Contest Problems
Philadelphia Classic, Fall 2016
Hosted by the Dining Philosophers
University of Pennsylvania

dining philosophers

UNIVERSITY OF PENNSYLVANIA | COMPUTER SCIENCE CLUB

Rules and Information

This document includes 12 problems. Novice teams do problems 1-8; standard teams do problems 5-12.

Any team which submits a correct solution for any of problems 1-4 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-4.**

Problems 1-4 are easier than problems 5-8, which are easier than problems 9-12. These problems are correspondingly labeled “Novice”, “Intermediate”, and “Advanced.” Order does not otherwise relate to difficulty.

You may use the Internet only for submitting your solutions, reading Javadocs or Python documentation, and referring to any documents we link you to. You **may not** use the Internet for things like StackOverflow, Google, or outside communication.

As you may know, you can choose to solve any given problem in either **Java or Python**. We have provided stub files for all questions in both languages that takes care of the input parsing for you. **Do not modify any of the parsing or output code**. Just fill out the stub methods in each file and submit it with exactly the same name.
Do not modify any of the given methods or method headers in our stub files! They are all specified in the desired format. You may add class fields, helper methods, etc as you like, but modifying given parts will cause your code to fail in our testers.

There is no penalty for incorrect submissions. You will receive 1 point per problem solved. A team’s number of incorrect submissions will be used only as a tiebreaker.

Some problems use Java’s “long” type; if you are unfamiliar with them, they’re like an “int”, but with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
long myLong = 1000000000000L;
```

Otherwise, the “long” type functions just like the “int” type.

1. Ancient Greetings

The creators of PClassic have encountered a strange device while on an excavation trip before PClassic 1738. Every time the device is fed a person's name, it blurts out a greeting. After opening up the device and trying to reverse engineer it, the creators of PClassic have discovered that the device only blurts out the greeting "Good morrow" followed by the person's name. However, they have been trying to reprogram the device to use a different greeting. Specifically, they have been trying to use the more modern greeting "Hello" instead, but they have not yet been able to reprogram the device. Can you help them reprogram the device?

Your program will be given a string **`S`** representing a person's name. You must return the greeting: "Hello " followed by the string **`S`** followed by a period.

Sample Input	Sample Output	Explanation
`Newton`	`Hello Newton.`	The required greeting is "Hello Newton."
`Charles Babbage`	`Hello Charles Babbage.`	The required greeting is "Hello Charles Babbage."

2. Dot Matrix Printer Problems

Peace Bro! It's PClassic 1970. Things have been pretty crazy lately. The war in Vietnam is winding down, the North Tower of the World Trade Center has just been finished, and the Environmental Protection Agency has just been formed. Amidst all of this progress, trouble is brewing in PClassic land. The organizers, always keen on using the latest technologies, have insisted that the PClassic logo, a large capital "P", be printed on the newest Dot Matrix Printer. These printers work by printing dots from left to right, sequentially filling in each row of an image. Despite how simple this sounds, the PClassic organizers have been listening to *Lucy in the Sky with Diamonds* by the Beatles for too long and seem too disoriented to complete the task. Can you help them out?

The logo will be printed on a grid Nx(N-1) grid. The Dot Matrix Printer can either print a dot or no dot in each cell represented by a "X" for a dot and an "_"(underscore) for a cell that doesn't contain a dot. Your task is to draw the PClassic Logo (a capital "P") on this grid based on the size of the input (N). The longer vertical part of the P will be of length N (where N is an odd number). The horizontal parts of the P are of length N-1. The second horizontal part of the P will start on the exact middle row. Your input will be N and your output should be the string representing the Logo (including newline characters).

Sample Input	**Sample Output**	**Explanation**
5	XXXX X__X XXXX X___ X___	The longer vertical side has length 5. The horizontal parts have lengths 4. The second horizontal part is on the 3rd row.
7	XXXXXX X____X X____X XXXXXX X_____ X_____ X_____	The longer vertical side has length 7. The horizontal parts have lengths 6. The second horizontal part is on the 4rd row.

3. Athenian Voter Fraud

In the fifth century B.C.E., the Greek city-state of Athens developed the first known democracy on the planet. The Athenian system was a direct-democracy, meaning that every tax-paying Athenian who has completed their military training as ephebes[1] could vote directly on legislation. As you can imagine, as the first democratic state, Athens suffered from problems in its democratic system. Namely, it had widespread corruption and rampant voter fraud.

However, the Athenians think you can help with your fancy computers: they want you to write a program which detects which voters have committed voter fraud. Every voter is issued a unique integer identifier which qualifies them to vote in the city. Every time an Athenian casts a ballot, their identifier is recorded. Your program will be given an array **`votes`** which contain the identifier for every ballot which was submitted. Your program should return the identifiers of voters who voted more than once. You can return the identifiers in any order.

Note: All identifiers are between 0 and 10,000.

Sample Input	**Sample Output**	**Explanation**
`123,452,12,123`	`123`	123 voted twice
`14,15,14,15,16,14`	`14,15`	14 voted three times and 15 voted twice
`102,103`	`none`	Everyone voted once

1 https://en.wikipedia.org/wiki/Ephebos

4. ENIAC Accumulating

The ENIAC (Electronic Numerical Integrator And Computer)[2] was the first electronic general-purpose computer. It was built right here at Penn, and when it came online in 1946 it was one thousand times faster than existing machines, and could calculate missile trajectories in seconds. While the ENIAC was truly revolutionary, its architecture included some interesting quirks. For example, the ENIAC operated on base ten integers, not binary ones. It contained 20 accumulators (basic memory units), which stored base ten representations of integers and could perform an astonishing 5000 operations every second.

You are tasked with implementing the most basic of these operations: incrementation. Your program will allow ENIAC to take a base ten integer stored in an accumulator and increase it by 1. Specifically, your program will be given an array **a** of length **n + 1. a** represents an **n digit** positive integer, and each element of a is a digit. For example **`[0, 1, 2, 3]`** represents the number **123**. Note that a is always 0-padded. Your program should increment **a** by 1.

Sample Input	**Sample Output**	**Explanation**
`0,1,2,3`	`0,1,2,4`	123 + 1 = 124
`0,9,9,9`	`1,0,0,0`	999 + 1 = 1000
`0,1`	`0,2`	1 + 1 = 2

[2] https://en.wikipedia.org/wiki/ENIAC

5. Tricky Triangular Triples

Welcome to PClassic 250 BC! You are in Ancient Rome, where you have been working hard on creating the designs of the very first aqueducts. You have been given the important task of creating a support system that will hold the aqueduct. Fortunately, you have read a recently published text from the Greek mathematician, Pythagoras, about the applications of the Pythagorean Theorem in bridge construction. Inspired, you want to create a right triangular support structure where the length of the aqueduct is the hypotenuse and the length of the two support beams are the base and the height. However your building materials have already been cut into standard size cubes so you would like to make structures that don't require cutting or modifying your bricks. Given the length of the hypotenuse in bricks you would like to determine all the possible combinations of leg lengths that don't require cutting any bricks (that is the lengths are whole numbers when measured in bricks).

You will be given a positive integer that is the length of the aqueduct you want to construct. Your task is to return all possible configurations of the other two sides that are of positive integer length. Output each configuration as a separate element in a list in the format ***a b*** where ***a*** is in ascending order and ***b*** is in descending order in respect to the other configurations.

If there are no possible integer configurations, return an empty list.

Sample Input	**Sample Output**	**Explanation**
5	[3, 4, 4, 3]	$3^2 + 4^2 = 5^2$ $4^2 + 3^2 = 5^2$
25	[7, 24, 15, 20, 20, 15, 24, 7]	$7^2 + 24^2 = 25^2$ $15^2 + 20^2 = 25^2$ $20^2 + 15^2 = 25^2$ $24^2 + 7^2 = 25^2$
3	[]	There are no configurations where the base and height are integers.

6. Fastidious Farming

It's PClassic 10,000 BC, and you're working on the cutting edge of technology - agriculture. Specifically, you're competing to farm the maximum quantity of crops with limited resources.

You want to grow two kinds of crops, creatively named crops A and B. They each take a specific amount of water to grow and require a specific number of hours of tending (labor) per unit. Water and labor are both difficult to come by, and you have a limited amount of each. However, you want to make sure to use exactly all of the water and labor you do have access to, in order to not waste any resources.

Your task is to determine how many of each kind of crop you should grow, given the exact number of units of water and labor you can use. That is, you will have two integers on the first line of input separated by a space, representing the amount of water and labor, respectively, necessary to grow one unit of crop A. Similarly, the second line will have two integers separated by a space representing the amount of water and labor, respectively, necessary to grow one unit of crop B. Finally, the third line will have two integers separated by a space representing the amount of water and labor you should use. You need to output the amounts of crop A and crop B you should grow, respectively, as two integers separated by a space.

At maximum, one distinct integer solution will be possible. If no nonnegative integer outputs are possible, output "-1 -1" to indicate that there is no solution.You can assume that the inputs are nonnegative integers.

Sample Input	**Sample Output**	**Explanation**

`1 4`
`3 8`
`7 24`
	`4 1`	4 units of crop A require 4 units of water and 16 units of labor. 1 unit of crop B requires 3 units of water and 8 units of labor. The total is 7 units of water and 24 units of labor.

`11 13`
`5 17`
`102 176`
	`7 5`	7 units of crop A require 77 units of water and 91 units of labor. 5 units of crop B require 25 units of water and 85 units of labor. This totals to 102 units of water and 176 units of labor.

`1 5`
`2 10`
`5 4`
	`-1 -1`	No quantity of crop A and crop B will use exactly the correct amount of water and labor.

7. Treasons and Senators

Good morning from Philadelphia Classic 1763 and you're already helping Ben Franklin's readers plot a "revolt". Many historical revolutionary authors went under pseudonyms to publish their work, Publius or Martha Careful anybody? (Alexander Hamilton's and Ben Franklin's pennames respectively) Ben Franklin however has concocted another plan to disguise his attention. In order to successfully disseminate his freedom loving message Ben Franklin has "edited" some of the potentially "dangerous" words in his newspaper articles with their anagrams. However many of the readers are quite confused by these mysterious anagrams. Therefore Ben Franklin's readers have been grouping together these "anagrammed" words based on whether they anagram to the same word and ordering these words alphabetically. However the readers find this extraordinarily time consuming, your job is to write a program for them to avoid this process.

You will be given a list of words for this task. Your program should return the largest set of words that all anagram to the same word (meaning they contain the same letters) and return this set of words in alphabetical order. For this problem the largest set of words that anagram to each other will be unique and all words in the initial list are distinct.

Input Limits
There will be at most 100 words in the given list.

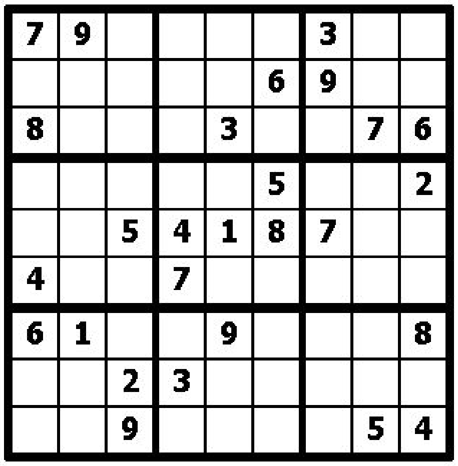
Sample Input	**Sample Output**	**Explanation**
`dbac dbca dabc dacb dcba dcab bdca bdac bacd badc bcad bcda adbc adcb abdc abcd acdb acbd cdab cdba`	`[abcd, abdc, acbd, acdb, adbc, adcb, bacd, badc, bcad, bcda, bdac, bdca, cdab, cdba, dabc, dacb, dbac, dbca, dcab, dcba]`	All the "words" are anagrams of each other so it suffices to output the words in alphabetical order.
`ab abajn bhsjhs ckn d efgdhs f ba gjlkj h ihdjs j khsjahsn ljsk mhsjhs n okjka pljks qgdhsg rdghs`	`[ab, ba]`	Only the first two words are anagrams of each other so only those two are returned. (Note that the length of strings in the given input may vary.)

8. Sudoku Solver

Welcome to PClassic 1986 where a new strange puzzle has come out of nowhere to frustrate thousands of people who are trying to solve it. This puzzle, called Sudoku, has been created by the crafty company Nikola who claims to have created the hardest puzzle ever known to man. The Sudoku puzzle involves a 9x9 grid of numbers, where each cell contains a number between 1 and 9. Each row and column in the grid contains all the numbers from 1 to 9 only once. There are also 9 bolded 3x3 squares in the Sudoku puzzle. Each bolded square also must contain all the numbers from 1 to 9 only once. Sounds hard doesn't it? However, many computer scientists around the world believe that the puzzle is very easy to solve using a computer program. You, as an aspiring computer scientist, wish to write such a computer program capable of solving these puzzles.

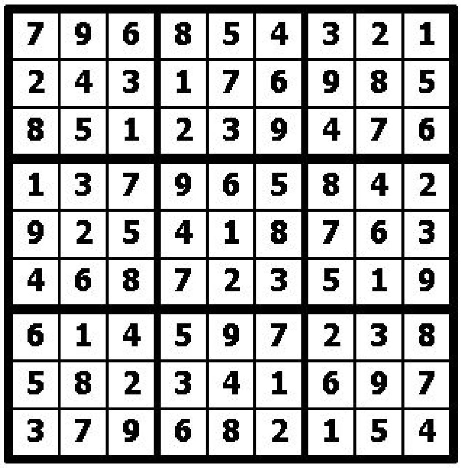
You will be given 9 rows with 9 integers each separated by a space. These integers will represent values on a Sudoku puzzle. If the integer is 0, then it represents an empty value. You will return 9 rows with 9 integers where every integer is a value between 1 and 9 inclusive. In the sudoku solution corresponding to your output, each row, each column, and each box must contain the values 1 to 9 only once. **Note: You will get a maximum of 15 empty squares**.

A sample sudoku puzzle is shown below (left side is the puzzle and the right side is the solution).



7	9					3		
					6	9		
8				3			7	6
					5			2
		5	4	1	8	7		
4			7					
6	1			9				8
		2	3					
		9					5	4

Copyright 2005 M. Feenstra, Den Haag



7	9	6	8	5	4	3	2	1
2	4	3	1	7	6	9	8	5
8	5	1	2	3	9	4	7	6
1	3	7	9	6	5	8	4	2
9	2	5	4	1	8	7	6	3
4	6	8	7	2	3	5	1	9
6	1	4	5	9	7	2	3	8
5	8	2	3	4	1	6	9	7
3	7	9	6	8	2	1	5	4

Copyright 2005 M. Feenstra, Den Haag

Sample Input	**Sample Output**	**Explanation**

7 9 6 8 5 4 3 2 1
2 4 3 1 7 6 9 8 5
8 5 1 2 3 9 4 7 6
0 0 7 0 6 5 8 4 2
0 2 5 4 1 8 7 6 3
4 0 8 0 2 3 5 1 9
	7 9 6 8 5 4 3 2 1
	2 4 3 1 7 6 9 8 5
	8 5 1 2 3 9 4 7 6
	1 3 7 9 6 5 8 4 2
	9 2 5 4 1 8 7 6 3
	4 6 8 7 2 3 5 1 9
		The solved sudoku solution is shown above. The sample output contains the digits 1-9 in every row, column, and bolded box only once.

6 1 4 5 9 7 2 3 8
5 8 2 3 4 1 6 9 7
3 7 9 6 8 2 1 5 4
	6 1 4 5 9 7 2 3 8
	5 8 2 3 4 1 6 9 7
	3 7 9 6 8 2 1 5 4	

9. Time Travel Tree Traversal

As you know every PClassic has a theme. In 2013 the theme was time travel. In that spirit the last question was to write a program that could travel through time. It was an extremely hard question. The only team that even attempted to solve it only ended up with a program that split time into two parallel timelines. Unfortunately since each alternate timeline also included the branching program it resulted in an exponential explosion of alternate timelines. After the contest the organizers had to clean up all those extra branches of reality, a process that can be quite time consuming. Each time there is a split in a timeline A we say that it creates two new timelines B and C. Any two timelines D and F share some common ancestor G from with they were both split. If D split off from G m splits ago, and F split off from G n splits ago, we say that the distance between D and F is m + n. The amount of time it takes to cleanup time is proportional to the maximum distance between any two timelines. For example if timeline A splits into timelines B and C, and then there is another split in B creating timelines D and E, the maximum distance is from D (or E) to C. Their common ancestor is A. D split from A 2 splits ago and C split from A 1 split ago so the distance between them is 3. The PClassic organizers wrote a program to figure out what this maximum distance would be but forgot to return it to this timeline. We would like you to recreate it for us.

You will be given a list of ordered triples of integers (a, b, c) such that a, b, and c are between 0 and n-1 inclusive where n is the total number of timelines. Each triple (a, b, c) indicates that timeline 'a' split into timelines 'b' and 'c'. Note that if a timeline spawns no new timelines it will never appear in the first position in an ordered triple. Additionally timeline 0 will always be the initial timeline, so 0 will never appear in the second or third position of a triple. You will return an integer indicating the largest distance between two timelines.

In the sample input the first line is the number of triples and each line after that contains a single triple

Sample Input	**Sample Output**	**Explanation**

`2`
`0 1 2`
`2 3 4`
	`2`	The distance between timelines 1 and 3 is 2.

`6`
`0 1 2`
`2 3 4`
`3 5 6`
`4 7 8`
`7 9 10`
`5 11 12`
	`6`	The distance between timelines 11 and 9 is 6.

10. ENIGMA

Something as simple as World War II is not enough to stop PClassic from happening! However, the contestants of PClassic 1943 have been asked to help out with the war effort by cracking the German ENIGMA cipher.

One method that the British have developed to crack the ENIGMA is to make use of statistical biases in "human-generated" randomness. The ENIGMA cipher needs a random password to work, but for some lazy operators, instead of relying on proper randomness generation techniques, have just been randomly typing on a keyboard instead. The Allied forces want your help to figure out which is which, so that they can win World War II!

Your task is to read in pairs of binary strings and figure out which is generated using a strong random number generator (i.e. flipping a coin) and which is not. To aid in this problem, we have provided two sample files of data (Enigma.random.txt and Enigma.nonrandom.txt). The first contains truly random strings on each line; each character in the string is generated by selecting 0 or 1 using a strong random number generator. The second contains "human generated" random strings, which are created by ENIGMA operators randomly typing on a keyboard: letters typed using the left hand are encoded as 0, and letters typed using the right hand are encoded as 1. Each string will consist of 500 binary numerals. Clearly it is impossible to write a program the exactly solves this problem since any string could be generated by a random process. Instead what we want is for you to use the provided data to come up with some heuristics that exploit the fact that people are not truly random.

For each line of input (which will **not** be drawn from the sample files provided to you, but will be generated in a similar manner), output either "random", if the first string is truly random, or "nonrandom", if the first string is generated by typing.

Sample Input	Sample Output	Explanation

`01010101001010101010`
`01100001010111000100`
	`nonrandom`	The first was generated by randomly typing on a keyboard, the second was generated by flipping a coin

`10101110000010011000`
`01001010111010110100`
	`random`	The first was generated by flipping a coin, the second was generated by randomly typing on a keyboard

11. Minimizing a Maritime Merchant’s Costs

Good morrow from PClassic 1653! It is the age of exploration and Philadelphia is a bustling center of trade. This year the contest’s corporate sponsor is a local merchant who hopes that the contestants can solve a problem for his business (the 17th century organizers were much less concerned with conflicts of interest). The merchant has many goods he needs to move to far corners of the world, but unfortunately many of the destinations do not have direct shipping routes from Philadelphia. Each ship has a set of ports that it stops at, visiting each in turn in a loop. Ships do not charge based on how much of their journey the cargo takes but with simply a flat charge when the cargo is unloaded. Thus the merchant needs to minimize the number of times his goods will have to change ships. He wants you to write a program that will compute this minimum number of stops.

You will be given an array of lists of integer ids. Each list corresponds to a ship and its contents represents the ids of the stops that ship makes. You will also be given the ids of the starting and ending port of the shipped goods. You will output the minimum number of ship transfers needed to get from the starting port to the ending port. If you cannot reach the destination, output -1 instead.

Input Limits

There will be at most 2000 ships.
Each ship will visit at most 2000 ports.

The first line of each input contains the number of ships N, the maximum number of ports M, and the starting and stopping ports. The next N lines list the stops each ship makes so each line will contain integers from 0 to M-1. Note that is possible for no ship to visit a port.

Sample Input	Sample Output	Explanation

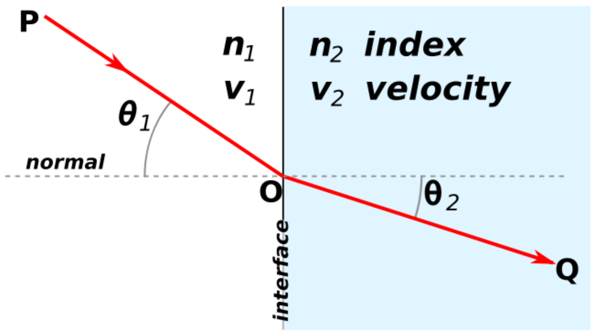
`2 4 0 2`
`0 1 3`
`1 2`
	`1`	You need to ride from port 0 to port 1 on the ship 0 and then from port 1 to 2 on ship 1

`2 4 1 2`
`0 1 3`
`1 2`
	`0`	You can ride directly from 1 to 2 without any transfers

`3 9 0 8`
`0 1 2`
`4 2`
`4 8 1`
	`1`	You can ride from 0 to 1 and then 1 to 8 using the first and last ships.

12. Newton's Nightmare

It is PClassic 1666 and Sir Isaac Newton is experimenting with optics and prisms, getting ready to publish his famous treatise *Opticks*. He has a number of layers of glass and other materials with different heights and refractive indices, and for his experiment to succeed, he needs to shine a beam diagonally through these layers to hit a given target. Of course, it isn't as easy as just aiming the beam directly at the target: you are familiar with Snell's law, which states that when a light beam crosses over to a new material, it refracts (changes direction) as follows: $\frac{\sin \theta_1}{n_1} = \frac{\sin \theta_2}{n_2}$ (refer to the diagram on the right, image from Wikipedia).



Newton's light beam starts at (0, 0) on a 2D plane. There are some layers of materials in between these two points, stacked parallel to the x-axis. For instance, there could be a layer of glass (refractive index 1.33) with height 3, followed by a layer of diamond (refractive index 2.4) with height 1. Your job is to figure out what angle above the x-axis you need to shoot the light beam at so as to hit some point (x, y). For simplicity, the value y will be equal to the sum of all heights (i.e. the point you need to hit will be at the very top of all layers; in the above example, the value of y would be 3 + 1 = 4). Help Newton to complete his experiment!
Note: Your answer will be rounded to the nearest 0.0001 degree. There will be at most 40000 layers of materials

Sample Input	**Sample Output**	**Explanation**

`1 3`
`3 4.5`
	`45.0000`	There is only one layer of material, of height 3. Shooting at a 45 degree angle, you will hit (3, 3), since the light never refracts.

`2 2`
`1 1`
`1 2`
	`60.0000`	There are two materials, both of height 1, Though the first (with y values 0 to 1) has refractive index 1, and the second (y values 1 to 2) has refractive index 2. Shooting the light beam at a 60 degree angle above the x-axis, the beam will refract between the two layers and hit (2, 2) correctly