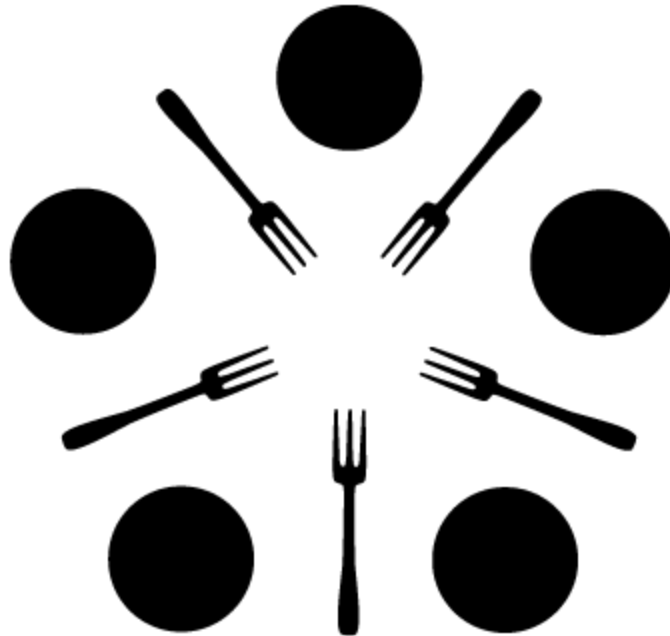


Contest Problems
Philadelphia Classic, Spring 2014
Hosted by the Dining Philosophers
University of Pennsylvania



dining philosophers
UNIVERSITY OF PENNSYLVANIA COMPUTER SCIENCE CLUB

Rules and Information

This document includes 14 problems. Novice teams do problems 1-10; standard teams do problems 6-14.

Any team which submits a correct solution for any of problems 1-5 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-5.**

Problems 1-5 are easier than problems 6-10, which are easier than problems 11-14. Problems are correspondingly labelled “Novice”, “Intermediate”, and “Advanced”. Order does not otherwise relate to difficulty, except that problem 14 is the hardest.

You may use the internet only for PC² (to submit your solutions), reading Javadocs, and referring to any documents we link you to. You **may not** use the internet for things like StackOverflow, Google, or outside communication.

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 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.

If you aren't a novice team, go to Problem 6!

1. Novice Problem - Philadelphia Eagles

Like any red-blooded American, you love the great gladiatorial game known as football. And there is no better team in the land than the Philadelphia Eagles!

As a quick refresher, football fields are 100 yards long. Football teams have four opportunities when on offense, known as *downs*, to advance the ball a net of ten yards forward. If one does not advance ten cumulative yards in four downs, your opponents get the ball. This is called a *turnover*. If a team does advance the ball a net of ten yards in four downs, a new *first down* is issued, and the team has four more downs to advance the ball another ten yards.

If a team successfully travels the full 100 yards, they score a *touchdown*. If a team ever falls behind its own 0 yard line (its starting position), a *safety* is issued to the other team.

The duration of a series of offensive plays is known as a *drive*. A drive ends when a turnover happens, or when a team scores.

We have some records of the yards gained and lost in each of the plays of some drives, but we need your help reconstructing what actually happened. Given an array of yardages, each one representing a different play of the drive, return whether:

- there was a turnover (“TURNOVER”)
- there was a touchdown (“TOUCHDOWN”)
- there was a safety (“SAFETY”)
- the drive is still in progress (“IN PROGRESS”)

Assume that the first of those events that happens stops the drive — i.e., after a touchdown is scored, return that a touchdown is scored; do not worry about continuing to read downs. The Eagles always start on the 0 yard line — i.e., they have 100 yards to go to get a touchdown (this is just to make it fair for the other teams, of course).

Sample Input:	Sample Output:
1). [5, 2, 11, 3, -2, 0]	IN PROGRESS
2). [7, -3, -5, 8, 9, 0, 3, 3, 4]	SAFETY
3). [3, 2, -4, 11, 6, 5, -2, 2, 16, -5, 3, 0, 2, 11, 4, -3, 1, 6, 8]	TURNOVER

2. Novice Problem - Every which way but West...or East

Philadelphia has a very convenient structure for laying out and naming its streets. All of the numbered streets go North/South and the named streets go East/West. In addition, most of the North/South streets are one way.

While trying to get around the city, you find yourself wondering which way each street goes. Naturally, you decide to write a program to determine which way a street goes, given the street's number. This could require you to write cases for each of the streets, but conveniently enough, there is a pattern to the way streets go in the city.

The general format of this pattern is that within some set of streets, even streets go one way and odds go the other. Here are the rules for which way streets go. We also are only considering 1st through 69th street.

- East of Broad (1st-13th): even go south, odd go north
- West of Broad (15th-32nd): even go north, odd go south
- West Philly (33rd-45th): even go south, odd go north
- West Philly (≥ 46 th): two-way

Of course, there are a few exceptions to these rules:

- 1st and 14th do not actually exist as street names -- they're called Front and Broad. We'll ignore this and pretend they do.
- Broad (14th), 25th, 38th, 41st, and 42nd are all two-way.
- 24th and 59th go south.
- 58th goes north.

The stub function has as its only argument an integer that holds the street number. You should return a string indicating which direction the street goes:

- "N" for north
- "S" for south
- "N/S" for both ways
- "N/A" if the street does not exist (i.e it is < 1 or > 69)

Sample Input:	Sample Output:
1). 14	N/S
2). 9	N
3). 18	N

3. Novice Problem - Cheesesteak Conundrum

Like many of the residents of Philadelphia, you make a living selling cheesesteaks. You notice that you aren't making as much money as you think you could be, and decide to use your computer science skills to improve your business plan.

Your cheesesteaks have three basic ingredients: cheese, steak, and rolls. Each cheesesteak uses one roll, and the varieties you sell are detailed in the table below:

	Standard	Double Meat	Deluxe
Cheese	2 units	2 units	3 units
Steak	2 units	4 units	3 units
Price	\$5	\$6	\$7

Each month you get a variable amount of each ingredient from your cheesesteak ingredient supplier. Your cheesesteaks are wildly popular, so you always sell out of whatever you make, but any leftover ingredients go to waste.

Given the quantity of each ingredient you get from your supplier in a month (in cheese, steak, roll order), determine the maximum number of dollars in sales you can bring in. Your supplier will always give you at least one of each ingredient, but never more than 100 of any one ingredient.

Sample Input:	Sample Output:
1). 4, 4, 2	10
2). 6, 8, 3	16
3). 7, 7, 3	17

4. Novice Problem - Middle Father

Philadelphia is famous for its historical past and its connections to the founding of the country. Most famously, the Second Continental Congress met here and, among other things, drafted and signed the Declaration of Independence. Much is known about the lives of the Founding Fathers, but you, as a budding computer scientist, are interested in the more quantifiable qualities. Specifically you are interested in their ages.

You are trying to find a Founding Father with an age such that there are the same number of Founding Fathers that are older than him as there are Founding Fathers that are younger. Since there is not a clear definition of who exactly is a Founding Father, you decide to write a program that generalizes this problem to any subset of the Founding Fathers.

You are provided with a class for representing a Founding Father with two fields - their name, and their age in years. Then, given an array of an odd number of Founding Fathers, return the name of the Founding Father who is older than the same number of founding fathers that he is younger than.

We guarantee that this Father is unique, that there are an odd number of Founding Fathers, and that there will not be more than 51 Founding Fathers.

Sample Input:	Sample Output:
1). {"Franklin" 76, "Jefferson" 56, "Adams" 57}	Adams
2). {"Franklin" 76, "Jefferson" 56, "Adams" 57, "Lee" 60, "Hancock" 75}	Lee
3). {"Franklin" 76, "Jefferson" 56, "Adams" 57, "Morris" 56, "Rutledge" 76}	Adams

5. Novice Problem - Phanatic Phabric

You have been put in charge of creating replica costumes of the famed Phillie Phanatic for all the people on your baseball team. Before starting to make the costumes, you will need to order enough of the special Phanatic green hairy fabric to put on the outside of the costume.

Since the shade of green in the Phanatic costume is copyrighted by the Phillies, you must get it through the black market, making it very expensive. Since you also have almost no money, you decide that instead of just generalizing how much fabric you will need for each costume, you come up with a formula that will compute more accurately the amount of fabric needed for each costume given the height and weight of the person who will wear it. This formula is given by:

$$S(h, w) = \frac{1}{100000}(39w^2h + 80w^3)$$

where h is the height in inches and w is the weight in pounds of the wearer (just pretend that the coefficients have constants such that the units work out). Given the heights and weights of all the people in the group, compute the total amount of fabric you must buy.

You are given an array where every two numbers signify a height and a weight. For example, Sample Input #1 represents two people: one of height 60" and weight 160 pounds, and one of height 69" and weight 145 pounds.

You should return the total amount of fabric needed to make all the costumes, rounded down to the nearest integer. The output is guaranteed to fit inside a Java int.

Sample Input:	Sample Output:
1). [60, 160, 69, 145]	6879
2). [70, 190, 55, 101, 40, 70, 56, 120]	9561
3). [10, 38]	49

6. Intermediate Problem - Philadelphia Experiment

In 1943, the US Navy allegedly performed mysterious experiments in the Philadelphia harbor. The destroyer USS *Eldridge* was said to become invisible and teleport to North Carolina; the experiment was based upon Albert Einstein's unified field theory. Einstein had not released this theory to the public, since "mankind was not ready for it," making this experiment all the more controversial. While the US Navy denies this ever occurred, a shady figure named Carlos Allende insists it all happened.

The U.S. Navy needed a backdoor password to turn the cloaking device on and off at will. They also needed to make sure that this password was not only impossible to guess for humans, but also impossible for a computer to brute-force attack. Furthermore, they needed to be able to transmit this password to other Naval officers that weren't stationed in Philadelphia. [1]

Now that we're in 2014, the ship is de-commissioned, but you and some friends snuck onto the *Eldridge* late at night and stumbled into the cloaking room. Its computer asked for a password to cloak the ship, and you found a sticky note with the formula:

$$y = g^x \pmod{p}$$

and a string of garbled characters. You suspect that rotating the ASCII values of the characters up by x will give you the password. [2]

Given y , g , p , and the encrypted password, you must first compute x , and then use it to decipher the garbled characters. Can you crack the password? Assume that both the encrypted password and plaintext password contain only capital letters. Assume that there is a solution to the discrete log (i.e. working value of x) that is less than p . All numeric inputs will be between 1 and 150.

Sample Input:	Sample Output:
1). 80,5,105,GZRDNZGGDN	LEWISSELLIS
2). 15,6,51,ZQAUXQHJ	JAKEHART
3). 23,7,29,JWPAYDKZKOD	NATECHODOSH

[1] This transmission problem is legitimately solved via the Diffie-Hellman algorithm, which the Philadelphia Experiment question hints at. The algorithm's security comes from the fact that it is (believed to be) computationally infeasible to calculate a discrete logarithm for sufficiently large numbers.

[2] "mod" means modulo, and "mod p " is just fancy talk for "take the remainder of this when divided by p ", and Java's `%` operator does this computation for you. If you determine that $x = 2$, then you would decode $A \rightarrow C$, $B \rightarrow D$, ..., $Z \rightarrow B$.

7. Intermediate Problem - Counting Costs

You are an employee of the Philadelphia Museum of Art tasked with buying new pieces of art. Each year, you get a budget to spend and are told to spend it on art at your discretion. Since you don't really know anything about art, you decide that the best way to complete your task is to just completely spend the budget you've been given.

The art world is a strange place, and each year there end up being N price brackets for all of the art on sale. Every piece of art is placed into one of these brackets, and the price of any two pieces of art in the same bracket is the same. If the brackets are \$4k, \$3k, and \$1k, then every piece of art on sale that year has one of those three prices.

As you start to plan what art you want to buy, you quickly realize that there are many different ways to spend the money. For example, if you had a budget of \$12k, you could buy 4 \$3k pieces, or 3 \$4k pieces, or just 12 \$1k pieces, or some combination in between. The number of configurations quickly overwhelms you, so you decide to write a program to find the number of ways you can spend your budget. Assume, of course, that there is enough art in the world for you to buy as many pieces from each bracket as your heart could desire.

Given a budget and an array of price brackets (all integers, in thousands of dollars - a budget of 12 is really \$12,000), calculate how many different ways you could spend your entire budget exactly, no more, no less. All budgets will be at most \$1,000,000 (1000), and there are never more than 15 different price brackets.

Sample Input:	Sample Output:
1). 12, 3, 4, 3, 1	11
2). 4, 3, 5, 3, 8	0
3). 8, 2, 5, 1	2

8. Intermediate Problem - SEPTA Scheduling

You've decided to go for a ride on the Southeastern Pennsylvania Transportation Authority's subway (which is colloquially referred to as various things including "the El", "the MFL", "the Blue Line", or even just "SEPTA"). You need to figure out how long you'll be away from home. This total travel time includes time waiting for the subway, time on the subway, time spent at your destination, time waiting for the subway home, and time on the subway home.

The parameters, then, are what time you leave home, where you're going, how much time you need to spend there, and the subway timetable, and you want to find out what time you'll get home.

Fortunately, the trains are predictable: every half hour on the hour, one train leaves the 2nd Street station headed west, and one leaves the 69th Street station headed east. This means a train headed in either direction arrives at the 40th Street station (where you live) at 10 and 40 minutes after each hour. Make the optimistic assumption that the trains are never delayed and never deviate from this cycle.

You are given the subway schedule - a list of station names, and how many minutes it takes a train to get there from 40th Street. The three parameters are:

- Your leaving time in military notation (the first two digits are the current hour, and the last two are the minutes)
- The street number of the stop you're going to
- The number of minutes you need to spend at your destination

You should return a string representing what time you get back to 40th Street (also in military notation).

8. SEPTA Scheduling (continued)

Station Name	Travel Time (from 40th)	Station Name	Travel Time (from 40th)
69th Street	10 minutes	30th Street	3 minutes
63rd Street	7 minutes	15th Street	5 minutes
60th Street	6 minutes	13th Street	6 minutes
56th Street	4 minutes	11th Street	7 minutes
52nd Street	3 minutes	8th Street	8 minutes
46th Street	2 minutes	5th Street	9 minutes
34th Street	2 minutes	2nd Street	10 minutes

Sample Input:	Sample Output:
1). "1201" 15 20	1240
2). "1200" 15 70	1340
3). "0416" 63 100	0640

9. Intermediate Problem - Philly Boxing

Among its many beloved sports, Philly is known for its affinity to boxing. Champions such as Meldrick Taylor, Bernard Hopkins, Sonny Liston, and Joe Frazier called the city home, not to mention legendary character Rocky Balboa, famous for running the steps of the Philadelphia Art Museum. However, boxing has long suffered from questionable judging, so we need your help with telling who won a boxing match.

Boxers have three basic moves - jab, hook, and uppercut - and a boxing match is determined by each boxer's fight strategy, represented as a list of moves the boxer chooses to use. Each boxer starts with some amount of health, and the loser is whoever's health hits 0 first, or whoever has lower health at the end of the match.

Jabs, hooks, and uppercuts do 1, 2, and 3 base damage respectively, and the moves match up somewhat like rock, paper, scissors: jabs beat uppercuts which beat hooks which beat jabs. Hooks do damage against any move, but an uppercut does no damage against a jab. The damage taken by each player in each move matchup is shown in this table:

Move	Jab	Hook	Uppercut
Jab	1, 1	2, 1	0, 1
Hook	1, 2	2, 2	3, 2
Uppercut	1, 0	2, 3	3, 3

Finally, each boxer has a unique set of skills determining how good they are at each move, in the form of multipliers which affect how much damage the boxer does with each move. For example, if a boxer has a jab multiplier of 1.2 and an uppercut multiplier of 0.8, his jabs will do 1.2 damage and his uppercuts will do only 2.4.

Given two boxers, each with a name, health, move strengths, and a fight strategy, return the name of the winning boxer.

Sample Input:	Joe 20 1.0 1.5 0.5 H H H H H H H Rocky 50 1.0 1.0 1.0 J J J J J J J	Foreman 10 1.0 0.5 1.5 U U U U U U U Ali 10 1.5 1.0 0.5 J J J J J J J	Joe 20 1.1 1.0 0.9 H H J U U H Don 20 1.0 1.1 0.9 J U H H J U
Sample Output:	Rocky	Ali	Don

10. Intermediate Problem - A Few Good Numbers

Every student at Penn is given a unique ID represented by a string of digits of length 8. Since you are a student of number theory, you naturally think that a good Penn ID would be one that satisfied an interesting property. The property that intrigues you the most, of course, is whether or not an ID can be written as a string of Fibonacci numbers.

For example, the ID 11235813 would be a good ID because 1, 2, 3, 5, 8, and 13 are all Fibonacci numbers (but the numbers don't need to be in increasing order). Your interest in the problem soon extends beyond just ID numbers and now you wish to find out if any number is a "good" number. Doing this by hand quickly becomes difficult, so you decide to write an algorithm that determines if a number is good or not.

Given a string that contains the decimal representation of a number, determine whether it is "GOOD" or "BAD". The number will have less than 500 digits and will not contain any Fibonacci greater than the size of a Java int.

Sample Input:	Sample Output:
1). 11235813	GOOD
2). 1382105	BAD
3). 14413318	GOOD

Novices, stop here!

11. Advanced Problem - Levenshtein Distance

Philadelphia is known for its unique dialect. Locals use words like “hoagie” and “jawn.” As a budding linguist, you would like to quantify just how different Philadelphia slang is from other regional synonyms. You will therefore compute the Levenshtein distance between two strings as a measure of this regional difference.

The Levenshtein distance between two strings is the minimum number of modifications to one string needed to make it equal the other string. Here, we define a modification to mean any one of:

- an insertion: “hat” -> “heat” (inserting a new character to the string)
- a deletion: “hate” -> “hat” (removing a character already present in the string),
- a substitution: “dave” -> “date” (changing a letter in the string to any other letter).

Note that insertions and deletions can happen at the beginning or end of the string, not just in the middle.

Given two strings, calculate the Levenshtein distance between them.

Sample Input:	Sample Output:
1). [“hoagie”, “hero”]	5
2). [“salty”, “upset”]	5
3). [“sprinkles”, “jimmies”]	6

12. Advanced Problem - Lancaster County Corn Maze

Having had so much fun thus far in Philadelphia, you decide to explore the rest of Eastern Pennsylvania. After a short bus ride, you arrive at the world famous Lancaster County, home of the Pennsylvania Dutch. After stuffing yourselves at the all you can eat buffet, you and your friends decide to try navigating corn mazes. However, before you start, your friend worries that the evil corn farmer may have made a maze without a solution.

Your mission, then, is to design an algorithm which decides whether a maze is solvable. The maze will be represented as a matrix (2D array) of chars.

- '#' represents a wall
- '-' represents a space

Each maze starts at the top-left open space (i.e. `maze[0][1]`) and ends in the lower-right open space of the maze (i.e. `maze[maze.length - 1][maze[0].length - 2]`). (Note that the column indices are off by one because the very top-left and bottom-right spaces must be wall spaces.)

You can be guaranteed that the start and finish spaces will be open ('-') and that, if a maze has a solution, it has only 1 solution (you can't walk in circles (or diagonally, for that matter)). Finally, no side of the maze will be longer than 45 characters. Return true if you can make it through the maze (i.e. there exists a path of '-' characters from the start to the end), and false otherwise.

Sample Input:	<pre>#-##### #-----# #####-# #-#---# #-#-### #-----# #####-#</pre>	<pre>#-##### #-----# #####-# #-#---# #-##### #-----# #####-#</pre>	<pre>#-##### #-#-----# #-#-#####-# #-#-#---#-# #-###-#-#-# #---#-#-#-# ###-#-#-#-# #-#-#-#---# #-#-#-###-# #-----#-#-# #####-#-#</pre>
Sample Output:	True	False	True

13. Advanced Problem - Find the nearest Wawa

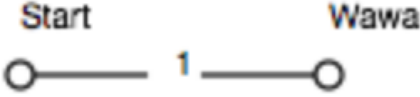
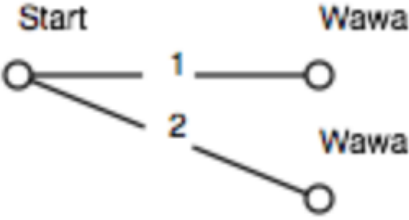
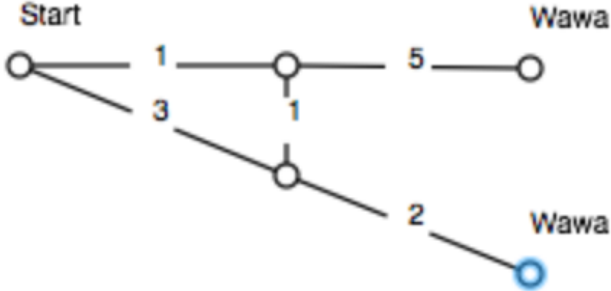
Among Philly's famous institutions, the favorite among college students is Wawa, a 24 hour convenience store with many locations around the city. You've been hard at work programming all day, so you want to grab something to eat. Unfortunately, finding a Wawa is not always an easy task, and you don't want to walk any farther than you must, so clearly you need to write a program to help find the nearest Wawa.

You're given a store as a starting place. Each store has:

- a String name
- a boolean isWawa
- an int distanceFromStart (initialized to infinity, because we do not yet know how far away it is)
- a Set of ConnectedStores (a list of stores you can get to directly from this store, plus the distance to each store).

Your goal is to find the distance to the nearest wawa (i.e. the length of the shortest path from your starting store to a store with isWawa == true).

There will be no more than 20 stores, and at most 50 paths between stores.

Sample Input:	Sample Output:
1). 	1
2). 	1
3). 	4

14. Advanced Problem - How many diverse composites?

You were visiting Penn, and you stumbled into a professor's office. He specializes in Computational Number Theory, so he gave you this problem:

We call a number a “diverse composite” if it is a positive integer divisible by at least four distinct primes less than 100. The smallest diverse composite is 210 ($2 \cdot 3 \cdot 5 \cdot 7$), and some other examples include:

- 5775 ($3 \cdot (5^2) \cdot 7 \cdot 11$)
- 5698 ($2 \cdot 7 \cdot 11 \cdot 37$)
- 21210 ($2 \cdot 3 \cdot 5 \cdot 7 \cdot 101$)

Notice that a number can still be a diverse composite if it has prime factors over 100, as long as it also has at least four prime factors less than 100. 12120 ($(2^3) \cdot 3 \cdot 5 \cdot 101$) is not a diverse composite because it has only three distinct prime factors less than 100, even though one of those factors is repeated.

Given a Java long n , find the number of diverse composites less than n . The upper bound for n in our tests is 10^{10} , but rumor has it there's a way to do it for even bigger values of n !

Sample Input:	Sample Output:
1). 500	5
2). 1000	23
3). 10000	811

Note: If you are unfamiliar with Java's “long” type, they're like an “int” with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
long my_long = 1000000000000L;
```

Otherwise, they function just like an “int” type.