Berkeley Programming Contest Fall 2016

Alex Dai, Pasin Manurangsi, 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 9 problems on 16 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/index.html`

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 -I`*our-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

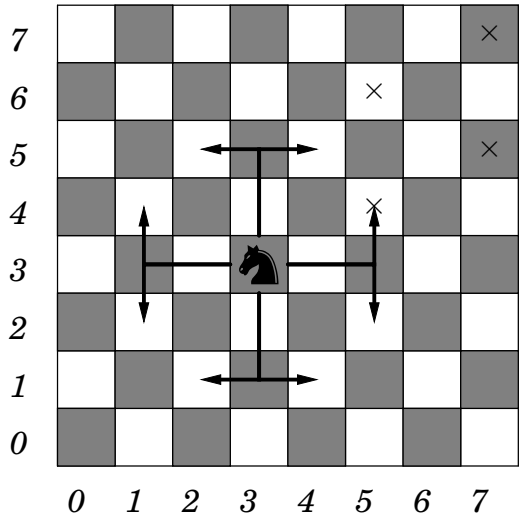
`http://inst.cs.berkeley.edu/~ctest/contest/index.html`

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. Knight's Path

Time limit: 2 seconds
Memory limit: 512 MBytes

In chess, a knight moves up, left, right, or left two squares and then one square at right angles:



Write a program that, given two squares on a board, prints the smallest number of knight moves needed to travel from one square to the other. For example, given the row and column numbering shown in the board above, the minimum number of moves required for the knight at (3, 3) to reach the upper-right corner square (7, 7) is four (using the squares marked with '$\times$' in the diagram, for example.)

The input to your program will consist of some number of quintuples of integers—$2 < N \leq 1000, 0 \leq R_1 < N, 0 \leq C_1 < N, 0 \leq R_2 < N, 0 \leq C_2 < N$—in free format, one for each test case. Each test case asks for the minimum number of knight moves on an $N \times N$ board from square (R_1, C_1) (row R_1, column C_1) to square (R_2, C_2).

For each test case, print out the minimum number of moves needed on its own line. If it is not possible to get to the target square, print `-1`.

Example:

Input	Output
`8 3 3 7 7`	`4`
`50 1 1 2 1`	`3`
`3 1 1 1 1`	`0`

Problem 2. Problem Selection

Time limit:	1.5 seconds
Memory limit:	512 megabytes

[This problem is based on a true story.]

Imagine for a moment that you are a lazy graduate student. One night while you are enjoying some good Netflix series, your friend called you and asked you to help write some problems for the Berkeley ICPC programming contest. Being a nice friend that you are, you immediately said yes. In hindsight, this may not be the best decision, but there is no escaping now.

Since you are a lazy graduate student, you plan to just pull the problems from your collection of past problems. Each problem in your collection has three attributes:

- *hardness*: how hard the problem is, which is just a integer from one to four.
- *category*: the category of the problem (e.g. dynamic programming, greedy), which is a string.
- *workload*: the amount of additional work required to make the problem available for the contest (i.e. writing test data, coming up with fancy story, and all that good stuff). This is an integer between zero and a million.

Your friend is very picky. He has asked you to give him exactly eight problems, two problems from each hardness level. Moreover, he does not want more than two problems from the same category to appear in the contest. Since you want to spend as little effort as possible on these problems but still make your friend happy, help yourself write a program to find eight problems from you collection that satisfy your friend's conditions while also minimize the total amount of workload.

The input, in free format, starts with a single integer $n \leq 50$, the number of problems in your collection. There follow n triples of data about the problems in your collection; the i-th contains h_i, c_i and w_i where $1 \leq h_i \leq 4$ and $0 \leq w_i \leq 1000000$ are integers giving to the hardness and the workload for problem i. Each c_i is a non-empty string consisting of at most 30 letters indicating the category of the problem.

The output is a single integer giving the minimum total amount of workload you need to prepare the problems that satisfy your friend's constraints. It is guaranteed that there are eight problems in the collection that make your friend happy.

Example:

Input	Output
10 1 DP 3 2 GEO 36 1 DP 6 2 GREEDY 10 2 DP 5 3 DS 15 3 NT 3 4 DS 20 4 DP 16 4 GEO 100	193

Problem 3. Divisor Cover

Time limit: 1 second
Memory limit: 512 megabytes

A set of distinct positive integers $A = \{a_1, \ldots, a_m\}$ is said to *cover* a positive integer b if and only if b is divisible by at least three elements of A. For instance, 36 is covered by $\{6, 7, 9, 36\}$ because $6, 9$ and 36 divide 36. However, 36 is not covered by $\{5, 7, 12, 37\}$ because only 12 divides 36.

Given positive integers $b_1, \ldots, b_n \leq 2000$, print out a minimum number m such that there exists a set A of m distinct positive integers that covers $b_1, \ldots, b_{n-1}$ and b_n. If such set A does not exist, print -1.

The input, in free format, begins with a positive integer $n \leq 200$. There follow n integers $b_1, \ldots, b_n \leq 2000$.

The output is a single integer, which is the minimum size of a set A that covers $b_1, \ldots, b_n$ or -1 if no such A exists.

Example 1:

Input	Output
3 16 24 36	3

Example 2:

Input	Output
3 1 3 7	-1

Problem 4. Coin Collector

Time limit: 1 second
Memory limit: 512 megabytes

In the Coin Collector game, you start off in the upper-left cell of an $n \times n$ grid, in which each cell contains a certain number of gold coins. In each step, you can either move right or move down. For each cell you have passed through, you can collect all the gold coins in it; the coins disappear forever once you collect them. The goal of the game is to travel from your starting point to the lower-right cell of the grid while collecting as many coins as possible. An example of a game and a sequence of moves can be found in Figure 1(a).

Being generous moderators that we are, we allow you one special move today: *teleportation*. When you use this move, you can teleport from where you are now to anywhere on the grid. You are allowed to use this move at most once (and need not use it at all). Given the grid, along with the number of coins on each cell, compute the maximum number of gold coins you can collect if you travel from the upper-left cell to the lower-right cell being with at most one teleportation. Figure 1(b) shows an optimal sequence of moves involving a teleportation.

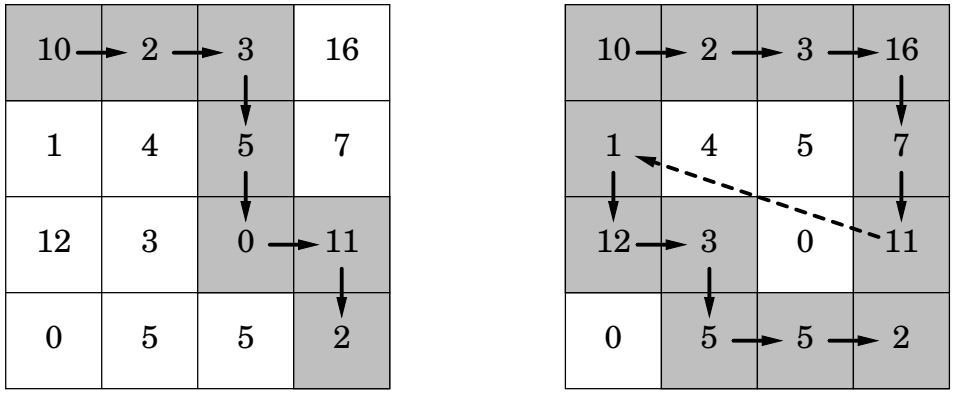


Figure 1: *

Fig. 1: A coin collector game and examples of moves. Each number of the 4×4 grid denotes the number of gold coins in that cell. In (a), a valid sequence of moves not involving teleportation is shown; for this sequence, you can collect 33 gold coins. In (b), an optimal sequence of moves involving a teleportation is illustrated; the teleportation is represented with the dashed line. For these moves, you can collect 77 gold coins.

The input to your program is in free format. It starts with a positive integer $n \leq 400$, which is the size of the grid. Next follow n groups of n non-negative integers representing the number of gold coins in each cell of the board. Specifically, the j-th number in the i-th group, $0 \leq c_{i,j} \leq 100000$, is the number of gold coins in the cell on the i-th row (numbering from 0 at the top) and j-th column.

The output from your program consists of a single integer representing the maximum number of coins you can collect.

Example 1:

Input	Output
4 10 2 3 16 1 4 5 7 12 3 0 11 0 5 5 2	77

Example 2:

Input	Output
3 2 3 7 4 1 2 11 6 5	40

Problem 5. Shooting Scorer

Time limit: 2 seconds
Memory limit: 512 megabytes

You are invited to develop a scoring system in a shooting club! A player will shoot the target with M bullets, which results in M bullet holes on the target paper. There is a scoring region on the target paper. If a bullet hole locates **strictly** inside the scoring region, the player gets 1 point. If we consider the target paper as a 2-dimensional plane, then the scoring region is a convex polygon with N boundary points and a single bullet hole can be seen as a point in the plane. Now given the coordinates of the N boundary points of the scoring region and the M bullet holes, your system need to compute how many points the player gets.

The input, in free format, starts with a single integer N ($3 \leq N \leq 10^5$). There follow N pairs of integers, describing the boundary points of the scoring region in counterclockwise order: x_i^S and y_i^S where $|x_i^S|, |y_i^S| \leq 10^9$. All these boundary points are distinct and the scoring region must have a positive area.

Next comes a single integer M ($0 \leq M \leq 10^5$), followed by M pairs of integers describing all the bullet holes: x_i^B and y_i^B where $|x_i^B|, |y_i^B| \leq 10^9$. All the bullet holes are distinct.

The output consists of a single line with one integer, the player's final score.

Example:

Input	Output
4 0 0 5 0 5 5 1 5 4 0 0 1 1 6 6 5 3	1

Problem 6. Backend System

Time limit: 3 seconds
Memory limit: 512 megabytes

You are requested by the customer to develop a data management system. You are given an array A with N elements, $A[1 \ldots N]$. The customer requires the system to execute Q operations sequentially. Each operation will be in one of the 3 following forms:

- `1 t g c`: you need to multiply $A[t], A[t+1], \ldots, A[g]$ by c.
- `2 t g c`: you need to increase the value of $A[t], A[t+1], \ldots, A[g]$ by c.
- `3 t g`: you need to print the sum of $A[t], A[t+1], \ldots, A[g]$.

For the 3rd form of operation, the output number will be extremely large. So, you only need to output the result of the true sum modulo P.

The input, in free format, starts with two integers N ($1 \le N \le 10^5$) and P ($1 \le P \le 10^9$). Next come N integers, the initial value of the array, $A[1 \ldots N]$ ($0 \le A[i] \le P$). Next comes a single integer Q ($1 \le Q \le 10^5$). Finally come Q specifications of the operations. Each operation must be one of the 3 forms with $1 \le t \le g \le N$ and $0 \le c \le 10^9$.

For each operation in the 3rd form, output a single line with one integer, the sum modulo P.

Example:

Input	Output
7 43 1 2 3 4 5 6 7 5 1 2 5 5 3 2 4 2 3 7 9 3 1 3 3 4 7	2 35 8

Problem 7. Lucky Seat

Time limit: 1 second
Memory limit: 512 megabytes

You know that a good seat may help you achieve better scores in the final exam! Since the exam is approaching, you hack the laptop of the dean and obtain detailed information about the exam hall. Now you want to infer which seat is the best choice. The hall has N rows and each row has N seats. The ith row has a row lucky number $R[i]$ while the jth column has a column lucky number $C[j]$. The luckiness of seat at ith row and jth column is $R[i] * C[j]$.

Your lucky seat is the seat with the K-th highest luckiness. You want to determine the luckiness of your lucky seat.

The input, in free format, starts with two integers, N and K ($1 \leq N \leq 50000$, $1 \leq K \leq N^2$). Then come N integers, the row lucky numbers $R[1 \ldots N]$ ($0 \leq R[i] \leq 10^5$). Finally, there follow N integers, the column lucky numbers $C[1 \ldots N]$ ($0 \leq C[i] \leq 10^5$).

The output consists of a single line containing a single integer, the luckiness of your lucky seat.

Example:

Input	Output
5 3 1 5 7 2 8 9 7 5 10 11	77

Problem 8. Textbook Sorting

Time limit: 1 second
Memory limit: 512 megabytes

Alex has a stack of N textbooks labeled from 1 to N. Textbook 1 is supposed to be at the bottom, and N at the top. Unfortunately, the ordering of the textbooks is all messed up, so Alex wants to sort them in ascending order. However, the textbooks are really heavy, so the only operation he can do is pull some textbook out of the stack and put it at the top. Since moving textbooks is really tiring, he wants to know the minimum number of operations needed to sort the stack. Unfortunately, this problem is too hard for Alex. Can you help him?

The input, in free format, starts with an integer $1 \leq N \leq 10^5$. There follow N integers representing the initial order of the textbooks from bottom to top.

The output consists of a single integer giving the minimum number of operations needed to sort the stack.

Example 1:

Input	Output
`4` `1 4 2 3`	`1`

Example 2:

Input	Output
`4` `1 2 3 4`	`0`

Problem 9. Circular Walk

Time limit: 1 second
Memory limit: 512 megabytes

It's a beautiful day outside! Oski has decided to take a pleasant walk around the lake in the nearby park. Unfortunately, Oski is a busy bear, so he doesn't want his walk to take too long. How long is the shortest walk that takes Oski around the lake and returns him to his starting position? Oski can only move horizontally and vertically.

The input, in free format, starts with two integers N and M ($1 \leq N, M \leq 100$). Next come N strings each containing M characters, forming a grid that represents the park. Water cells are denoted 'W', empty cells are denoted '.', and Oski's starting position is denoted 'S'. It is guaranteed that the water cells form a connected component and that there is at least one water cell in the input. It is also guaranteed that there is exactly one 'S' in the input.

The output consists of a single integer: the length of the shortest walk around the lake, starting and ending at Oski's starting position.

Example 1:

Input	Output
3 3 ... .W. .S.	8

Example 2:

Input	Output
5 8 S....... ..WWWW.. ..W..... ..WWWW..	20

Example 3:

Input	Output
4 5 ..W.. S.W.. ..W.. ..W..	-1