Berkeley Programming Contest Fall 2017

Yi Wu, and P. N. Hilfinger

Please make sure your electronic registration is up to date, and that it contains the correct account you are going to be using to submit solutions (we connect names with accounts using the registration data).

To set up your account, execute

```
source ~ctest/bin/setup
```

in all shells that you are using. (We assume you are running bash. Others will have to examine this file and do the equivalent for their shells.)

This booklet should contain 8 problems on 15 pages. You have 5 hours in which to solve as many of them as possible. Each program must reside entirely in a single file. In Java, the class containing the main program for problem N must be named `P`N. Do *not* make class `P`N public (an artifact of how we process it). Each C/C++ file should start with the line

```
#include "contest.h"
```

and must contain no other `#include` directives, except as indicated below. Upon completion, each C/C++ and Java program *must* terminate by calling `exit(0)` (or `System.exit(0)` in Java).

Aside from files in the standard system libraries and those we supply (none this year), you may not use any pre-existing computer-readable files to supply source or object code; you must type in everything yourself. Selected portions of the standard `g++` class library are included among of the standard libraries you may use: specifically, for C++:

```
cstdlib cstdio climits stdarg.h cstring cctype iostream
iomanip string sstream vector list stack queue map set
bitset algorithm cmath unordered_map unordered_set
```

In Java, you may use the standard packages `java.lang`, `java.io`, `java.math`, `java.text`, and `java.util` and their subpackages. You may not use utilities such as `yacc`, `bison`, `lex`, or `flex` to produce programs. Your programs may not create other processes (as with the `system`, `popen`, `fork`, or `exec` series of calls or their Java-library equivalents). You may use any inanimate reference materials you desire, but no people. You can be disqualified for breaking these rules.

To submit a solution, go to our contest announcement page:

`http://inst.cs.berkeley.edu/~ctest/contest`

and click on the "web interface" link. You will go to a page from which you can upload and submit files from your local computer (at home or in the labs). On this page, you can also find out your score, and look at error logs from failed submissions.

You will be penalized for incorrect submissions that get past the simple test administered by `submit`, so be sure to test your programs (if you get a message from `submit` saying that it failed, you will *not* be penalized). All tests (for any language) will use the compilation command

```
contest-gcc N
```

followed by one or more execution tests of the form (Bourne shell):

```
./N < test-input-file > test-output-file 2> junk-file
```

which sends normal output to *test-output-file* and error output to *junk-file*. The output from running each input file is then compared with a standard output file, or tested by a program in cases where the output is not unique. In this comparison, leading and trailing blanks are ignored and sequences of blanks are compressed to single blanks. Otherwise, the comparison is literal; be sure to follow the output formats *exactly*. It will do no good to argue about how trivially your program's output differs from what is expected; you'd be arguing with a program. Make sure that the last line of output ends with a newline. Your program must not send any output to `stderr`; that is, the temporary file *junk-file* must be empty at the end of execution. Each test is subject to a time limit (on ashby.cs) specified in the problem statement. You can see the test results using the web interface described above.

In the actual ACM contests, you will not be given nearly as much information about errors in your submissions as you receive here. Indeed, it may occur to you to simply take the results you get back from our automated judge and rewrite your program to print them out verbatim when your program receives the corresponding input. Be warned that I will feel free to fail any submission in which I find this sort of hanky-panky going on (retroactively, if need be).

The command `contest-gcc` N, where N is the number of a problem, is available to you for developing and testing your solutions. It converts your source program—N`.cc`, N`.java`, or N`.py`—into an executable program called just N.

For C and C++ programs, it is roughly equivalent to

```
g++ -std=gnu++11 -Wall -o N -O3 -g -Iour-includes N.* -lm
```

for C/C++. (We compile C with C++; it works most of the time.) The *our-includes* directory (typically `~ctest/include`) contains `contest.h` for C/C++, which also supplies the standard header files.

For Java programs, it is equivalent to

```
javac -g -classpath .:our-classes N.java
```

followed by a command that creates an executable file called N that runs the command

```
java PN
```

when executed (so that it makes the execution of Java programs look the same as execution of C/C++ programs).

For Python 3 programs, `contest-gcc` simply copies your file to N, makes it executable, and puts a line `#!/usr/bin/env python3` in front, which causes Unix to run it through the python3 interpreter.

The files in `~ctest/submission-tests/`N, where N is a problem number, contain the input files and standard output files that `submit` uses for its simple tests.

All input will be placed in `stdin`. You may assume that the input conforms to any restrictions in the problem statement; you need not check the input for correctness. Consequently, you C/C++ programmers are free to use `scanf` to read in numbers and strings and `gets` to read in lines.

Terminology. The terms *free format* and *free-format input* indicate that input numbers, words, or tokens are separated from each other by arbitrary whitespace characters. By standard C/UNIX convention, a whitespace character is a space, tab, return, newline, formfeed, or vertical tab character. A *word* or *token,* accordingly, is a sequence of non-whitespace characters delimited on each side by either whitespace or the beginning or end of the input file.

Scoring. Scoring will be according to the ACM Contest Rules. You will be ranked by the number of problems solved. Where two or more contestants complete the same number of problems, they will be ranked by the *total time* required for the problems solved. The total time is defined as the sum of the *time consumed* for each of the problems solved. The time consumed on a problem is the time elapsed between the start of the contest and successful submission, plus 20 minutes for each unsuccessful submission, and minus the time spent judging your entries. Unsuccessful submissions of problems that are not solved do not count. As a matter of strategy, you can derive from these rules that it is best to work on the problems in order of increasing expected completion time.

Protests. Should you disagree with the rejection of one of your problems, first prepare a file containing the explanation for your protest, and then use the `protest` command (without arguments). It will ask you for the problem number, the submission number (submission 1 is your first submission of a problem, 2 the second,

etc.), and the name of the file containing your explanation. Do not protest without first checking carefully; groundless protests will result in a 5-minute penalty (see Scoring above). The Judge will *not* answer technical questions about C, C++, Java, the compilers, the editor, the debugger, the shell, or the operating system.

Notices. During the contest, the Web page at URL

`http://inst.cs.berkeley.edu/~ctest/contest/index.html`

will contain any urgent announcements, plus a running scoreboard showing who has solved what problems. Sometimes, it is useful to see what problems others are solving, to give you a clue as to what is easy.

Problem 1. Jumbled Compass

Time limit:	1 second
Memory limit:	128 MBytes

Jonas is developing the JUxtaPhone and is tasked with animating the compass needle. The API is simple: the compass needle is currently in some direction (between 0 and 359 degrees, with north being 0, east being 90), and is being animated by giving the degrees to spin it. If the needle is pointing north, and you give the compass an input of 90, it will spin clockwise (positive numbers mean clockwise direction) to stop at east, whereas an input of -45 would spin it counterclockwise to stop at northwest.

The compass gives the current direction the phone is pointing and Jonas's task is to animate the needle by taking the shortest path from the current needle direction to the correct direction. Many ifs, moduli, and even an arctan later, he is still not convinced his `minimumDistance` function is correct; he calls you on the phone.

The input, in free format, begins with an integer n_1 ($0 \leq n_1 \leq 359$), the current direction of the needle. Next comes an integer n_2 ($0 \leq n_2 \leq 359$), the correct direction of the needle.

Output the change in angle that would make the needle spin the shortest distance from n_1 to n_2. A positive change indicates spinning the needle clockwise, and a negative change indicates spinning the needle counter-clockwise. If the two input numbers are diametrically opposed, the needle should travel clockwise. That is, in such cases, output 180 rather than -180.

Examples:

Input	Output
315 45	90
180 270	90
45 270	-135

Problem 2. Card Hand Sorting

Time limit:	1 second
Memory limit:	128 MBytes

When dealt cards in the card game Plump it is a good idea to start by sorting the cards in hand by suit and rank. The different suits should be grouped and the ranks should be sorted within each suit. But the order of the suits does not matter and within each suit, the cards may be sorted in either ascending or descending order on rank. It is allowed for some suits to be sorted in ascending order and others in descending order.

Sorting is done by moving one card at a time from its current position to a new position in the hand, at the start, end, or in between two adjacent cards. What is the smallest number of moves required to sort a given hand of cards?

The input, in free format, starts with an integer n ($1 \leq n \leq 52$), the number of cards in the hand. Next come n pairwise distinct space-separated cards, each represented by two characters. The first character of a card represents the rank and is either a digit from 2 to 9 or one of the letters T, J, Q, K, and A representing Ten, Jack, Queen, King and Ace, respectively, given here in increasing order. The second character of a card is from the set `{s, h, d, c}`, representing the suits spades, hearts, diamonds, and clubs.

Output the minimum number of card moves required to sort the hand as described above.

Examples:

Input	Output
4 2h Th 8c Qh	1
7 9d As 2s Qd 2c Jd 8h	2
4 2h 3h 9c 8c	0

Problem 3. Artwork

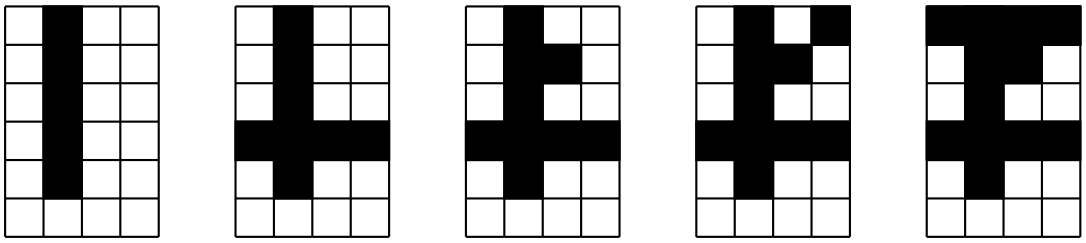
Time limit:	3.5 seconds
Memory limit:	256 MBytes

A template for an artwork is a $w \times h$ grid of white squares. The artwork will be created by painting horizontal and vertical black strokes. A stroke starts from square (x_1, y_1) and ends at square (x_2, y_2), where $1 \le x_i \le w$, $1 \le y_i \le h$, and either $x_1 = x_2$ or $y_1 = y_2$. It changes the color of all squares (x, y) to black where $x_1 \le x \le x_2$ and $y_1 \le y \le y_2$. The beauty of an artwork is the number of *regions* in the grid. Each region consists of one or more white squares that are connected to each other using a path of white squares in the grid, walking horizontally or vertically but not diagonally. The initial beauty of the artwork is 1. Your task is to calculate the beauty after each new stroke.

The input, in free format, consists of two positive integers w and h, and the number, q of strokes, where $1 \le w \le 1000$, $1 \le h \le 1000$, $1 \le q \le 10000$. This is followed by q groups of four integers x_1, y_1, x_2, y_2, as described above.

The output consists of one integer for each of the q groups, one on each line, giving the beauty (number of regions) after each stroke.

The following figure illustrates how the beauty of the artwork varies in the sample input.



1 region 3 regions 3 regions 4 regions 3 regions

Example:

Input	Output
4 6 5 2 2 2 6 1 3 4 3 2 5 3 5 4 6 4 6 1 6 4 6	1 3 3 4 3

Problem 4. Resort Life

Time limit:	1 second
Memory limit:	256 MBytes

Will went to the maze in the Disneyland Resort. The maze consists of $n + 1$ rooms ($1 \leq n \leq 1000$), labeled from 1 to $n + 1$. Will is now in room 1 and he wants to go to room $n + 1$.

Each room except $n+1$ has two doors. The first door of room i leads to room $i+1$, and the second door of room i leads to room p_i, where $1 \leq p_i \leq i$. Each time Will enters a room, he first marks a cross on the ceiling of the room. Then he counts the number of the crosses on the ceiling. If the number of the crosses is odd, he goes through the second door, which leads to room p_i. If the number of the crosses is even, he goes through the first door, which leads to room $i + 1$.

Initially Will marks a cross at the ceiling of room 1. Please help Will to find how many times he needs to use the doors to get to room $n + 1$. Will knows that it will be long day, so you only need to tell him the final number modulo 1000000007 ($10^9 + 7$).

The input, in free format, starts with a positive integer n. Next there are n integers, giving the values of the p_i, ($1 \leq p_i \leq i$).

Output a single number, the number of times Alex needs to cross the door to reach room $n + 1$, modulo $10^9 + 7$.

Examples:

Input	Output
2 1 2	4
4 1 1 2 3	20
5 1 1 1 1 1	62

Problem 5. Walk Your Dogs

Time limit:	5 seconds
Memory limit:	256 MBytes

You have two lovely dogs Shadow and Lydia and you need to walk them every day. You are lazy, so you don't really want to walk them one by one. Instead, you want to save some time to do your favorite programming contest!

Shadow and Lydia have a particular walk they each prefer and know by heart. If you just let them out, they will follow their favorite walk, eventually ending up in their respective doghouses. Problem solved!

Sadly, you realize that if you just let both dogs out at the same time and let them do their walks on their own, they might get too close to each other. If they get too close, they will leave their favorite walk to "have some fun" and you are not sure you can find good homes for any more puppies. To ensure this does not happen, you need to calculate the minimum distance between the dogs when they are out walking on their own.

Both dogs start at the same time and keep exactly the same pace. Immediately after a dog arrives at its doghouse it stays inside and goes to sleep, so we no longer need to worry about the distance to the other dog, even though the other dog may still walk for a while longer. A dog is still awake at the exact moment of entering its house and falls asleep immediately after entering.

The input, in free format, starts with an integer n ($2 \leq n \leq 100000$), the number of points describing the walk of Shadow. Next come $2n$ integers each, giving the x and y coordinates of Shadow's walk. Two consecutive points in the walk always differ in at least one coordinate. All coordinates are non-negative and at most 10000. Similarly, the next line contains an integer m ($2 \leq m \leq 100000$), the number of points describing the walk of Lydia. The next $2m$ lines describe its walk in the same format as for Shadow.

Output the minimum distance between the two dogs during their walks. The answer should be rounded to 3 decimal digits.

Examples:

Input	Output
2 0 0 10 0 2 30 0 15 0	10.000
5 10 0 10 8 2 8 2 0 10 0 9 0 8 4 8 4 12 0 12 0 8 4 8 4 12 0 12 0 8	1.414

For your reference, here is a more precise answer for the second example: 1.414213562373.

Problem 6. Summer Festival

Time limit:	1 second
Memory limit:	256 MBytes

As a member of the Berkeley Programming Association, Alex is invited to the big summer Festival of Programming! There are n entertainment activities in the festival, the ith of which starts at time s_i and ends at time t_i ($1 \leq i \leq n$). For each activity, Alex can choose to participate or not. However, if he chooses to participate in the ith activity, then he must stay there for the whole event, i.e., he cannot join after the activity begins or leave before the activity ends. Moreover, Alex cannot participate in more than one activity at any moment, even at the beginning or the end of any activity. For example, if he finishes an activity at 4PM, then he cannot immediately join another activity starting at 4PM.

Alex wants to participate in as many activities as possible. Help him determine the maximum number of activities he can participate in.

The input, in free format, starts with the number of activities n ($1 \leq n \leq 100000$). Then follow $2n$ integers, s_i and t_i ($1 \leq s_i \leq t_i \leq 10^9$), denoting the starting time and ending time of the ith activity.

Output a single integer: the maximum number of activities Alex can participate in.

Example:

Input	Output
5 1 3 2 5 4 7 6 9 8 10	3

Problem 7. Bless You Autocorrect!

Time limit:	3 seconds
Memory limit:	512 MBytes

Typing on phones can be tedious. It is easy to make typing mistakes, which is why most phones come with an autocorrect feature. Autocorrect not only fixes common typos, but also suggests how to finish the word while you type it. Jenny has recently been pondering how she can use this feature to her advantage, so that she can send a particular message with the minimum amount of typing.

The autocorrect feature on Jenny's phone works like this: the phone has an internal dictionary of words sorted by their frequency in the English language. Whenever a word is being typed, autocorrect suggests the most common word (if any) starting with all the letters typed so far. By pressing tab, the word being typed is completed with the autocorrect suggestion. Autocorrect can only be used after the first character of a word has been typed—it is not possible to press tab before having typed anything. If no dictionary word starts with the letters typed so far, pressing tab has no effect.

Jenny has recently noticed that it is sometimes possible to use autocorrect to her advantage even when it is not suggesting the correct word, by deleting the end of the autocorrected word. For instance, to type the word "autocorrelation", Jenny starts typing "aut", which then autocorrects to "autocorrect" (because it is such a common word these days!) when pressing tab. By deleting the last two characters ("ct") and then typing the six letters "lation", the whole word can be typed using only 3 ("aut") + 1 (tab) + 2 (backspace twice) + 6 ("lation") = 12 keystrokes, 3 fewer than typing "autocorrelation" without using autocorrect.

Given the dictionary on the phone and the words Jenny wants to type, output the minimum number of keystrokes required to type each word. The only keys Jenny can use are the letter keys, tab, and backspace.

The input, in free format, starts with two positive integers n ($1 \leq n \leq 10^5$), the number of words in the dictionary, and m ($1 \leq m \leq 10^5$), the number of words to type. Then follow n words, sorted in decreasing order of how common the word is (the first word is the most common). No word appears twice in the dictionary. Then follow m words to type. The dictionary and the words to type only use lower case letters 'a'–'z'. The total size of the input file is at most 1 MByte.

For each word to type, output a line containing the minimum number of keystrokes required to type the corresponding word.

Examples:

Input	Output
5 5 austria autocorrect program programming computer autocorrelation programming competition zyx austria	12 4 11 3 2
5 3 yogurt you blessing auto correct bless you autocorrect	5 3 9

Problem 8. Cherry Cake

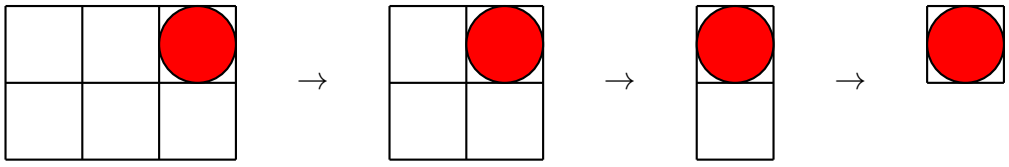
Time limit:	1 second
Memory limit:	256 MBytes

Alice and Bob like cake. A cake can be viewed from above as a 2D $m \times n$ grid. It can be divided into mn pieces. However, one of the pieces contains a cherry, and neither Alice nor Bob likes cherries, so they come up with the following way to decide who gets the cherry piece.

First Alice cuts the cake in two pieces along a grid line and takes the piece without the cherry. Then Bob does the same with the remaining cake and they keep alternating until only the square containing the cherry is left, forcing the next person to take it.

For example, with a 2×3 grid cake:

1. Alice cuts the leftmost column. (So Bob is left with a 2-by-2 grid cake.)
2. Bob cuts the leftmost column. (So Alice is left with a 2-by-1 grid cake.)
3. Alice cuts the bottom square off. (Bob is left with the cherry piece and is forced to take it.)



Assume Alice and Bob are both smart enough to avoid the cherry if possible. Given a cake and the position of the cherry, who will take the cherry piece?

The input, in free format, starts with an integer T ($1 \le T \le 100$), the number of test cases. Then $4T$ integers follow, four for each test case. Each test case—m, n, r, c—specifies m and n ($2 \le m, n \le 48$), the width and the length of the cake, and (r, c), the zero-based position of the cherry piece in the cake grid ($0 \le r < m, 0 \le c < n$).

For each test case, print the name of the person that gets the bad piece (cherry piece), assuming Alice makes the first cut and that Alice and Bob always cut the cake at the optimal location.

Example:

Input	Output
2 2 3 0 2 11 11 5 5	Bob Alice