Problem Statement

The inchworm is a creature of regular habits. She inches forward some distance along the branch of a tree, then stops to rest. If she has stopped at a leaf, she makes a meal of it. Then she inches forward the same distance as before, and repeats this routine until she has reached or passed the end of the branch.

Consider an inchworm traveling the length of a branch whose leaves are spaced at uniform intervals. Depending on the distance between her resting points, the inchworm may or may not be able to eat all of the leaves. There is always a leaf at the beginning of the branch, which is where the inchworm rests befor setting out on her journey.

You are given three int values that specify, in inches: the length of the branch; the distance traveled by the inchworm between rests; and the distance between each consecutive pair of leaves. Given that the inchworm only eats at rest, calculate the number of leaves she will consume.

Definition

Class:	Inchworm
Method:	lunchtime
Parameters:	int, int, int
Returns:	int
Method signature:	int lunchtime(int branch, int rest, int leaf)

(be sure your method is public)

Notes

- The inchworm starts by gobbling up the leaf at the beginning of the branch.
- If there is a leaf at the end of the branch, the inchworm eats it only if it falls at a resting point.

Constraints

- **branch** is between 1 and 1000000 (one million), inclusive
- **rest** is between 1 and 1000, inclusive
- **leaf** is between 1 and 1000, inclusive

Examples

0)

```
11
2
4
Returns: 3
```

Leaves grow 0, 4, and 8 inches from the beginning of the branch. The inchworm eats them all.

1)

```
12
6
4
Returns: 2
```

The inchworm misses the leaves growing at distances 4 and 8, but eats those at 0 and 12.

2)

```
20
3
7
Returns: 1
```

The inchworm eats only the leaf at the outset.

3)

```
21
7
3
Returns: 2
```

The inchworm eats the leaves at the beginning and end of the branch.

4)

```
15
16
5
Returns: 1
```

The inchworm eats only the leaf at the outset.

5)

```
1000
3
7
Returns: 48
```

6)

```
1000
7
3
Returns: 48
```

Problem Statement

There are **N** friends sitting in a circle, numbered clockwise in increasing order from 1 to **N**. At the beginning of the game, player 1 receives a ball. The players take turns passing the ball to each other. At the beginning of the game and before each next pass the following actions are performed. If the player with the ball has already received the ball **M** times, the game is over. Otherwise, if the player has received the ball *p* times, he'll pass the ball directly to the person **L** places to his left if *p* is even, or **L** places to his right if *p* is odd (see examples for clarification). Given **N**, **M** and **L**, return the number of times that the ball is passed.

Definition

Class:	ThrowTheBall
Method:	timesThrown
Parameters:	int, int, int
Returns:	int
Method signature:	int timesThrown(int N, int M, int L)

(be sure your method is public)

Constraints

- **N** will be between 3 and 50, inclusive.
- **M** will be between 1 and 50, inclusive.
- **L** will be between 1 and **N**-1, inclusive.

Examples

0)

```
5
3
2
Returns: 10
```

First, player 1 gets the ball. Since he has held the ball 1 time, he passes the ball to player 4, who is two places to his right. This is player 4's first time holding the ball, so he gives it to player 2, who passes it to player 5. Player 5 then passes the ball to player 3, who passes it back to player 1. Since player 1 has now held the ball 2 times, he passes it to player 3, who passes it to player 5, who then passes the ball to player 2. Finally, player 2 passes the ball to player 4, who then passes it to player 1. Player 1 has now held the ball 3 times, and the game ends.

1)

```
4
1
3
Returns: 0
```

Here, the ball is never passed.

2)

```
10
3
5
```

```
Returns: 4
```

3)

```
15
4
9
```

```
Returns: 15
```

Problem Statement

Given a String, **input**, with up to 50 characters, find the length of the longest substring that appears at least twice (non-overlapping) in **input**. If no substring appears twice, non-overlapping, return 0.

Strings are case sensitive, and only upper case letters and lower case letters are allowed in **input**.

For example, in the string "ABCDEXXXYYYZZZABCDEZZZYYYXXX" the longest substring which appears at least twice is "ABCDE". These two substrings do not overlap so you would return 5.

Definition

Class:	Reppity
Method:	longestRep
Parameters:	String
Returns:	int
Method signature:	int longestRep(String input)
(be sure your method is public)	

Notes

- We are looking for subSTRINGS not subSEQUENCES. All of the elements of a substring appear consecutively in the original string. In other words, the substring can be formed from the original string by deleting zero or more characters from the begining and deleting zero or more characters from the end, but NO deletions from the middle are allowed.

Constraints

- **input** will contain between 1 and 50 characters inclusive.
- Each character in **input** will be between 'a' and 'z' inclusive or between 'A' and 'Z' inclusive.

Examples

0)

```
"ABCDEXXXYYYZZZABCDEZZZYYYXXX"
Returns: 5
```

The example from above.

1)

```
"abcdabcdabcdabCD"
Returns: 6
```

"abcdab"+"cd"+"abcdab"+"CD"

2)

```
"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy"
```

```
Returns: 25
```

3)

```
"againANDagainANDagainANDagainANDagainANDagain"
Returns: 21
```

4)

```
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWX"
Returns: 0
```

Problem Statement

The cows in Byterland are mooing loudly, annoying the residents very much. Mrs. Darcy of the villa Pemberley is planning to resolve this problem by allowing only one cow to moo. She will pick the cow whose mooing is the least offensive to all the other cows.

The farmland in Pemberley is divided into n*m squares of grassland. Each square is either empty or occupied by a single cow. When a cow at (x,y) moos, the dissatisfaction of a cow at (i,j) is equal to the square of the distance between them: $((x-i)^2 + (y-j)^2)$. The total dissatisfaction is the sum of the dissatisfaction of all the cows.

Return the minimal total dissatisfaction that can be achieved by allowing only a single cow to moo. The **farmland** will be given as a String[], where '.' characters denote empty squares, and 'C' characters denote squares occupied by cows.

Definition

Class:	MooingCows
Method:	dissatisfaction
Parameters:	String[]
Returns:	int
Method signature:	int dissatisfaction(String[] farmland)

(be sure your method is public)

Constraints

- **farmland** will contain between 1 and 50 elements, inclusive.
- Each element of **farmland** will contain between 1 and 50 characters, inclusive.
- Each element of **farmland** will contain the same number of characters.
- Each character in **farmland** will be either uppercase 'C' or '.'.
- **farmland** will contain at least one uppercase 'C'.

Examples

0)

```
{"C..",
 ".C.",
 ".C."}
```

```
Returns: 3
```

- If we set the uppermost cow to moo, the total dissatisfaction will be 2+5;
- If we set the middle cow to moo, the total dissatisfaction will be 2+1;
- If we set the bottommost cow to moo, the total dissatisfaction will be 1+5.

Certainly we will choose the second plan.

1)

```
{"CCCC",
 "CCCC",
 "CCCC"}
```

```
Returns: 26
```

2)

```
{"C"}
```

```
Returns: 0
```

3)

```
{"CCC....",
 "C......",
 "....C.C",
 ".C.CC..",
 "C......"}
```

```
Returns: 81
```

Problem Statement

Rugs come in various sizes. In fact, we can find a rug with any integer width and length, except that no rugs have a distinct width and length that are both even integers. For example, we can find a 4x4 rug, but not a 2x4 rug. We want to know how many different choices we have for a given area.

Create a class RugSizes the contains a method rugCount that is given the desired **area** and returns the number of different ways in which we can choose a rug size that will cover that exact area. Do not count the same size twice -- a 6 x 9 rug and a 9 x 6 rug should be counted as one choice.

Definition

Class:	RugSizes
Method:	rugCount
Parameters:	int
Returns:	int
Method signature:	int rugCount(int area)

(be sure your method is public)

Constraints

- **area** will be between 1 and 100,000, inclusive.

Examples

0)

```
4
Returns: 2
```

The choices are 1 x 4 (or equivalently 4 x 1) and 2 x 2.

1)

```
8
Returns: 1
```

Only 1 x 8 is available. Note that 2 x 4 has the desired area, but is not available since its width and length differ and are both even numbers.

Problem Statement

You are given a String[] **data** representing a rectangular grid where each cell contains a digit. Find the largest square in this grid that contains the same digit in all of its corner cells. The sides of the square must be parallel to the sides of the grid. If there is more than one such largest square, pick any one of them.

Return the number of cells in the square. Note that a single cell is also considered a square, so there will always be an answer.

Definition

Class:	SquareOfDigits
Method:	getMax
Parameters:	String[]
Returns:	int
Method signature:	int getMax(String[] data)

(be sure your method is public)

Constraints

- **data** will contain between 1 and 50 elements, inclusive.
- Each element of **data** will contain between 1 and 50 digits ('0'-'9'), inclusive.
- All elements of **data** will have the same length.

Examples

0)

```
{"12",
 "34"}
```

```
Returns: 1
```

All digits in the grid are different, so the biggest feasible square has only one cell.

1)

```
{"1255",
 "3455"}
```

```
Returns: 4
```

Four '5' digits form a feasible square.

2)

```
{"42101",
 "22100",
 "22101"}
```

```
Returns: 9
```

The largest square here is the 3 x 3 square that contains the digit '1' in each of its corner cells.

3)

```
{"1234567890"}
```

```
Returns: 1
```

4)

```
{"9785409507",
 "2055103694",
 "0861396761",
 "3073207669",
 "1233049493",
 "2300248968",
 "9769239548",
 "7984130001",
 "1670020095",
 "8894239889",
 "4053971072"}
```

```
Returns: 49
```

Problem Statement

Bob is playing with his ball destroyer robot. Initially, Bob has **r** red balls, **g** green balls and **b** blue balls. The robot will repeat the following 3-step program until there are no balls left:

- If there is at least one red ball available, destroy one red ball.
- If there is at least one green ball available, destroy one green ball.
- If there is at least one blue ball available, destroy one blue ball.

You are given the longs **r**, **g** and **b**. You are also given a long **k**. Find the color of the **k**-th ball (1-index based) that will be destroyed.

- If the color of the **k**-th ball to be destroyed is red, return "RED" (quotes for clarity, returned values are case-sensitive).
- If the color is green, return "GREEN".
- If the color is blue, return "BLUE".

Definition

Class:	AlternateColors
Method:	getColor
Parameters:	long, long, long, long
Returns:	String
Method signature:	String getColor(long r, long g, long b, long k)

(be sure your method is public)

Constraints

- **r**, **g** and **b** will each be between 1 and 1000000000000 (10^12), inclusive.
- **k** will be between 1 and **r**+**g**+**b**, inclusive.

Examples

0)

```
1
1
1
3
Returns: "BLUE"
```

The order in which the balls are destroyed is: Red, green and blue. The third ball was blue.

1)

```
3
4
5
4
```

```
Returns: "RED"
```

The order in which the balls are destroyed is: Red, green, blue, red, green, blue, red, green, blue, green, blue and blue.

2)

```
7
7
1
7
```

```
Returns: "GREEN"
```

3)

```
1000000000000
1
1
1000000000002
```

```
Returns: "RED"
```

Once the only green and blue balls are destroyed, all of the remaining balls will be red.

4)

```
653
32
1230
556
```

```
Returns: "BLUE"
```

Problem Statement

It is time to arrange the seating around a circular table for a party. We want to alternate boys and girls around the table. We have a list of all the attendees and their genders. Each attendee is specified by a String that consists of the name, followed by one space, followed by either "boy" or "girl".

In addition to the attendees, we need to seat the HOST (a boy) and the HOSTESS (a girl) with the HOSTESS directly across from the HOST. That means that half the attendees should be on the HOST's left, and half on his right.

Create a class PartySeats that contains a method seating that is given a String[] **attendees** that lists all the attendees and their genders. The method returns a String[] that gives the seating plan, starting with "HOST" and proceeding clockwise around the table, including all the attendees and the HOSTESS.

If there is more than one possible seating plan, return the one that comes first lexicographically. "First lexicographically" means that each successive element in the return should be chosen to be the earliest alphabetically that is consistent with a legal seating plan. If there is no legal seating plan, the return should contain 0 elements.

Definition

Class:	PartySeats
Method:	seating
Parameters:	String[]
Returns:	String[]
Method signature:	String[] seating(String[] attendees)

(be sure your method is public)

Constraints

- **attendees** will contain between 1 and 50 elements inclusive
- each element of **attendees** will consists of a name followed by a single space followed by either "boy" or "girl". There will be no leading or trailing spaces.
- each name will contain between 1 and 20 characters inclusive
- each name will contain only uppercase letters 'A'-'Z'
- no name will be "HOST" or "HOSTESS"

Examples

0)

```
{"BOB boy","SAM girl","DAVE boy","JO girl"}
Returns: { "HOST",  "JO",  "BOB",  "HOSTESS",  "DAVE",  "SAM" }
```

A girl must follow the HOST, and JO comes earliest lexicographically. Then comes a boy, and BOB is the earliest lexicographically. HOSTESS must come next so she can be opposite the HOST and then DAVE and SAM must follow in that order to honor the alternating gender requirement.

1)

```
{"JOHN boy"}
```

```
Returns: { }
```

There are more boys than girls so we cannot alternate.

2)

```
{"JOHN boy","CARLA girl"}
```

```
Returns: { }
```

There is no way to alternate gender and also have the HOST sit directly across from the HOSTESS

3)

```
{"BOB boy","SUZIE girl","DAVE boy","JO girl",
"AL boy","BOB boy","CARLA girl","DEBBIE girl"}
```

```
Returns:
{ "HOST",
  "CARLA",
  "AL",
  "DEBBIE",
  "BOB",
  "HOSTESS",
  "BOB",
  "JO",
  "DAVE",
  "SUZIE" }
```

Problem Statement

You are given two Strings **A** and **B** that have the same length and contain only lowercase letters ('a'-'z'). The distance between two letters is defined as the absolute value of their difference. The distance between **A** and **B** is defined as the sum of the differences between each letter in **A** and the letter in **B** at the same position. For example, the distance between "abcd" and "bcda" is 6 (1 + 1 + 1 + 3).

You must change exactly **K** characters in **A** into other lowercase letters. Return the minimum possible distance between **A** and **B** after you perform that change.

Definition

Class:	ChangingString
Method:	distance
Parameters:	String, String, int
Returns:	int
Method signature:	int distance(String A, String B, int K)

(be sure your method is public)

Constraints

- **A** and **B** will each contain between 1 and 50 characters, inclusive.
- **K** will be between 1 and the length of **A**, inclusive.
- **A** and **B** will contain the same number of characters.
- **A** and **B** will contain only lowercase letters ('a' - 'z').

Examples

0)

```
"ab"
"ba"
2
Returns: 0
```

The minimum distance (equal to 0) can be achieved when we change 'a' to 'b' and 'b' to 'a'.

1)

```
"aa"
"aa"
2
Returns: 2
```

We must change both letters 'a' to some other letters. Changing them to 'b' results in the smallest distance.

2)

```
"aaa"
"baz"
```

```
1
Returns: 1
```

3)

```
"fdfdfdfdfdsfabasd"
"jhlakfjdklsakdjfk"
8
Returns: 24
```

4)

```
"aa"
"bb"
2
Returns: 0
```

Problem Statement

You are downloading some files from the Internet, and you want to know how long it will take until they are completely downloaded.

For each download, you are given its current speed (in KB/s) and remaining time based on that speed (in seconds). The sum of all the speeds is your total bandwidth, which remains constant and is utilized fu all times during the downloads. This means that when files finish downloading, the newly available bandwidth is distributed among the remaining files. The way it's distributed does not affect the final answer.

For example, consider the following scenario where you are downloading two files.

1) Speed = 3 KB/s Remaining Time 57 seconds

2) Speed = 2 KB/s Remaining Time 22 seconds

After 22 seconds, the second file will finish downloading. The first file still has 35 seconds remaining, but that time is based on the original speed. The bandwidth freed up by the second file is now allocated to the first file, and its new speed is 3+2=5 KB/s. The new remaining time is: Old_Remaining_Time * Old_Speed / New_Speed = 35*3/5 = 21 seconds.

So the actual remaining time for all the files is 21+22=43 seconds.

You will be given a String[] **tasks**, each element of which represents a single file being downloaded. Each file is represented as two positive integers with no leading zeroes, separated by a single space. The first integer is the current download speed in KB/s and the second integer is the remaining time in seconds based on the current speed. Return a double representing the remaining time in seconds for all the downloads to finish.

Definition

Class:	DownloadingFiles
Method:	actualTime
Parameters:	String[]
Returns:	double
Method signature:	double actualTime(String[] tasks)
(be sure your method is public)	

Constraints

- **tasks** will contain between 1 and 50 elements, inclusive.

- Each element of **tasks** will be formatted "<speed> <time>" (quotes for clarity), where <speed> is an integer between 1 and 100, inclusive, with no leading zeroes, and <time> is an integer between 1 and 10000, inclusive, with no leading zeroes.

Examples

0)

```
{"3 57","2 22"}
Returns: 43.0
```

The example from above.

1)

```
{"3 1057","2 1022"}
Returns: 1043.0
```

This is the same as the first example but all the files will take 1000 seconds more to completely download.

2)

```
{"25 1000","5 5000","10 5000"}
Returns: 2500.0
```

In this case, when the first file finishes downloading, we will have 25 KB/s of newly available bandwidth. We can share it between the remaining downloads however we want without affecting the final answer. Suppose that 15 KB/s goes to the second file and 10 KB/s goes to the third file. The new speeds and remaining times for those files will be:

20 KB/s and remaining time 2000 seconds

20 KB/s and remaining time 1000 seconds

Those would take 1500 seconds to complete. So the answer is 1000 + 1500 = 2500.

3)

```
{"1 10","1 20","2 40"}
Returns: 27.5
```

For this example, suppose that all newly available bandwidth goes to the slowest task every time. When the first download finishes (after 10 seconds), the second task doubles its speed to 2, and thus halves its remaining time from (20-10=10) to 5. When the second download finishes, the third one doubles its speed, so its remaining time goes from (40-15=25) to 12.5. The total time is 10+5+12.5=27.5.

4)

```
{"6 88","39 7057","63 2502","45 2285","28 8749","62 3636","1 5546","49 5741"}
Returns: 4414.542662116041
```

And here is a nice random example for you.