since this obsession is compromising her coding skills) to work out the largest number of different samples that she can try. Note that you only need to tell her how many; getting a maximum sampling is up to her.

Input Description:

The first line will contain a single number K that will indicate the number of different test cases to be computed. Blank lines may also separate the test cases.

Each of the K test cases will contain the following:

The first line of the case will have four numbers: T, the number of different spots that can be available on each day; D, the total number of days for which this case will run; B, the total number of different types of refreshment; and Q, the length of the queue. What follows will be a list of Q pairs, each giving a type (numbered from 0 to B-1) followed by how many days it will last.

Note that types can be reused (if they get more) but may not last for the same duration as before (if a different amount is available, or its popularity changes). The spots start empty, so the first T types in the queue are put into use immediately. Slots can be vacant at the end of the time period if there is nothing left in the queue to replace them.

Your output should consist of, for each case, a statement that says how many different samples are possible. Number your cases from 1.

```
Sample Input:
2
3 6 5 8
0 3
1 2
2 1
3 6
4 4
2 2
0 1
1 3

2 6 5 6
1 3
0 2
2 3
0 2
3 2
4 1


Consider this last case to illustrate how the spots are filled over
its 6 days, as follows:

day 0: type 1, type 0
day 1: type 1, type 0
day 2: type 1, type 2
day 3: type 0, type 2
day 4: type 0, type 2
day 5: type 3, type 4

We have enough of type 3 for one more day, but the sampling period is over.

Sample Output:
Case 1: 5 different samples are possible.
Case 2: 4 different samples are possible.


Our contest software this year requires all data sets to be in one
file (D.in for this problem). You should put a copy of D.in in the
directory with your program.  However your program should take input
from System.in/stdin/cin. This will be connected to D.in when you
submit.
```

Postscript: this is a matching problem which can be viewed as a network flow problem.

Problem E Pay Up

A group of students and coaches went to St John for a contest. During the trip various members paid bills that should have been shared by other members. At the end of the trip a settlement needs to be arranged. Shown below is sample data file E.in (7 lines).

```
 400 A C E C
 200 B D F
-200 G E A

-4 A Dept
 5 A B C A B
-6 B Dept
```

The input will contain one or more data sets separated by empty lines. There are 2 data sets in this example. Each data set has one or more lines. Each line has an amount followed by the name of the person who paid it and one or more names of others who should have shared the expense. In the first line above A paid 400 but C and E shared the benefit. Since C's name occurs twice C is responsible for two fourths of the 400 and A and E one fourth each. When the amount is negative as in the third line that means the person paid a positive amount but is not responsible for a share. The following names are. In the third line G paid 200. E and A both owe 100 for that one and G is owed 200.

When we calculate each share of an expense we round to the nearest dollar. Thus for the second line B, D and F each owe one third of 200 which we round to 67. Because of rounding, after all settlements are made, some may end up with a small amount too much or too litle.

The output for the above example should be (8 lines):

```
C pays A 200
E pays G 200
D pays B 67
F pays B 66

C pays A 1
Dept pays A 6
Dept pays B 4
```

In what follows an "ower" is someone who still owes and an "owee" is someone who is still owed.

The format of the output consists of one settlement for each data set with an empty line between settlements. Each settlement has two stages each of which produces zero or more lines of output. In the first stage, in the interest of reducing the number of payments, owers seek out owees who are owed exactly the same amount and pay them this amount. This explains the first two lines of output. After reading the first data set we find that C and E both owe 200 and A and G are both owed 200. When there is a choice about who pays whom, as there is in this example, the alphabetically first ower pays the alphabetically first owee. In the second stage of a settlement the alphabetically first remaining ower pays the alphabetically first remaining owee. The amount paid is the lessor of what the ower still owes and the owee is still owed. This process continues until either there are no more owers or no more owees. Consider again the first data set. After the first stage the owers are D and F who both owe 67 and the remaining owee is B (owed 200-67=133). So D pays B 67 and then B is still owed 66 which is what F pays B. After this F still owes (1) but there are no more owees.

For the second data set the first stage produces no lines of output because no one is owed the same amount as someone else owes.

File E.in needs to be copied to the directory with your solution. Use it for testing using input redirection and when you submit your program the system will run your program with System.in/stdin/cin connected to the judge's version of the file.

Problem F: Beam alignment

A certain company makes steel beams which have small holes predrilled at regular intervals in the center of the beam through which things like electrical wires can go. In new beams these holes have covers but the covers are easily knocked off when necessary at a construction site.

Another company has some used beams each of which has one or more of the hole covers already knocked off. We want to align the beams so that as many holes as possible line up.

Our contest software this year requires all data sets to be in one file (F.in for this problem). You should have a copy of F.in in the directory with your program. However, your program should take input from System.in/stdin/cin. This will be connected to F.in for you.
The public F.in has 8 lines:

```
1 2  4 6
2 3  6 9
3 6  7 11
4 7 11 12

1 3 6 9
2 4 7 10 15
3 5 8 11
```

There are two data sets here separated by a blank line. The judges have different data. It consists of one or more data sets separated by blank lines. Each data set has one or more lines. Each line has one or more integers indicating which hole covers have been knocked off for one beam. These integers are non-negative and in ascending order.

With the second data set above we see that by shifting the first beam over two units and the second beam over one unit we can get 4 holes to line up. With the first data set the best we can do is have 2 holes line up in all 4 beams (by shifting the first two beams over 5 places).

The output should contain one integer for each data set indicating the maximum number of holes which can be made to line up. For the data above the output should be (on one line):
2 4
Extra blanks will be accepted.
Note programs are not allowed to take more than 120 seconds to execute.