Alberta Collegiate Programming Contest

October 1st, 2005 – 11:00 to 16:00

ACPC 2005

Problem A: Dropping Water Balloons
Problem B: Throwing Cards Away I
Problem C: Land Surveyor's Job
Problem D: Blackbeard the Pirate
Problem E: Flea Circus
Problem F: Ants, Aphids and a Ladybug
Problem G: Throwing Cards Away II
Problem H: Words Adjustment
Problem I: Can of Beans

Main Site:
http://ugweb.cs.ualberta.ca/~acpc/

Live Contest Page:
http://pages.cpsc.ucalgary.ca/acmicpc/acpclive/

Prizes and funding for this contest is provided by the Alberta Informatics Circle of Research Excellence.

Problem A: Dropping Water Balloons

It's frosh week, and this year your friends have decided that they would initiate the new computer science students by dropping water balloons on them. They've filled up a large crate of identical water balloons, ready for the event. But as fate would have it, the balloons turned out to be rather tough, and can be dropped from a height of several stories without bursting!

So your friends have sought you out for help. They plan to drop the balloons from a tall building on campus, but would like to spend as little effort as possible hauling their balloons up the stairs, so they would like to know the lowest floor from which they can drop the balloons so that they do burst.

You know the building has n floors, and your friends have given you k identical balloons which you may use (and break) during your trials to find their answer. Since you are also lazy, you would like to determine the minimum number of trials you must conduct in order to determine with absolute certainty the lowest floor from which you can drop a balloon so that it bursts (or in the worst case, that the balloons will not burst even when dropped from the top floor). A trial consists of dropping a balloon from a certain floor. If a balloon fails to burst for a trial, you can fetch it and use it again for another trial.

The input consists of a number of test cases, one case per line. The data for one test case consists of two numbers k and n, $1 \le k \le 100$ and a positive n that fits into a 64 bit integer (yes, it's a *very* tall building). The last case has $k=0$ and should not be processed.

For each case of the input, print one line of output giving the minimum number of trials needed to solve the problem. If more than 63 trials are needed then print `More than 63 trials needed.` instead of the number.

Sample input

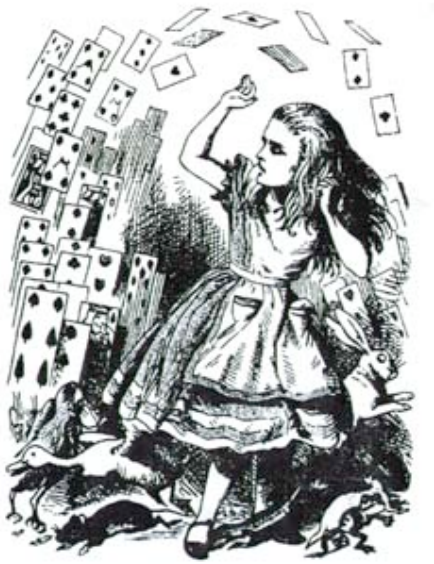
```
2 100
10 786599
4 786599
60 1844674407370955161
63 9223372036854775807
0 0
```

Output for sample input

```
14
21
More than 63 trials needed.
61
63
```

Piotr Rudnicki

Problem B: Throwing Cards Away I



Given is an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. The following operation is performed as long as there are at least two cards in the deck:

> Throw away the top card and move the card that is now on the top of the deck to the bottom of the deck.

Your task is to find the sequence of discarded cards and the last, remaining card.

Each line of input (except the last) contains a number $n \le 50$. The last line contains 0 and this line should not be processed. For each number from the input produce two lines of output. The first line presents the sequence of discarded cards, the second line reports the last remaining card. See the sample for the expected format.

Sample input

```
7
19
10
6
0
```

Output for sample input

```
Discarded cards: 1, 3, 5, 7, 4, 2
Remaining card: 6
Discarded cards: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 8, 12, 16, 2, 10, 18, 14
Remaining card: 6
Discarded cards: 1, 3, 5, 7, 9, 2, 6, 10, 8
Remaining card: 4
Discarded cards: 1, 3, 5, 2, 6
Remaining card: 4
```

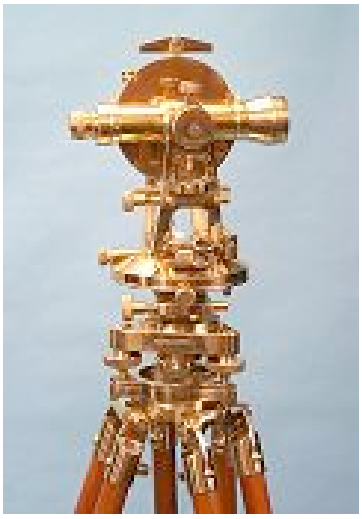
Folklore, adapted by Piotr Rudnicki

Problem C: Land Surveyor's Job

The job of a land surveyor has been substantially changed with the arrival of GPS and satellite imaging. However, the traditional tools of a land surveyor still are a theodolite, for measuring angles, and a measuring tape for measuring distances. Although in real land surveying we are concerned with three dimensions, for this problem we will restrict ourselves to a plane.

The backbone of any land survey is a traverse consisting of a sequence of n stations numbered 1,2, ..., n. Stations are chosen by some criteria that do not concern us here. For each station i two measurements are taken:

- the length of the forward shot, i.e. the distance from station i to station $i+1$, in case of station n this is the distance to station 1,
- the angle between the back shot and the fore shot measured clockwise; this is the angle from the segment $(i,i-1)$ to the segment $(i,i+1)$ measured clockwise. In case of station 1, this angle is entered as 0 since the n-th station is not known in advance. For station n this angle is the angle between the segments $(n,n-1)$ and $(n,1)$. For all stations other than station 1 this angle is non-zero.



For a given traverse, when locations of stations i and $i+1$ are available we can compute the location of station $i+2$. In consequence, given a traverse and actual locations of stations 1 and 2 we can calculate the locations of all other stations. However, because of the inaccuracy of theodolites and measuring tapes, the results of our calculations are approximate. Therefore, after computing locations of stations $n-1$ and n we can compute the location of station 1 and compare it to the assumed actual location. We say that a traverse is acceptable if the distance between the actual location of station 1 and its location as calculated from the traverse is less than 0.1% of the total length of the traverse.

The input consists of a number of cases. The data of each case appears on a number of input lines, the first of which contains a non-negative integer n giving the number of the stations in a traverse, $3 \le n \le 1000$. The following n lines contain the distance and angle measured for each station in increasing order of their numbers. Except for station 1, all angles are specified in degrees, minutes and seconds in the format shown in the sample input. Note that the angle for station 1 is shown as 0.

Input is terminated by a line with n equal to 0.

For each case of input, output one line stating whether the traverse of this case was `Acceptable` or `Not acceptable`.

Sample input

```
4
1       0
1       90d00'00"
0.999   90d00'00"
1       90d00'00"
4
10      0
10      270d00'00"
9.95    270d00'00"
10      270d00'00"
0
```

Output for sample input

```
Acceptable
Not acceptable
```

H. James Hoover, Piotr Rudnicki

Problem D: Blackbeard the Pirate

Blackbeard the Pirate has stashed up to 10 treasures on a tropical island, and now he wishes to retrieve them. He is being chased by several authorities however, and so would like to retrieve his treasure as quickly as possible. Blackbeard is no fool; when he hid the treasures, he carefully drew a map of the island which contains the position of each treasure and positions of all obstacles and hostile natives that are present on the island.

Given a map of an island and the point where he comes ashore, help Blackbeard determine the least amount of time necessary for him to collect his treasure.

Input consists of a number of test cases. The first line of each test case contains two integers *h* and *w* giving the height and width of the map, respectively, in miles. For simplicity, each map is divided into grid points that are a mile square. The next *h* lines contain *w* characters, each describing one square on the map. Each point on the map is one of the following:

- `@` The landing point where Blackbeard comes ashore.
- `~` Water. Blackbeard cannot travel over water while on the island.
- `#` A large group of palm trees; these are too dense for Blackbeard to travel through.
- `.` Sand, which he can easily travel over.
- `*` A camp of angry natives. Blackbeard must stay at least one square away or risk being captured by them which will terminate his quest. Note, this is one square in any of eight directions, including diagonals.
- `!` A treasure. Blackbeard is a stubborn pirate and will not leave unless he collects all of them.

Blackbeard can only travel in the four cardinal directions; that is, he cannot travel diagonally. Blackbeard travels at a nice slow pace of one mile (or square) per hour, but he sure can dig fast, because digging up a treasure incurs no time penalty whatsoever.

The maximum dimension of the map is 50 by 50. The input ends with a case where both *h* and *w* are 0. This case should not be processed.

For each test case, simply print the least number of hours Blackbeard needs to collect all his treasure and return to the landing point. If it is impossible to reach all the treasures, print out -1.

Sample Input

```
7 7
~~~~~~~
~#!###~
~...#.~
~~....~
~~~.@~~
.~~~~~~
...~~~.
10 10
```

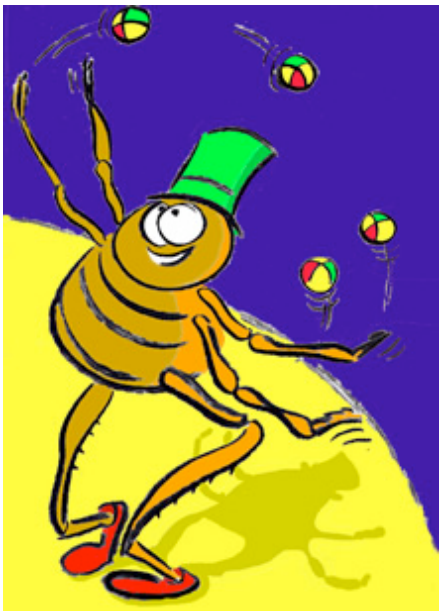
```
~~~~~~~~~~
~~!!!###~~
~##...###~
~#....*##~
~#!..**~~~
~~....~~~~
~~~....~~~
~~..~..@~~
~#!.~~~~~~
~~~~~~~~~~
0 0
```

Output for sample input

```
10
32
```

Broderick Arneson

Problem E: Flea Circus



Sometimes, if you are searching for ladybugs, all you find are tree fleas. An old tree near the anthill serves as a home of two fleas who sometimes steal honeydew from aphids living on the tree. On the old tree, branches connect the forks (spaces where two or more branches meet) and leaves where the fleas can rest. Fleas are so small that the ants cannot see them and thus fleas are safe. Because of their size, the fleas satiate their appetite pretty quickly and then have a lot of energy to jump. They decide to jump toward each other complying with the following rules:

- There is exactly one way for the fleas to go from one leaf or fork in the tree to another leaf or fork without ever turning back. Each of the fleas starts jumping along such a route to the current location of the other flea.
- The fleas can only jump from one fork or leaf of the tree to another fork or leaf if they are connected by a branch.
- If the two fleas land at the same time in the same place then they decide to sit and chat for quite a while.
- If the two fleas land at the same time in two neighboring places on the tree (forks or leaves) then they keep jumping forever.

The input file contains multiple test cases. Each test case starts with an integer n, the number of leaves and forks in the tree, $1 \le n \le 5000$. We assume that leaves and forks are numbered from 1 to n. Each of the following n-1 lines describe tree branches; each branch is described by two integers a and b, meaning that the branch connects the fork or leaf labeled a and the fork or leaf labeled b. In the $(n+1)$-st line there is one integer l, $1 \le l \le 500$, saying how many starting positions of the fleas you are to consider for the tree. Each of the following l lines contains two positive integers (not greater than n). These numbers define the tree places in which the fleas start their jumping. Input is terminated by the case with n equal to 0.

Your program should output l lines for each test case. The i-th line for a case should look like

`The fleas meet at` *p*.

where p identifies the place where the fleas meet, or

`The fleas jump forever between` *p* `and` *r*.

where p and r are two neighboring places on the tree with $p \le r$

Sample input

```
8
1 2
1 3
2 4
```

```
2 5
3 6
3 7
5 8
5
5 1
7 4
1 8
4 7
7 8
0
```

Output for sample input

```
The fleas meet at 2.
The fleas meet at 1.
The fleas jump forever between 2 and 5.
The fleas meet at 1.
The fleas jump forever between 1 and 2.
```

Piotr Rudnicki

Problem F: Ants, Aphids and a Ladybug

If you are searching for ladybugs, the best place to look is near ants. The ants suck sweet honeydew from aphids and therefore, where there are ants, there must be aphids. Where there are aphids, there are more likely to be ladybugs. Ants defend aphids against ladybugs --- for which aphids are a much sought for delicacy.

An old tree near the anthill serves as an aphid farm. On the old tree, branches connect the forks (spaces where two or more branches meet) and leaves where the aphids feed. There are k ant-guards (numbered from 1 to k) working on the farm. A ladybug hunting aphids lands on the places where aphids feed, i.e. on leaves or forks. When a ladybug lands on the tree, the guard-ants set off in her direction in order to chase her away. They comply with the following rules:

- There is exactly one way for an ant to go from one fork or leaf in the tree to another fork or leaf without ever turning back. Upon the ladybug's landing, all of the ant-guards simultaneously start going along such a route to the place where the ladybug landed.
- If there is an ant in the place of the ladybug's landing, the ladybug takes off immediately.
- If at any time an ant rushing towards the ladybug spots another ant anywhere ahead on its route, the ant stops and remains in its current position, yielding to the ant that is farther ahead.
- If two or more ants try to reach the same fork of the tree at the same time, only one does so -- the one with the smallest ID number. The other ants trying to reach the same fork remain in their places.
- An ant which gets to the place of ladybug's landing chases it away and remains in this place.

The ladybug is stubborn and lands on the tree again, and then the ant-guards set off again trying to chase the intruder away. In order to simplify the problem we assume that for each ant it takes a unit of time to walk between two neighboring forks or a fork and a leaf (2 leaves cannot be neighbors).

The input file contains multiple test cases. Each test case starts with an integer n, the number of leaves and forks in the tree, $1<=n<=5000$. We assume that leaves and forks are numbered from 1 to n. Each of the following n-1 lines describes tree branches; each branch is described by two integers a and b, meaning that the branch connects forks or leaves labeled a and b. In the $(n+1)$-st line there is one integer k, $1<=k<=1000$ and $k<=n$; k is the number of ants that guard the tree. There is one positive integer (not greater than n) in each of the following k lines. The integer written in the $(n+1+i)$-th line is the start position of the i-th ant. There is no fork or leaf in the tree occupied by more than one ant. The line $n+k+2$ contains one integer l, $1<=l<=500$, l saying how many times a ladybug lands on the tree. Each of the following l lines contains one positive integer (not greater than n). These numbers define a sequence of tree places in which the ladybug lands. Input is terminated by the case with n equal to 0.

Your program should output *k* lines for each test case. In the *i*-th line for a case there should be two integers separated by a single space --- the final position of the *i*-th ant (number of a fork or a leaf) and the number of times the *i*-th ant chased the ladybug away.

Sample input

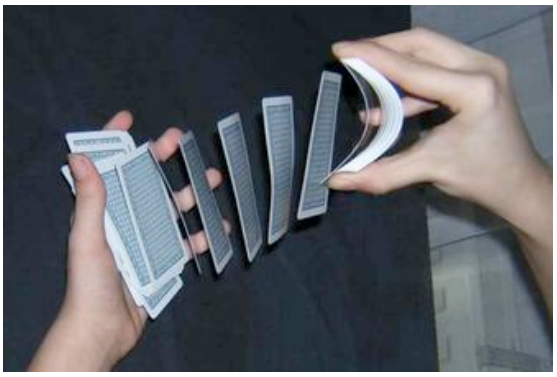
```
8
1 2
1 3
2 4
2 5
3 6
3 7
5 8
3
1
2
6
5
2
4
5
8
7
0
```

Output for sample input

```
5 2
2 2
7 1
```

Krzysztof Lorys, adapted by P. Rudnicki

Problem G: Throwing Cards Away II



Given is an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. The following operation is performed as long as there are at least two cards in the deck:

> Throw away the top card and move the card that is now on the top of the deck to the bottom of the deck.

Your task is to find the last, remaining card.

Each line of input (except the last) contains a positive number $n \leq 500000$. The last line contains 0 and this line should not be processed. For each number from input produce one line of output giving the last remaining card.

Input will not contain more than 500000 lines.

Sample input

```
7
19
10
6
0
```

Output for sample input

```
6
6
4
4
```

Folklore, adapted by Piotr Rudnicki

Problem H: Words Adjustment

In this problem, you are given two words x and y, and a finite sequence of words $\{ w_1, w_2, ... w_k \}$. If it is possible to obtain the same word by appending to x and y some words from the given sequence of words, we say that x and y can be *adjusted*. We would like to check whether the words x and y can be adjusted using the words from the given sequence.

Given a word w, we can perform an operation $w*w_i$, 1<=i<=k, consisting in appending the word w_i to the word w at the right. We define this as an append operation. The task is to find the smallest number of append operations that are necessary to adjust two given words using the words from a given sequence.

For example, words `abba` and `ab` can be adjusted by the words from the sequence { `baaabad`, `aa`, `badccaa`, `cc` }. It suffices to append to `abba` two words: `aa` and `badccaa`, and to `ab` three words: `baaabad`, `cc` and `aa`. In both cases we obtain: `abbaaabadccaa`.

The first line of input contains the number of cases that follow. The first two lines of data for each case contain the words x and y, respectively. The third line contains the integer k, 0<=k<=1000, which is the length of the sequence of words that can be used for word adjustment. The following k lines contain one word each. All words use only lowercase letters and contain between 1 and 1000 characters.

For each case output one nonnegative integer giving the minimal number of operations that are needed to adjust two given words, or output `-1` if it is impossible.

Sample input

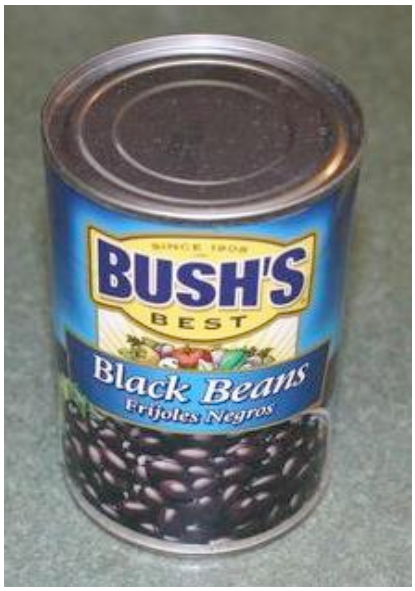
```
2
abba
ab
4
baaabad
aa
badccaa
cc
a
ab
4
bb
ab
ba
aa
```

Output for sample input

```
5
-1
```

Wojciech Rytter, adapted by P. Rudnicki

Problem I: Can of Beans



Starving, Jimmy went to the cupboard to see what he could make for lunch. Unfortunately for him, the only food item inside was an old can of beans. Under normal circumstances, he wouldn't think about it, but the old, beaten up appearance of the can worried him. "How long has this been in here?", he wondered out loud.

Turning the can over, he saw three cryptic numbers stamped into the bottom: `09 06 03`

Quickly realizing it was the best before date, he heaved a sigh of relief: the can was still good for another 4 years or so...if the first number was a year. Suddenly realizing the "09" could be the day and not the year, he wasn't so sure anymore. In fact, if the last number (the "03") was the year, then the can could have gone bad 2 years before! (This Jimmy lives in the year 2005).

You have to help out Jimmy! Given the century that Jimmy lives in and the three numbers on the bottom of the can, print out the earliest valid date the can could have gone bad.

The first line will contain an integer t which is the number of test cases to follow. Each test case consists of four integers, c, x, y, z on a line by themselves. c+1 is the century that Jimmy lives in, and is in the range $0 <= c < 2^{30}$. x, y, z are the possible day, month, and year in the century but in an unknown order. All are in the range $0 <= x, y, z < 100$.

Recall we want the earliest VALID date, so don't forget about leap years! Here are a few facts you can use about leap years:

- the year 0 is a leap year;
- every fourth year after 0 is a leap year, but
- every hundred year is NOT a leap year, but
- every four hundred year IS a leap year.

This isn't exactly true in our world, but Jimmy lives on a different planet in a different galaxy. Strangely enough, they use a calendar system astonishingly like our own, speak English, and also eat beans packaged in cans!

Jimmy's calendar, like ours, has the months January through December, numbered 01 through 12. Each of Jimmy's months contain the same number of days as our months on the standard, Gregorian calendar. The days within months are numbered from 01 to 31, and no month has more than 31 days. On a leap year, Jimmy's Februaries also have 29 days instead of 28.

The output will contain the integers in the order "YY MM DD" of the earliest possible date at which the can of beans could have gone bad. If there is no valid date, print "-1". Each test case should be written on a line by itself.

Sample Input

```
2
20 9 6 3
13244324 99 99 99
```

Sample Output

```
03 06 09
-1
```

Broderick Arneson