UW Madison's

2006 ACM-ICPC Individual Placement Test

October 1, 12:00-5:00pm, 1350 CS

Overview:

This test consists of seven problems, which will be referred to by the following names (respective of order):

rsp, tree, sub, pals, bet, ferry, tug.

Recall that we have sorted these problems with respect to what we believe to be their relative difficulty, so it would be a good strategy (but by no means required) to start with problem rsp and work your way through consecutively. Also, you may have seen some of these problems before, but read them carefully, as some details **may** have changed.

Input/Output:

Your programs should take input from standard in (i.e., the keyboard) and output to standard out (i.e., the terminal). As is standard practice for the ICPC, you may assume that the input strictly follows the description in the problems. It is your responsibility to ensure that your output **precisely** matches that described in the problems, or else risk your program being rejected with a "Presentation Error".

Problem Submission:

To submit a program, send e-mail to mwa+icpc@cs.wisc.edu and attach your source code. The subject line should contain only the problem name, **prob,** that you are submitting, and the attached source code should be named **prob**.{c|cpp|java}. For example, if you are using C++ and submitting the problem "rsp", the file should be named "rsp.cpp". You will receive the results of your submission as soon as possible via e-mail.

Clarifications:

As in the ICPC, you may submit clarification requests as well. They should be sent to mwa+icpc@cs.wisc.edu, with a subject of "Clarification-**prob**", where **prob** is the name of the problem you wish to be clarified. Replace **prob** with "general" if there is an issue with the contest as a whole. Accepted clarification requests will be answered to all those taking the test via e-mail. Experience of previous years learns that most clarification requests are rejected, and receive a simple response such as "Read the problem description".

Printing:

You may print to the printer at any time during the test.

Written Solutions:

We encourage you to spend the last half hour of the test to write down the main idea behind the solution to any of the problems for which you have not had a program accepted. Please be concise, using at most a few sentences within the space provided on the opposite side of this page. We will take these partial solutions into account along with your official ranking when composing teams, although they will have less weight.

After the Test:

The proctor will announce when time is up. Please stop working at this time and take a moment to fill out the form on the back of this sheet and turn it in to the proctor (you may keep the problems). You are invited to join us for pizza and soda after the test in 1325 CS.

Information Form:

Name:

CS Login:

Student status (e.g., Junior, first year grad student):

Year of birth:_____ , Year starting college:_____

What programming languages do you prefer?

What do you feel are your strengths with respect to the ICPC?

****Use the space below to briefly write down the basic idea behind the solution to any problems for which you did not get a program accepted.****
RSP:

Tree:

Sub:

Pals:

Bet:

Ferry:

Tug:

Rock, Scissors, Paper

Bart's sister Lisa has created a new civilization on a two-dimensional grid. At the outset each grid location may be occupied by one of three life forms: *Rocks*, *Scissors*, or *Papers*. Each day, differing life forms occupying horizontally or vertically adjacent grid locations wage war. In each war, Rocks always defeat Scissors, Scissors always defeat Papers, and Papers always defeat Rocks. At the end of the day, the victor expands its territory to include the loser's grid position. The loser vacates the position.

Your job is to determine the territory occupied by each life form after *n* days. The first line of input contains *t*, the number of test cases. Each test case begins with three integers not greater than 100: *r* and *c*, the number of rows and columns in the grid, and *n*. The grid is represented by the *r* lines that follow, each with *c* characters. Each character in the grid is R, S, or P, indicating that it is occupied by Rocks, Scissors, or Papers respectively.

For each test case, print the grid as it appears at the end of the *n*th day. Leave an empty line between the output for successive test cases.

Sample Input

```
2
3 3 1
RRR
RSR
RRR
3 4 2
RSPR
SPRS
PRSP
```

Output for Sample Input

```
RRR
RRR
RRR

RRRS
RRSP
RSPR
```

The Tree Movers

Given two binary search trees, A and B, with nodes identified by (that is, having keys equal to) positive, non-zero integers, and the use of commands ``delete *K* '' and ``add *K* '' (defined below), what is the smallest number of commands that can be used to transform tree A into tree B?

Recall that in a binary search tree, the keys of all nodes in the left subtree of a node with key *K* must be less than *K* . Similarly, the keys of all nodes in the right subtree of a node with key *K* must be greater than *K* . There are no duplicate nodes.

The ``delete *K* '' command will delete the tree (or subtree) with its root at the node with the key *K* . Deleting the root of the entire tree leaves an empty tree. The ``add *K* '' command will add a new node identified by the integer *K* . This node will naturally be a leaf node.

Since we seek to transform tree A into tree B, it follows that commands will be applied only to tree A; tree B is ``read only".

It is easy to see that it should never require more than $N + 1$ commands to achieve the transformation of A into B, since deletion of the root node of tree A followed by the addition of one node for each of the *N* nodes in B (in the proper order) will achieve the desired goal. Equally easy to determine is the minimum number of commands required: if A and B are identical, then zero commands are required.

Input

There will be multiple input cases. For each case, the input contains the description of tree A followed by the description of tree B. Each tree description consists of an integer *N* that specifies the number of nodes in the tree, following by the keys of the *N* nodes in an order such that *N* ``add" commands would create the tree. The last case is followed by the integer `-1'. No node will have a key larger than 10^9 , and *N* will be no larger than 100.

Output

For each case, display a single line containing the input case number (1, 2,...) and the number of commands required to transform tree A into tree B, formatted as shown in the examples below.

Sample Input

```
4 5 2 7 4 6 5 3 7 1 4 9
0 0
1 100 0
0 1 100
3 100 49 37 2 200 152
-1
```

Sample Output

```
Case 1: 5 commands.
```

```
Case 2: 0 commands.
Case 3: 1 command.
Case 4: 1 command.
Case 5: 3 commands.
```

Subway

You have just moved from a quiet Madison neighbourhood to a big, noisy city. Instead of getting to ride your bike to school every day, you now get to walk and take the subway. Because you don't want to be late for class, you want to know how long it will take you to get to school.

You walk at a speed of 10 km/h. The subway travels at 40 km/h. Assume that you are lucky, and whenever you arrive at a subway station, a train is there that you can board immediately. You may get on and off the subway any number of times, and you may switch between different subway lines if you wish. All subway lines go in both directions.

The input begins with an integer representing the number of cases to solve. Each case consists of the x,y coordinates of your home and your school, followed by the number s of subway lines in the city. The descriptions of these s lines follow- each description consists of the non-negative integer x,y coordinates of each stop on the line, in order. You may assume the subway runs in a straight line between adjacent stops, and the coordinates represent an integral number of metres. Each line has at least two stops. The end of each subway line is followed by the dummy coordinate pair -1,-1. In total there are at most 200 subway stops in the city.

For each test case, output the case number followed by the number of minutes it will take you to get to school in that case, rounded to the nearest minute, taking the fastest route. Print the answer for each test case on a separate line.

Sample Input

```
1
0 0 10000 1000 2
0 200 5000 200 7000 200 -1 -1
2000 600 5000 600 10000 600 -1 -1
```

Sample Output

```
Case 1: 21 minutes
```

Making Pals

A palindrome is a sequence that is the same when read forward or backward. For example, "pop" is a palindrome, as are ``Poor Dan is in a droop" (ignoring spaces and case), and ``12321".

In this problem, you are to find the ``cheapest" way to transform a sequence of decimal digits into a palindrome. There are only two types of modifications you may make to the sequence, but each of these may be repeated as many times as necessary. You may delete a digit from either end of the sequence, or you may add a digit to either end of the sequence. Each of these operations incurs a ``cost" of 1. For each input sequence, determine the smallest cost of transforming the sequence into a palindrome, and the length of the resulting palindrome. If two palindromes can be produced with the same cost, the length of the longer palindrome (the one with more digits) is to be reported.

For example, suppose the initial sequence was ``911". This can be transformed into a palindrome by deleting the leading ``9" (yielding ``11") or by adding an additional ``9" to the right end of the sequence (yielding ``9119"). Since both of these transformations have a cost of 1, and the second transformation yields a longer palindrome, it is this one which would be reported as your result.

Note that the particular palindrome produced by the cheapest sequence of transformations is not necessarily unique, but since you are not required to report the resulting palindrome, any of these will suffice.

Input

There will be multiple cases to consider. Each case has a single line of input that contains one or more decimal digits followed by the end of line. The maximum number of digits in a sequence will be **2000**. The last case is followed by an empty line (that is, only an end of line).

Output

For each input case, display the case number (1, 2,...), the cost of the cheapest transformation, and the length of the resulting palindrome. Your output should follow the format shown in the examples below.

Sample Input

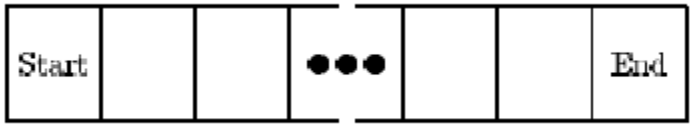
```
911
9118
11234
          <-- This line is blank
```

Sample Output

```
Case 1, cost = 1, length = 4
Case 2, cost = 2, length = 4
Case 3, cost = 3, length = 8
```

To Bet or Not To Bet

Alexander Charles McMillan loves to gamble, and during his last trip to the casino he ran across a new game. It is played on a linear sequence of squares as shown below.



A chip is initially placed on the Start square. The player then tries to move the chip to the End square through a series of turns, at which point the game ends. In each turn a coin is flipped: if the coin is heads the chip is moved one square to the right and if the coin is tails the chip is moved two squares to the right (unless the chip is one square away from the End square, in which case it just moves to the End square). At that point, any instruction on the square the coin lands on must be followed. Each instruction is one of the following:

1. Move right n squares (where n is some positive integer)

2. Move left n squares (where n is some positive integer)

3. Lose a turn

4. No instruction

After following the instruction, the turn ends and a new one begins. Note that the chip only follows the instruction on the square it lands on after the coin flip. If, for example, the chip lands on a square that instructs it to move 3 spaces to the left, the move is made, but the instruction on the resulting square is ignored and the turn ends. Gambling for this game proceeds as follows: given a board layout and an integer T, you must wager whether or not you think the game will end within T turns.

After losing his shirt and several other articles of clothing, Alexander has decided he needs professional help-not in beating his gambling addiction, but in writing a program to help decide how to bet in this game.

Input

Input will consist of multiple problem instances. The first line will consist of an integer n indicating the number of problem instances. Each instance will consist of two lines: the first will contain two integers m and T ($1 <= m <= 50$, $1 <= T <= 40$), where m is the size of the board excluding the Start and End squares, and T is the target number of turns. The next line will contain instructions for each of the m interior squares on the board. Instructions for the squares will be separated by a single space, and a square instruction will be one of the following: +n, -n, L or 0 (the digit zero). The first indicates a right move of n squares, the second a left move of n squares, the third a lose-a-turn square, and the fourth indicates no instruction for the square. No right or left move will ever move you off the board.

Output

Output for each problem instance will consist of one line, either

Bet for. x.xxxx

if you think that there is a greater than 50% chance that the game will end in T or fewer turns, or

Bet against. x.xxxx

if you think there is a less than 50% chance that the game will end in T or fewer turns, or

Push. 0.5000

otherwise, where x.xxxx is the probability of the game ending in T or fewer turns rounded to 4 decimal places. (Note that due to rounding the calculated probability for display, a probability of 0.5000 may appear after the Bet for. or Bet against. message.)

Sample Input

```
5
4 4
0 0 0 0
3 3
0 -1 L
3 4
0 -1 L
3 5
0 -1 L
10 20
+1 0 0 -1 L L 0 +3 -7 0
```

Sample Output

```
Bet for. 0.9375
Bet against. 0.0000
Push. 0.5000
Bet for. 0.7500
Bet for. 0.8954
```

Ferry Loading

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

There is a ferry across the river that can take *n* cars across the river in *t* minutes and return in *t* minutes. *m* cars arrive at the ferry terminal by a given schedule. What is the earliest time that all the cars can be transported across the river? What is the minimum number of trips that the operator must make to deliver all cars by that time?

The first line of input contains *c*, the number of test cases. Each test case begins with *n, t, m*. *m* lines follow, each giving the arrival time for a car (in minutes since the beginning of the day). The operator can run the ferry whenever he or she wishes, but can take only the cars that have arrived up to that time. For each test case, output a single line with two integers: the time, in minutes since the beginning of the day, when the last car is delivered to the other side of the river, and the minimum number of trips made by the ferry to carry the cars within that time.

You may assume that 0 < n, t, m < 1440. The arrival times for each test case are in non-decreasing order.

Sample input

```
2
2 10 10
0
10
20
30
40
50
60
70
80
90
2 10 3
10
30
40
```

Output for sample input

```
100 5
50 2
```

Tug of War

A tug of war is to be arranged at the local office picnic. For the tug of war, the picnickers must be divided into two teams. Each person must be on one team or the other; the number of people on the two teams must not differ by more than 1; the total weight of the people on each team should be as nearly equal as possible.

The first line of input contains the number of test cases *t*. The description of each of the *t* test cases follows. Each begins with *n* the number of people at that picnic. *n* lines follow- the first line gives the weight of person 1; the second the weight of person 2; and so on. Each weight is an integer between 1 and 450. There are at most 100 people at the picnic.

Your output will be a single line for each test case containing the case number followed by 2 numbers separated by a comma: the total weight of the people on one team, and the total weight of the people on the other team for that case. If these numbers differ, give the lesser first.

Sample Input

```
1
3
100
90
200
```

Output for Sample Input

```
Case 1: 190, 200
```