3rd Annual

Brookfield Computer Programming Challenge

2019

Problem Set

Problems:

1. 99 Red Balloons (Balloons)
2. Frenemies (Frenemies)
3. Frosty the Milkshake (Frosty)
4. Linus and Snoopy (Linus)
5. Planes (Planes)
6. Special Snowflakes (Snowflakes)
7. Three (Three)
8. Windows 10 (Windows)

*All problems have a strictly-enforced 5-second time limit. Modern computers can do roughly 10^8 operations per second in Java or C++ (Python is slower). Make sure that your solution won't take more than 10^8 operations for all input following the input requirements.

**The word 'integer' in this problem set means the mathematical definition of integer. It may be larger than 10^9 and may not fit in an 'int' data type. In some cases, it may not even fit in a 'long' data type (for instance, in Windows 10, the integer to output can be much more than 18 digits).

99 Red Balloons (Balloons)

You and I in a little toy shop buy a bag of balloons with the money we've got.
We set them free at the break of dawn 'till one-by-one they were gone.
Back at base, bugs in the software flash the message "Something's out there!"

You've just been hired as the United States' lead programmer for the detection of alien invasions. In order to help prevent the software from causing World War 3, the NSA has informed you that someone has begun releasing one balloon per second from break of dawn (time 0) until time **n**-1 (**n** balloons in total). Each balloon moves up one foot per second, so it will be at height x exactly x seconds after it was released. Given this information, you to determine exactly how many balloons are in the height range **L**...**H** at time **T**.

Good luck, sargent! The future of the world depends on you!

Input:

- Input will consist of 4 integers: **n**, **L**, **H**, and **T**, describing the number of balloons released, the height range of interest, and the time of interest.

$1 \leq \mathbf{n} \leq 99$
$1 \leq \mathbf{L} \leq \mathbf{H} \leq 10^6$
$1 \leq \mathbf{T} \leq 10^6$

Output:

- Output a single integer: the number of balloons in the height range [**L**, **H**] at time **T**.

Sample Input	**Sample Output**
5 3 10 3	1
99 1 1000000 500000	99
99 1 100 200	0
5 3 5 7	3

In the first sample, there is exactly one balloon in the height range [3, 10] at time 3; it is at height 3.

Frenemies (Frenemies)

We look good in hats, long tails, and spats. When we hit the town together, Baby, we know where it's ats.
Cuz we're frenemies; we like disliking one another. Yes we're frenemies--he's like my least favorite brother!
You and I, we're not enemies or friends, we're just frenemies till the end!

The Organization Without a Cool Acronym (OWCA) has begun organizing meetups around the tri-state area to encourage residents to make more frenemies. The great thing about these meetups is that whenever two people talk to each other, they instantly become frenemies if they aren't already. And the great thing about frenemies is that being frenemies is transitive; the frenemy of your frenemy is also your frenemy. So, if Alice and Bob are already frenemies and Perry talks to Alice, then Alice, Bob, and Perry will all be each other's frenemies afterwards!

OWCA needs your help determining how many pairs of frenemies there are at each point during the meetup. Initially, there are no frenemies in the tri-state area. Over the course of the meetup, there will be **m** times in which two of the **n** people will talk to each other and become frenemies if they aren't already. Each of these **m** times, you need to report how many pairs of people are now frenemies.

Input:

- The first line contains two integers: **n**, the number of people, and **m**, the number of times two people will talk.
- **m** lines follow, each containing two unique integers between 1 and **n**, describing the two people who talk. The same pair of people might talk multiple times across the event.

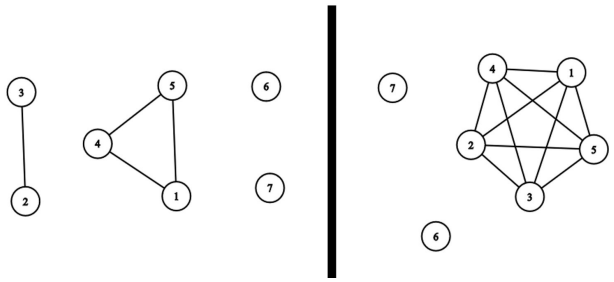
$2 \leq \mathbf{n} \leq 3*10^5$
$1 \leq \mathbf{m} \leq 3*10^5$

Output:

- After each pair of people talk, output the number of pairs of people that are frenemies. Note that this number can be bigger than 10^9. Output each number on a new line.

Sample Input	**Sample Output**
7 5 1 4 4 5 2 3 3 1 1 5	1 3 4 10 10

All pairs of frenemies after the 3rd and 4th conversations



Frosty the Milkshake (Frosty)

Frosty the Milkshake was a jolly happy soul,
With a 10-inch spoon, and a paper cup, and a clown maraschino nose.

Frosty is on his journey to the North Pole to meet up with his best friend, the Jingle Bell Rock. However, the sun is pretty hot today and Frosty doesn't want to melt and get all sticky. Thankfully, all **m** of Santa's elves have offered to let Frosty cool off in their freezers. Frosty starts off at **D** (**D** ≤ 0) degrees, but he can wait in a freezer (either his own or an elf's) for one minute to get one degree cooler. If Frosty's temperature ever gets above 0 degrees though, he will melt! Frosty starts at position 0 where he has his own freezer and wants to reach Jingle Bell Rock's house at position **n**. On a turn, Frosty can move one position in either direction and get one degree warmer, or, if he is at a space that has a freezer, can cool off in the freezer for one second to become one degree cooler.

How many turns does Frosty need to reach Jingle Bell Rock's house?

Input:

- The first line will contain three integers **n**, **m**, and **D**: the position of the Rock's house, the number of elves, and Frosty's starting temperature.
- The second line contains **m** unique integers in increasing order between 1 and **n**-1; the location of each elf's freezer. (Frosty has his own freezer at position 0, and Jingle at **n**)

2 ≤ **n** ≤ 1000 0 ≤ **m** ≤ **n**-1 -1000 ≤ **D** ≤ 0

Output:

- Output a single line: the minimum number of turns required for Frosty to reach Jingle Bell Rock's house.

Sample Input	**Sample Output**
4 1 0 3	8
10 2 -5 6 8	15

In the first sample, Frosty can wait in his own fridge for 3 turns, walk forward for the next 3 turns, wait in the first elf's house for 1 turn, and then walk forward for 1 turn, arriving in 8 turns.

Linus and Snoopy (Linus)

Linus and Snoopy have prepared a terrific duet for the holiday season. They have each memorized the same sequence of **n** notes (numbered 1...**n**) with Linus playing them on the piano and Snoopy on the bass guitar. Everything was going great... at least until the commercial break.

You see, Linus and Snoopy forgot what part of the song they were at when the commercial started, so they both decided to start playing at a random point in the song when the show continues. They will continue playing one note per beat until either they play different notes and realize they messed up, or until one of them reaches the end of the song and puts down their instrument. So if the song was "ABA" and both Linus and Snoopy both start at position 1, they would play all 3 beats. But Linus started at 1 and Snoopy at 2, Linus would play A and Snoopy B, so they would realize they messed up and stop after playing 1 beat. The network (the Best Christmas Productions Channel) would really like to know how much longer they should expect the duet to last.

That's where you come in. Given the notes in the song, you need to determine the expected number of beats that Linus and Snoopy will play. Print this as a reduced fraction in the form p/q.

Input:

- The first line will contain a single integer **n**, the number of beats in the song.
- The second line will contain **n** musical notes (uppercase 'A' - 'G') representing the note played at each beat.

$1 \leq \mathbf{n} \leq 3000$

Output:

- Output a fraction in the form "**p/q**" where and p and q are relatively prime (that is, they have a gcd of 1). This fraction should be the expected number of notes played in the song. If the expected number of notes is a whole number, print "**p/1**".

Sample Input	**Sample Output**
3 ABA	4/3
7 ABACABA	80/49

In the first sample, if they start at (1, 1), they will play for 3 beats, (1, 2) for 1, (1, 3) for 1, (2, 1) for 1, (2, 2) for 2, (2, 3) for 1, (3, 1) for 1, (3, 2) for 1, and (3, 3) for 1. The expected number of beats is (3+1+1+1+2+1+1+1+1)/9 = 12/9 = 4/3

Planes (Planes)

As a self-proclaimed computational geometry enthusiast, you love planes of all kinds: Cartesian planes, woodworking planes for planing, plains, and even airplanes. There is one thing you don't really like about airplanes though, and that is long layovers. Thankfully, the Belarusian Carrying by Plane Company (BCPC) has come to the rescue by introducing several new flights to minimize the number of layovers you might need.

The BCPC has **n** airports, and currently has **m** flights where each flight connects two airports and goes in both directions. It plans to add **n** new flights (you can choose what these **n** new flights are) to have **n**+**m** flights in total. The BCPC would like it so that, after adding **n** more flights it is possible to travel from any airport to any other airport by taking one or more flights. In addition, they would like to minimize the number of flights needed to travel between the pair of airports that are farthest apart (where the pair that is 'farthest apart' is the pair that requires the most flights to travel between).

If the BCPC creates new flights optimally, what is the maximum number of flights required to travel from any airport to any other airport?

Input:

- The first line will contain **n** and **m**, the number of airports and flights to add, and the number of existing flights.
- The next **m** lines will each contain two distinct integers between 1 and **n** describing the airports on this existing flight. Some of the existing flights might be the same.

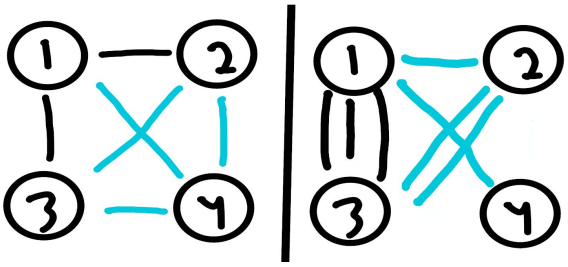
$2 \leq \mathbf{n} \leq 10^5$ $0 \leq \mathbf{m} \leq 10^5$

Output:

- Output a single integer: the minimum **x** so that, after the addition of **n** new flights, you can travel between any pair of airports using no more than **x** flights.

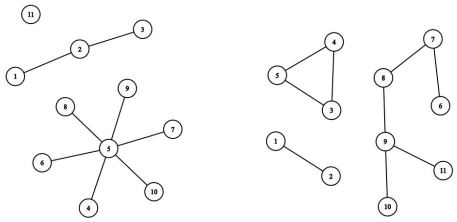
Sample Input	**Sample Output**
4 2 1 2 2 3	1
4 3 1 3 3 1 1 3	2

Possible solutions to samples 1 and 2. Black edges indicate existing flights, and blue edges indicate added flights.



Special Snowflakes (Snowflakes)

A connected component of a graph is called a snowflake if it contains exactly one node that has a degree[1] not equal to 1.



Left: Three snowflakes. Right: Three non-snowflakes.

A snowflake is special if there are no other snowflakes in the graph that are the same as it. Two snowflakes are the same if they have the same number of nodes.

Given a graph, how many special snowflakes does it contain?

Input:

- The first line will contain two integers: **n** and **m**: the number of nodes and edges.
- **m** lines follow, each containing two unique integers in the range 1...**n**, describing an edge

$2 \leq \mathbf{n} \leq 100$ $0 \leq \mathbf{m} \leq 100$

Output:

- Output a single integer, the number of special snowflakes in the graph

Sample Input	**Sample Output**
11 8 1 2 2 3 4 5 6 5 7 5 8 5 9 5 10 5	3
4 1 1 2	0

[1] The degree of a node is the number of edges connected to it.

Three (Three)

Hooray! It's time for the obligatory problem concerning the random mathematical properties of an arbitrary number! And since this is the third annual Brookfield Computer Programming Challenge, this year's number is 3.

Three is a pretty unique number. It's pretty much equal to pi, and also to e. It's also a fibonacci number, and a prime! Not only is it prime, it's a Mersenne prime (one less than a power of 2), Fermat Prime (one more than 2 to a power of 2), a factorial prime (one more than 2!), and the only prime triangular number! Speaking of triangles, for any **n** ≥ 3, there exists a polygon with n sides and **n** corner points. We could give you a proof for this, but we think it would be more fun if you proved it to us by writing a program.

Given an integer **n** ≥ 3, output **n** points that form a polygon when the first point is connected to the second, the second to the third, and so on, and then the last point to the first. Your polygon:

- Must not be self intersecting: no two segments on your polygon should not intersect, except for adjacent sides which can intersect at their single endpoint of course.
- Must not contain two consecutive parallel sides (i. e. corners of exactly 180 degrees).
- Must only contain points on integer coordinates with absolute value at most 10^9.
- Can be listed in either clockwise or counter-clockwise order.

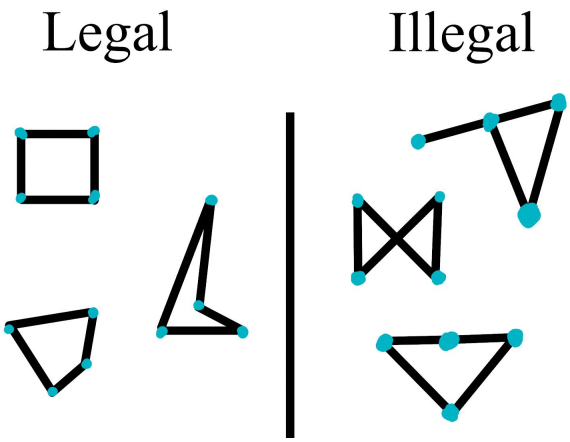
Input:

- The first line will contain a single integer **n**, the number of corner points.

$3 \leq \mathbf{n} \leq 1000$

Output:

- Print **n** lines, each containing two integers: the x and y coordinates of the next point.



Sample Input	**Sample Output** (Any valid output will be accepted)
4	0 0 3 0 1 1 0 3

Windows 10 (Windows)

Many of you are probably running Windows 10 on your computer right now. The reasoning behind the name for Windows 10 is quite interesting. As you may know, Windows 10 is really the 9th major version of Windows (1, 2, 3, 4, XP, Vista, 7, 8, 10). However, the company chose to skip the name Windows 9. Some believe this was because programs to detect if the operating system was too old would check if the OS name starts with "Windows 9" (expecting Windows 95 or Windows 98), and this code would behave unexpectedly if the new name started with Windows 9.

However, the real reason likely has more to do with marketing. You see, consumers generally view the number 9 as unsatisfactory; it is almost a perfect 10, but not quite. Companies would like to skip all naming iterations that contain any unsatisfactory digits. In the case of Windows 9 and the iPhone 9, the only unsatisfactory digit is 9, but it is possible that up to 8 of the 10 decimal digits are seen as unsatisfactory someday. Given which digits are unsatisfactory, find the nth positive number that contains no unsatisfactory digits, so that companies can easily find out what the marketing name for their nth product will be.

Input:

- The first line will contain an integer **d**, the number of unsatisfactory digits.
- The second line will contain **d** distinct integers in increasing order, representing the digits customers find unsatisfactory. (These **d** digits will all be positive; 0 is never unsatisfactory)
- The third line will contain an integer **n**, the iteration of the product.

$1 \leq \mathbf{d} \leq 8$ $1 \leq \mathbf{n} \leq 10^{18}$

Output:

- Output a single integer: the **n**th positive number that contains no unsatisfactory digits. Note that this number can have more than 18 digits.

Sample Input	**Sample Output**
1 9 9	10
2 1 8 10	23

In the second sample, the first 10 iterations would be: 2, 3, 4, 5, 6, 7, 9, 20, 22, and 23.