467-SRM 378 DIV2

Problem Statement

You have decided to create your own simple encryption method for strings containing only lowercase letters and spaces. You start by splitting the alphabet into two groups. The first group consists of the first firstSize letters of the alphabet, and the second consists of the remaining 26 - firstSize letters. To encrypt a character in your message, you do the following:

If it a space, it is kept as is.

If it is a letter in the first group, it is moved firstRotate letters forward in the group, wrapping back to the start if necessary. For example, if firstSize is 6 and firstRotate is 2, then 'A' would become

'C', and 'F' would become 'B'.

A b c d e f

If it is a letter in the second group, then it is moved secondRotate letters forward in the group, again wrapping back to the start of the group if necessary.

Given firstSize, firstRotate, secondRotate and a message, return the encrypted form of the message.

Definition

Class: TwoRotationCypher

Method: encrypt

Parameters: int, int, int, String

Returns: String

Method signature: String encrypt(int firstSize, int firstRotate, int secondRotate, String message)

(be sure your method is public)

Constraints-

firstSize will be between 1 and 25, inclusive.

-

firstRotate will be between 0 and firstSize - 1, inclusive.

-

secondRotate will be between 0 and 25 - firstSize, inclusive.

-

message will contain between 1 and 50 characters, inclusive.

-

message will contain only lowercase letters ('a' - 'z') and spaces.

Examples

0)

13

0

0

"this string will not change at all"

Returns: "this string will not change at all"

1)

13

7

0

"only the letters a to m in this string change"

Returns: "onfy tbl flttlrs h to g cn tbcs strcna jbhnal"

2)

9

0

16

"j to z will change here"

Returns: "z sn y vikk chamge heqe"

3)

17

9

5

"the quick brown fox jumped over the lazy dog"

Returns: "yqn izalc kwgsf ogt bzehnm grnw yqn djvu mgp"

4)

3

1

2

" watch out for strange spacing "

Returns: " ybvaj qwv hqt uvtbpig urbakpi "

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

469-SRM 379 DIV2

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 fully at 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.

This problem statement is the exclusive and proprietary property of TopCoder, Inc.

Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc.

is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

471-SRM 380 DIV2

Problem Statement

A lucky ticket is an integer with exactly 2\*n digits (written without leading zeroes),

where the sum of the leftmost n digits is equal to the sum of the rightmost n digits.

You are given a String s, which contains only non-zero digits.

Find the longest contiguous substring of s that is a lucky ticket and return its length.

If there is no such lucky ticket, return 0 instead.

Definition

Class:

LuckyTicketSubstring

Method:

maxLength

Parameters:

String

Returns:

int

Method signature:

int maxLength(String s)

(be sure your method is public)

Constraints

-

s will contain between 1 and 50 characters, inclusive.

-

s will contain non-zero digits ('1'-'9') only.

Examples

0)

"123231"

Returns: 6

The entire string, 123231, is a lucky ticket because the first 3 digits sum up to 1+2+3=6,

and the last 3 digits sum up to 2+3+1=6.

1)

"74233285"

Returns: 4

4233 is the longest lucky ticket here.

2)

"986561517416921217551395112859219257312"

Returns: 36

3)

"1"

Returns: 0

4)

"112"

Returns: 2

A lucky ticket must contain an even number of digits.

This problem statement is the exclusive and proprietary property of TopCoder, Inc.

Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc.

is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

473-SRM 381 DIV2

Problem Statement

As some of you may know, there is no name better than JOHN.

Let's define the rules for comparing names. Each letter has a weight ('A' - 1, 'B' - 2, ..., 'Z' - 26).

The weight of a name is the sum of the weights of all its letters.

For example, the name MARK has weight 13 + 1 + 18 + 11 = 43.

When comparing two names, the one with the larger weight is considered better.

In case of a tie, the one that comes earlier lexicographically is better.

But there is one exception - the name JOHN is the best name of all.

You are given a String[] names, each element of which contains a single name.

Sort the names from best to worst and return the sorted String[].

Definition

Class:

TheBestName

Method:

sort

Parameters:

String[]

Returns:

String[]

Method signature:

String[] sort(String[] names)

(be sure your method is public)

Constraints

-

names will contain between 1 and 50 elements, inclusive.

-

Each element of names will contain between 1 and 50 characters, inclusive.

-

Each element of names will contain only uppercase letters ('A'-'Z').

Examples

0)

{"JOHN", "PETR", "ACRUSH"}

Returns: {"JOHN", "ACRUSH", "PETR" }

PETR has weight 59, ACRUSH has weight 70 and JOHN has a weight of only 47.

But nevertheless JOHN is the best name, ACRUSH takes second place and PETR is the last.

1)

{"GLUK", "MARGARITKA"}

Returns: {"MARGARITKA", "GLUK" }

MARGARITKA is definitely better than GLUK.

2)

{"JOHN", "A", "AA", "AAA", "JOHN", "B", "BB", "BBB", "JOHN", "C", "CC", "CCC", "JOHN"}

Returns:

{"JOHN",

"JOHN",

"JOHN",

"JOHN",

"CCC",

"BBB",

"CC",

"BB",

"AAA",

"C",

"AA",

"B",

"A" }

AA and B both have the same weight, but AA is better as it comes earlier lexicographically.

For the same reason, AAA is better than C and BBB is better than CC.

3)

{"BATMAN", "SUPERMAN", "SPIDERMAN", "TERMINATOR"}

Returns: {"TERMINATOR", "SUPERMAN", "SPIDERMAN", "BATMAN" }

Args:

{{"LILOKJMQML", "LKIIQIKIIIOMPOIOQOQOORNKIK", "QNQIJMIONIKIMI", "PRRRNKQKIMIIIQMNIO", "IOKKMPMIRMMPIKM", "NIMMKOQNRJKIMIQKL", "MOKIQIOQIQKNQMQKRKQII", "PQRNIIQRMORMONOKIMPM", "LLMNKIOIMOQNNOIQQKMKRI", "QOQKINQQMNQIQO", "OQIROIMIIJQNKOJO", "NNMKIIKINIKIII", "NMOOMMKQIIIIKMOKNQNKIMQPNI", "KIJPIIMIQIMNIKIQOJII", "KIMIINOKKPRK", "MKNPNQIOOQLOJNOIOMNNIMNIRKQQ", "IOILNMPKOQIIIMIRII"}}

Expected:

{"MKNPNQIOOQLOJNOIOMNNIMNIRKQQ", "NMOOMMKQIIIIKMOKNQNKIMQPNI", "LKIIQIKIIIOMPOIOQOQOORNKIK", "LLMNKIOIMOQNNOIQQKMKRI", "PQRNIIQRMORMONOKIMPM", "MOKIQIOQIQKNQMQKRKQII", "PRRRNKQKIMIIIQMNIO", "KIJPIIMIQIMNIKIQOJII", "IOILNMPKOQIIIMIRII", "NIMMKOQNRJKIMIQKL", "OQIROIMIIJQNKOJO", "QOQKINQQMNQIQO", "IOKKMPMIRMMPIKM", "QNQIJMIONIKIMI", "NNMKIIKINIKIII", "KIMIINOKKPRK", "LILOKJMQML"}

Received:

{"MKNPNQIOOQLOJNOIOMNNIMNIRKQQ", "NMOOMMKQIIIIKMOKNQNKIMQPNI", "LKIIQIKIIIOMPOIOQOQOORNKIK", "LLMNKIOIMOQNNOIQQKMKRI", "PQRNIIQRMORMONOKIMPM", "MOKIQIOQIQKNQMQKRKQII", "PRRRNKQKIMIIIQMNIO", "KIJPIIMIQIMNIKIQOJII", "NIMMKOQNRJKIMIQKL", "IOILNMPKOQIIIMIRII", "OQIROIMIIJQNKOJO", "QOQKINQQMNQIQO", "IOKKMPMIRMMPIKM", "QNQIJMIONIKIMI", "NNMKIIKINIKIII", "KIMIINOKKPRK", "LILOKJMQML"}

Args:

{{"EAACEAD", "DGDEGADABDA", "DFACEFAFDGCDDADGDFCAAAFEACCBAEGAFAFADD", "DACGDABDABFADCBFDDABGGDAEACGABD", "CDABBEEEAD", "CBCBFDBFDBFBAGDBAAAABFGBGGDFFDE", "AABCBECFFAGACGABBGB", "AFFBDBABGGCEABBDCEBDBGABAACGEAG", "GAGAAAAAGFDBAGGEFDCCEFAAAEGGAACAAADBEGEEFBGADFE", "FCEEGAADDFCGGAABAACECFGBEAFEGAACDF", "DAGAGGCACBACGCBGFBBEFDECACCEBAGBBEGEBDE", "CDAADAGGGFCAFGEDFFAADAEAAAEBGAAAACBDCC", "DFBEAAGDGDDAEDACBDABDFFAAAE", "BCAEAGADBCGCGABECBEDDACCE", "BBGEFBGCGAAACEGDEAB", "BBGCADEAFC", "ABCAACGACAA", "EDGEEDCAC", "EEEDABFFEFAFBECDGFCCBFCCAEDCBEAAGABF", "EBFGEEFFACGEEGEEA", "GECADABBGAD", "CCDEACDAAAAEAGAADDG", "AFDAAFFABEEBDBECEAFDAAEFEBFBCDAFEAEFAACGBC", "AEFBGAABGAAFADGDCF", "ACBDGEAADFABGFFBCCDGEAECGFCBGCAAFCBAGCAACAGDCAGC", "ADGDDAAEAAGECFDBDFACGDGEAGDAGECFCAACBBEFBFFEAEDCAD", "DAEFCCADFAAEFAEBAFCCFBBFCEEFBCEACFBFGAGBBABADA", "BABACBAFEDCCCAGFDFFGFGCDFFDAECAAGEADAED", "BGBAGCCDAEGFEGADFABBGBF", "FABFGGACCAAAAGECDFBABADFAFCFDADBFADCEACBE", "EAAAABGGCECDCAB", "BFEBDEGFFDFGFCABAAABB", "BGDABEDA", "CBAFAAAEBGCABDAGFCAGAEGBA", "GFCDACDDEAFFBBFADFBACCAABCADBAFFDCCGBABE", "BAFDEDCAGCFABDBAAADECFEEFDEFGAAGF", "DAFDCGCFADEBEBFGAACAFGEFCDDAGCE", "GECGDGGEAEA", "FGFCDFCAGACFFFFFCAAAAFGEF", "BAFGAAECCDBACAAADBADEGAAGAABCFGAFAAAA", "FABCFGDGDCCADAABDFAEEEAAAFCGDAAABDGDAC", "BGAEDACAAFFCBFGAF", "DGCEAGCAABGECDFAECCEAFGAGAFDGFEDDGF", "CDBDAAFAAEGAGEFAGAAEGACDGCGBGEEAAEBFDFBE"}}

Expected:

{"ADGDDAAEAAGECFDBDFACGDGEAGDAGECFCAACBBEFBFFEAEDCAD", "GAGAAAAAGFDBAGGEFDCCEFAAAEGGAACAAADBEGEEFBGADFE", "ACBDGEAADFABGFFBCCDGEAECGFCBGCAAFCBAGCAACAGDCAGC", "DAEFCCADFAAEFAEBAFCCFBBFCEEFBCEACFBFGAGBBABADA", "CDBDAAFAAEGAGEFAGAAEGACDGCGBGEEAAEBFDFBE", "DGCEAGCAABGECDFAECCEAFGAGAFDGFEDDGF", "BABACBAFEDCCCAGFDFFGFGCDFFDAECAAGEADAED", "AFDAAFFABEEBDBECEAFDAAEFEBFBCDAFEAEFAACGBC", "DAGAGGCACBACGCBGFBBEFDECACCEBAGBBEGEBDE", "DFACEFAFDGCDDADGDFCAAAFEACCBAEGAFAFADD", "FABFGGACCAAAAGECDFBABADFAFCFDADBFADCEACBE", "EEEDABFFEFAFBECDGFCCBFCCAEDCBEAAGABF", "GFCDACDDEAFFBBFADFBACCAABCADBAFFDCCGBABE", "FCEEGAADDFCGGAABAACECFGBEAFEGAACDF", "FABCFGDGDCCADAABDFAEEEAAAFCGDAAABDGDAC", "CDAADAGGGFCAFGEDFFAADAEAAAEBGAAAACBDCC", "BAFDEDCAGCFABDBAAADECFEEFDEFGAAGF", "DAFDCGCFADEBEBFGAACAFGEFCDDAGCE", "CBCBFDBFDBFBAGDBAAAABFGBGGDFFDE", "FGFCDFCAGACFFFFFCAAAAFGEF", "AFFBDBABGGCEABBDCEBDBGABAACGEAG", "BAFGAAECCDBACAAADBADEGAAGAABCFGAFAAAA", "DACGDABDABFADCBFDDABGGDAEACGABD", "DFBEAAGDGDDAEDACBDABDFFAAAE", "BGBAGCCDAEGFEGADFABBGBF", "BCAEAGADBCGCGABECBEDDACCE", "EBFGEEFFACGEEGEEA", "CBAFAAAEBGCABDAGFCAGAEGBA", "BFEBDEGFFDFGFCABAAABB", "BBGEFBGCGAAACEGDEAB", "AEFBGAABGAAFADGDCF", "AABCBECFFAGACGABBGB", "BGAEDACAAFFCBFGAF", "CCDEACDAAAAEAGAADDG", "GECGDGGEAEA", "EAAAABGGCECDCAB", "DGDEGADABDA", "EDGEEDCAC", "GECADABBGAD", "BBGCADEAFC", "CDABBEEEAD", "BGDABEDA", "ABCAACGACAA", "EAACEAD"}

Received:

{"ADGDDAAEAAGECFDBDFACGDGEAGDAGECFCAACBBEFBFFEAEDCAD", "GAGAAAAAGFDBAGGEFDCCEFAAAEGGAACAAADBEGEEFBGADFE", "ACBDGEAADFABGFFBCCDGEAECGFCBGCAAFCBAGCAACAGDCAGC", "DAEFCCADFAAEFAEBAFCCFBBFCEEFBCEACFBFGAGBBABADA", "CDBDAAFAAEGAGEFAGAAEGACDGCGBGEEAAEBFDFBE", "DGCEAGCAABGECDFAECCEAFGAGAFDGFEDDGF", "BABACBAFEDCCCAGFDFFGFGCDFFDAECAAGEADAED", "DAGAGGCACBACGCBGFBBEFDECACCEBAGBBEGEBDE", "AFDAAFFABEEBDBECEAFDAAEFEBFBCDAFEAEFAACGBC", "DFACEFAFDGCDDADGDFCAAAFEACCBAEGAFAFADD", "FABFGGACCAAAAGECDFBABADFAFCFDADBFADCEACBE", "EEEDABFFEFAFBECDGFCCBFCCAEDCBEAAGABF", "GFCDACDDEAFFBBFADFBACCAABCADBAFFDCCGBABE", "FCEEGAADDFCGGAABAACECFGBEAFEGAACDF", "FABCFGDGDCCADAABDFAEEEAAAFCGDAAABDGDAC", "CDAADAGGGFCAFGEDFFAADAEAAAEBGAAAACBDCC", "BAFDEDCAGCFABDBAAADECFEEFDEFGAAGF", "DAFDCGCFADEBEBFGAACAFGEFCDDAGCE", "CBCBFDBFDBFBAGDBAAAABFGBGGDFFDE", "FGFCDFCAGACFFFFFCAAAAFGEF", "AFFBDBABGGCEABBDCEBDBGABAACGEAG", "BAFGAAECCDBACAAADBADEGAAGAABCFGAFAAAA", "DACGDABDABFADCBFDDABGGDAEACGABD", "DFBEAAGDGDDAEDACBDABDFFAAAE", "BGBAGCCDAEGFEGADFABBGBF", "BCAEAGADBCGCGABECBEDDACCE", "EBFGEEFFACGEEGEEA", "CBAFAAAEBGCABDAGFCAGAEGBA", "BFEBDEGFFDFGFCABAAABB", "BBGEFBGCGAAACEGDEAB", "AEFBGAABGAAFADGDCF", "AABCBECFFAGACGABBGB", "BGAEDACAAFFCBFGAF", "CCDEACDAAAAEAGAADDG", "GECGDGGEAEA", "EAAAABGGCECDCAB", "DGDEGADABDA", "EDGEEDCAC", "GECADABBGAD", "BBGCADEAFC", "CDABBEEEAD", "BGDABEDA", "ABCAACGACAA", "EAACEAD"}

Here are some superheroes sorted by their names.

This problem statement is the exclusive and proprietary property of TopCoder, Inc.

Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc

. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

475-SRM 382 DIV2

Problem Statement

Given an integer sequence a[0], a[1], ..., a[N-1], find a contiguous subsequnce a[i], a[i+1], ..., a[j] such that:

The length of the subsequence (j-i+1) is at least K.

The average value in the subsequence (a[i] + a[i+1] + ... + a[j])/(j-i+1) is maximized.

You are given a int[] a containing the original sequence,

and an int K. Return a int[] containing exactly two elements.

The first element is the first index of the contiguous subsequence described above,

and the second element is the last index. Both indices are 0-based.

If there are multiple subsequences that satisfy the conditions above, return the longest one among them.

If there are still multiple subsequences, return the one among them with the smallest first index.

Definition

Class:

ContiguousSubsequences

Method:

findMaxAverage

Parameters:

int[], int

Returns:

int[]

Method signature:

int[] findMaxAverage(int[] a, int K)

(be sure your method is public)

Constraints

-

a will contain between 1 and 50 elements, inclusive.

-

Each element of a will be between 0 and 1000000, inclusive.

-

K will be between 1 and number of elements in a, inclusive.

Examples

0)

{1,3,7}

2

Returns: {1, 2 }

There are 3 possible contiguous subsequences:

{1,3}, average = 2.

{1,3,7}, average = 11/3.

{3,7}, average = 5.

So {3,7} is the best case.

1)

{5,1,3,4}

2

Returns: {2, 3 }

2)

{10}

1

Returns: {0, 0 }

There is only one possible subsequence - the whole sequence.

3)

{381,921,513,492,135,802,91,519}

1

Returns: {1, 1 }

When K = 1, we can select the subsequence containing only the maximal elements.

4)

{381,921,513,492,135,802,91,519}

4

Returns: {0, 3 }

5)

{3,5,7,7,2,5,4,7,7,2,4}

1

Returns: {2, 3 }

6)

{3, 1, 3, 3, 3, 1, 3, 3, 3}

1

Returns: {2, 4 }

7)

{1, 3, 2, 1, 1, 2, 2, 2, 2}

3

Returns: {5, 8 }

This problem statement is the exclusive and proprietary property of TopCoder, Inc.

Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc

. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

477-SRM 383 DIV2

Problem Statement

You are building a house and are about to lay the floorboards in a particular room. Your architect has designed the layout of the floorboards and you now want to know how many boards you need to buy. Each board is 1 unit wide and can be of any positive integer length. The room is rectangular and the boards are to be laid out on a unit-square grid inside the room, parallel to the walls of the room.

You are given a String[] layout describing the layout of the floorboards. Character j of element i of layout describes the grid-square at position (i, j) and will either be a '-' or a '|', depending on which direction the floorboard covering that square is oriented. If two '-' characters are adjacent at the same value of i, then they form part of the same east/west-oriented floorboard. Similarly, if two '|' characters are adjacent at the same value of j, they are part of the same north/south-oriented floorboard. Return an int containing the number of distinct floorboards in the layout.

Definition

Class:

FloorLayout

Method:

countBoards

Parameters:

String[]

Returns:

int

Method signature:

int countBoards(String[] layout)

(be sure your method is public)

Constraints

-

layout will contain between 1 and 50 elements, inclusive.

-

Each element of layout will contain between 1 and 50 characters, inclusive.

-

Each element of layout will contain the same number of characters.

-

Each character in layout will be a '-' or a '|'.

Examples

0)

{"----"

,"----"

,"----"

,"----"}

Returns: 4

This layout contains 4 boards laid east/west.

1)

{"-||--||--"

,"--||--||-"

,"|--||--||"

,"||--||--|"

,"-||--||--"

,"--||--||-"}

Returns: 31

This is an aesthetic pattern made up of boards of lengths 1 and 2.

2)

{"--------"

,"|------|"

,"||----||"

,"|||--|||"

,"||----||"

,"|------|"

,"--------"}

Returns: 13

3)

{"||-||-|||-"

,"||--||||||"

,"-|-|||||||"

,"-|-||-||-|"

,"||--|-||||"

,"||||||-||-"

,"|-||||||||"

,"||||||||||"

,"||---|--||"

,"-||-||||||"}

Returns: 41

4)

{"-||--|"

,"||||||"

,"|||-|-"

,"-||||-"

,"||||-|"

,"||-||-"}

Returns: 19

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

# 479-SRM 384 DIV2

Problem Statement

Jane tries her best to stay fit. Knowing this, her sister decided to play a prank on Jane: after Jane's old scale broke down, her sister got her a new scale that shows the square of a person's weight, instead of the real weight.

Not noticing the change, Jane has been using the scale for a while. One morning she screamed "Oh no! I gained apparentGain pounds!". Given an int apparentGain (the difference between the square of Jane's current weight and the square of her previous weight), return a int[] containing Jane's possible real weights after the latest measurement, sorted in ascending order.

All weights (displayed and real, previous and current) are positive integers.

Definition

Class:

Prank

Method:

realWeight

Parameters:

int

Returns:

int[]

Method signature:

int[] realWeight(int apparentGain)

(be sure your method is public)

Constraints

-

apparentGain will be between 1 and 100000, inclusive.

Examples

0)

233

Returns: {117 }

If Jane had previously weighed 116 pounds, the scale would have shown 116^2 = 13456. If she gained 1 pound, the scale would show 117^2 = 13689, for an apparent gain of 13689 - 13456 = 233.

1)

15

Returns: {4, 8 }

Assuming in this example Jane is actually a small animal, she could have gone from 1 to 4 or from 7 to 8.

2)

1440

Returns: {38, 39, 42, 46, 49, 53, 66, 77, 94, 123, 182, 361 }

3)

100000

Returns: {325, 350, 550, 665, 1025, 1270, 2510, 3133, 5005, 6254, 12502, 25001 }

This is the largest input.

4)

93301

Returns: {3595, 46651 }

Watch out for overflow!

5)

16

Returns: {5 }

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

481-SRM 385 DIV2

Problem Statement

Rumor has it that Russians don't obey speed limits. That may be because the speed limits are sometimes specified implicitly in Russia.

More specifically, every road in a city has a default speed limit of 60 kilometers per hour, and usually doesn't have any road signs to remind drivers of that. Analogously, every road outside the city has a default speed limit of 90 kilometers per hour.

The speed limit can still be specified with road signs, like '30' or '95'. There is also a special road sign, 'start of default speed limit zone', that tells you that the default speed limit is now in place. The signs are sometimes also used to remind drivers of the current speed limit, so you can meet several same signs in a row.

To summarize, one should pay attention to the following road signs to monitor the speed limit changes:

Speed limit X - marks the start of a zone with speed limit X kilometers per hour.

Start of default speed limit zone - marks the start of a zone with the default speed limit, either 60 if inside a city or 90 if outside.

City boundary - means the default speed limit changes from 60 to 90 or vice versa. If you are inside a special speed limit zone, this zone also ends, so the speed limit always becomes equal to the new default.

Given the list of road signs you met on your way as a String[] signs (in the order you met them), return the current speed limit. Each element of signs will be either a positive integer number X without leading zeros, denoting the sign 'Speed limit X', a string "default" denoting the sign 'start of default speed limit zone', or a string "city", denoting the sign 'city boundary' (quotes for clarity only). You start your journey inside a city, and outside any special speed limit zone.

Definition

Class:

RussianSpeedLimits

Method:

getCurrentLimit

Parameters:

String[]

Returns:

int

Method signature:

int getCurrentLimit(String[] signs)

(be sure your method is public)

Constraints

-

signs will contain between 1 and 50 elements, inclusive.

-

Each element of signs will be "default", "city" or a positive integer without leading zeros, between 1 and 100, inclusive (quotes for clarity only).

Examples

0)

{"80"}

Returns: 80

On highways, speed limits may be above the default value.

1)

{"40", "70", "default", "20", "50"}

Returns: 50

The limits are specified in the order you meet them, so you're interested in the last one.

2)

{"40", "70", "default"}

Returns: 60

The default limit is still 60.

3)

{"40", "80", "city"}

Returns: 90

The first "city" means we've left the city, thus the 90 limit.

4)

{"city", "60"}

Returns: 60

Speed limits can be overridden outside the city too.

5)

{"city", "50", "default"}

Returns: 90

The default value changes when outside the city.

6)

{"city", "city", "city", "city"}

Returns: 60

You've crossed four city boundaries. The first time, you left a city. Then, you entered a city. Then, you left that city. Finally, you entered another city.

7)

{"20", "city", "city", "50", "60"}

Returns: 60

The default speed limit may be specified with a usual sign.

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

483-SRM 386 DIV2

Problem Statement

You have several trophies sitting on a shelf in a straight line. Their heights are given in a int[] trophies, from left to right. The shelf is positioned so that whenever people enter your room, they see it directly from the left side. In other words, the leftmost trophy is completely visible to the viewer, the next trophy in line is directly behind it, and so on. Unfortunately, tall trophies near the left side of the shelf might block the view of other trophies. A trophy is visible only if every trophy in front of it (from the viewer's perspective) is strictly shorter than it is. You wonder if rotating the shelf 180 degrees would increase the number of visible trophies. Return a int[] containing exactly two elements. The first element should be the number of trophies visible when viewing the shelf directly from the left side, and the second element should be the number of trophies visible when viewing the shelf directly from the right side.

Definition

Class:

TrophyShelf

Method:

countVisible

Parameters:

int[]

Returns:

int[]

Method signature:

int[] countVisible(int[] trophies)

(be sure your method is public)

Constraints

-

trophies will contain between 1 and 50 elements, inclusive.

-

Each element of trophies will be between 1 and 100, inclusive.

Examples

0)

{1,2,3,4,5}

Returns: {5, 1 }

When viewed from the left, each trophy is taller than all the trophies in front of it. However, when viewed from the right, the first trophy blocks the view of all the other trophies.

1)

{5,5,5,5}

Returns: {1, 1 }

Since all trophies have the same height, only the first is visible when viewed from each direction.

2)

{1,2,5,2,1}

Returns: {3, 3 }

This trophy shelf is symmetric.

3)

{1,4,2,5,3,7,1}

Returns: {4, 2 }

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

485-SRM 387 DIV2

Problem Statement

An integer arithmetic progression is a sequence defined by two positive integers, p and q, where p is the first element in the sequence, and all other elements are obtained by adding q to the previous element. For example, if p = 1 and q = 2, the sequence would be: 1, 3, 5, 7, ...

An integer geometric progression is a sequence defined by two positive integers, p and q, where p is the first element in the sequence, and all other elements are obtained by multiplying the previous element by q. For example, if p = 3 and q = 2, the sequence would be: 3, 6, 12, ...

You are given a int[] A, which contains either an integer arithmetic or geometric progression. Determine which one it is and return the next element in the sequence. It is guaranteed that A will uniquely represent either an arithmetic or geometric progression and that result will fit in a 32-bit signed integer.

Definition

Class:

GuessingNextElement

Method:

guess

Parameters:

int[]

Returns:

int

Method signature:

int guess(int[] A)

(be sure your method is public)

Constraints

-

A will contain between 3 and 50 elements, inclusive.

-

Each element of A will be between 1 and 10^6, inclusive.

-

A will be sorted in ascending order.

-

A will uniquely represent either an arithmetic or geometric progression.

Examples

0)

{364,843,1322,1801}

Returns: 2280

This sequence represents an arithmetic progression where p = 364 and q = 479. The next element is 1801 + 479 = 2280.

1)

{394,1172,1950,2728,3506,4284,5062,5840}

Returns: 6618

2)

{13,117,1053,9477,85293}

Returns: 767637

This sequence represents a geometric progression where p = 13 and q = 9. The next element is 85293 \* 9 = 76737.

3)

{22,220,2200,22000}

Returns: 220000

4)

{250000, 500000, 1000000}

Returns: 2000000

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

487-SRM 388 DIV2

Problem Statement

A strictly increasing sequence is a sequence of numbers where each number is strictly greater than the previous one. A strictly decreasing sequence is a sequence where each number is strictly less than the previous one. A strictly monotone sequence is a sequence that is either strictly increasing or strictly decreasing. For example, 1, 5, 6, 10 and 9, 8, 7, 1, are strictly monotone sequences, while 1, 5, 2, 6 and 1, 2, 2, 3 are not.

Given a sequence seq, determine the length of the longest contiguous subsequence that is strictly monotone (see examples for clarifications).

Definition

Class:

MonotoneSequence

Method:

longestMonotoneSequence

Parameters:

int[]

Returns:

int

Method signature:

int longestMonotoneSequence(int[] seq)

(be sure your method is public)

Constraints

-

seq will contain between 1 and 50 elements, inclusive.

-

Each element of seq will be between 1 and 100, inclusive.

Examples

0)

{1, 7, 7, 8, 3, 6, 7, 2}

Returns: 3

The longest contiguous monotone subsequence is 3, 6, 7. The sequence 1, 3, 6, 7 is not valid because 1 and 3 are not adjacent, and 1, 7, 7, 8 is not valid because it is not strictly increasing.

1)

{1, 1, 1, 1, 1}

Returns: 1

A sequence of one element is valid.

2)

{10, 20, 30, 25, 20, 19, 20, 18, 23}

Returns: 4

3)

{3, 2, 1, 4}

Returns: 3

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

489-SRM 389 DIV2

Problem Statement

You are packing a stack of books into some boxes, packing as many books as you can into each box without exceeding a given weight limit. Once you have packed as many books into a box as you can, you close and seal that box, and then begin filling the next one. You take the books off the stack in order, packing each one before picking up the next.

The weights of the books will be given as a int[] weights, where the first element is the weight of the book on top of the stack and the last element is the weight of the book on the bottom of the stack. The maximum weight that can fit into each box will be given as an int maxWeight. Return the minimum number of boxes you will need.

Definition

Class:

BoxesOfBooks

Method:

boxes

Parameters:

int[], int

Returns:

int

Method signature:

int boxes(int[] weights, int maxWeight)

(be sure your method is public)

Constraints

-

weights will contain between 0 and 50 elements, inclusive.

-

maxWeight will be between 1 and 1000, inclusive.

-

Each element of weights will be between 1 and maxWeight, inclusive.

Examples

0)

{ 5, 5, 5, 5, 5, 5 }

10

Returns: 3

You have 6 books that weigh 5 kilograms each. Each box can hold 10 kilograms (2 books). Therefore, you need 3 boxes.

1)

{ 51, 51, 51, 51, 51 }

100

Returns: 5

Each box can hold 100 kg, but since the books weigh 51 kg each, you can only put one in each box.

2)

{ 1, 1, 1, 7, 7, 7 }

8

Returns: 4

You would like to put one 1 kg book and one 7 kg book in each of 3 boxes. But, since you must pack the books in order, you end up putting the three 1 kg books in one box and each of the three 7 kg books in its own box, for a total of 4 boxes.

3)

{ 12, 1, 11, 2, 10, 3, 4, 5, 6, 6, 1 }

12

Returns: 6

4)

{ }

7

Returns: 0

5)

{ 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,

20, 20, 20, 20, 20, 20, 20, 20, 20, 20,

20, 20, 20, 20, 20, 20, 20, 20, 20, 20,

20, 20, 20, 20, 20, 20, 20, 20, 20, 20,

20, 20, 20, 20, 20, 20, 20, 20, 20, 20 }

1000

Returns: 1

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

491-SRM 390 DIV2

Problem Statement

Your little son is counting numbers with his left hand. Starting with his thumb and going toward his pinky, he counts each finger in order. After counting his pinky, he reverses direction and goes back toward his thumb. He repeats this process until he reaches his target number. He never skips a finger. For example, to count to ten, he would count: thumb, index, middle, ring, pinky, ring, middle, index, thumb, index. Sadly, one of his fingers hurts and he can only count on it a limited number of times. His fingers are numbered 1 through 5 from thumb to pinky. You are given an int weakFinger, the finger that hurts, and an int maxCount, the maximum number of times he can use that finger. Return the largest number he can count to. If he cannot even begin counting, return 0.

Definition

Class:

FingerCounting

Method:

maxNumber

Parameters:

int, int

Returns:

int

Method signature:

int maxNumber(int weakFinger, int maxCount)

(be sure your method is public)

Constraints

-

weakFinger will be between 1 and 5, inclusive.

-

maxCount will be between 0 and 100000, inclusive.

Examples

0)

2

3

Returns: 15

The first 15 numbers are counted with fingers 1,2,3,4,5,4,3,2,1,2,3,4,5,4,3. He would then have to use finger 2 for the next number, but since he has already used it 3 times, he has to stop.

1)

1

0

Returns: 0

He needs to use his thumb when counting the first number, 1, but it's too weak to be used even once.

2)

5

0

Returns: 4

Even though his pinky cannot be used at all, he can count 1,2,3,4 with the other fingers.

3)

2

48

Returns: 193

4)

5

973

Returns: 7788

5)

3

99999

Returns: 399998

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

493-SRM 391 DIV2

Problem Statement

Southern China is suffering from a heavily snowy winter. The heavy snow even causes the closure of an important highway connecting southern and northern China. You've got several reports containing the start and end points of highway segments covered by heavy snow. Given those reports as two int[]s startPoints and endPoints, you are to return the total length of highway segments covered by snow. Note that the reported segments may overlap.

Definition

Class:

SnowyWinter

Method:

snowyHighwayLength

Parameters:

int[], int[]

Returns:

int

Method signature:

int snowyHighwayLength(int[] startPoints, int[] endPoints)

(be sure your method is public)

Constraints

-

startPoints will contain between 1 and 50 elements, inclusive.

-

endPoints will contain the same number of elements as startPoints.

-

Each element of startPoints and endPoints will be between 0 and 10000, inclusive.

-

The i-th element of startPoints will be smaller than the corresponding element in endPoints.

Examples

0)

{17,85,57}

{33,86,84}

Returns: 44

These segments don't overlap.

1)

{45,100,125,10,15,35,30,9}

{46,200,175,20,25,45,40,10}

Returns: 132

There are 3 segments covered by snow: 9-25, 30-46 and 100-200.

2)

{4387,711,2510,1001,4687,3400,5254,584,284,1423,3755,929,2154,5719,1326,2368,554}

{7890,5075,2600,6867,7860,9789,6422,5002,4180,7086,8615,9832,4169,7188,9975,8690,1423}

Returns: 9691

The snow covered segment is 284-9975.

3)

{4906,5601,5087,1020,4362,2657,6257,5509,5107,5315,277,6801,2136,2921,5233,5082,497,8250,3956,5720}

{4930,9130,9366,2322,4687,4848,8856,6302,5496,5438,829,9053,4233,4119,9781,8034,3956,9939,4908,5928}

Returns: 9510

There are 2 segments covered by snow: 277-4930 and 5082-9939.

4)

{51,807,943,4313,8319,3644,481,220,2161,448,465,1657,6290,22,6152,647,3185,4474,2168}

{1182,912,1832,7754,9557,7980,4144,3194,7129,5535,1172,2043,6437,7252,9508,4745,8313,8020,4017}

Returns: 9535

5)

{8786,7391,201,4414,5822,5872,157,1832,7487,7518,2267,1763,3984,3102,7627,4099,524,1543,1022,3060}

{9905,7957,3625,6475,9314,9332,4370,8068,8295,8177,7772,2668,7191,8480,9211,4802,2625,1924,9970,4180}

Returns: 9813

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

495-SRM 392 DIV2

Problem Statement

On January 1, 2007, a confectioner made several candies.

On the last day of each month she allows her children to eat several of those candies.

The lifetime of a candy is the number of days between January 1 and the day the candy is eaten, inclusive. For example, the lifetime of a candy eaten on January 31 is 31, and the lifetime of a candy eaten on December 31 is 365 (note that 2007 wasn't a leap year).

You are given a int[] eatenCandies, the i-th element of which is the number of candies eaten on the last day of the i-th month of 2007 (January is month 0, February is month 1, etc.). Return the average lifetime of the candies.

Definition

Class:

AverageCandyLifetime

Method:

getAverage

Parameters:

int[]

Returns:

double

Method signature:

double getAverage(int[] eatenCandies)

(be sure your method is public)

Notes

-

The year 2007 was not a leap year.

-

The number of days in the months of 2007, in order, were 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30 and 31.

-

The returned value must be accurate to within a relative or absolute value of 1E-9.

Constraints

-

eatenCandies will contain exactly 12 elements.

-

Each element of eatenCandies will be between 0 and 1000, inclusive.

-

The sum of all the elements in eatenCandies will be greater than 0.

Examples

0)

{1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Returns: 60.5

One candy was eaten on January 31 and the other was eaten on March 31. The lifetimes of the candies are 31 and 31+28+31=90. The average lifetime is (31+90)/2=60.5.

1)

{0, 1000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Returns: 59.0

All candies were eaten on February 28. The lifetime of each candy is 31+28=59, so the average candy lifetime is 59.0.

2)

{0, 0, 0, 0, 0, 1, 0, 0, 0, 50, 0, 0}

Returns: 301.5882352941176

Most of the candies were eaten on October 31 (Halloween), and the lifetime of each of those candies is 304. The average lifetime is smaller than 304, because of a candy with lifetime 181, eaten on June 30.

3)

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Returns: 252.80769230769232

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

497-SRM 393 DIV2

Problem Statement

In order to make its roads safer, the government of a particular country has decided to introduce a speed limit that varies throughout the day. At busy times, the speed limit will decrease to reduce the risk of a dangerous accident. You need to drive a distance of journeyLength units and want to know how long it will take you.

You are given details of the speed limit in a int[] speedLimit. Element i (zero-based) gives the speed limit in DISTANCE UNITS/TIME UNITS that is in force between times T = i and T = i + 1. speedLimit describes the speed limit for a full day, so after this the pattern repeats (i.e., if N is the number of elements in speedLimit, the speed limit between times N and N+1 is given by speedLimit[0], etc.). You start your journey at time T = 0 and should assume that you travel exactly at the speed limit for your entire journey. Return a double containing the amount of time it takes to complete your journey.

Definition

Class:

VariableSpeedLimit

Method:

journeyTime

Parameters:

int, int[]

Returns:

double

Method signature:

double journeyTime(int journeyLength, int[] speedLimit)

(be sure your method is public)

Notes

-

Your return value must be accurate to an absolute or relative tolerance of 1E-9.

Constraints

-

journeyLength will be between 1 and 100000 (10^5), inclusive.

-

speedLimit will contain between 1 and 50 elements, inclusive.

-

Each element of speedLimit will be between 1 and 100, inclusive.

Examples

0)

100

{50}

Returns: 2.0

Here the speed limit doesn't change. The journey therefore takes a time of DISTANCE/SPEED = 100/50 = 2 time units.

1)

100

{50,25}

Returns: 2.5

Now the speed limit drops to 25 every other time unit. You drive 50 units in the first time unit and 25 in the second. You reach the end of your journey halfway through the third time unit.

2)

1000

{50,40,30,40,50}

Returns: 24.0

3)

2058

{80,43,57,23,28,45,60,75,73,80}

Returns: 37.4

4)

17216

{26,30,62,55,51,56,58,4,60,23,31}

Returns: 415.03333333333336

5)

9839

{45,90,13,4,81,50,81,10,64,86,69}

Returns: 182.8395061728395

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

499-SRM 394 DIV2

Problem Statement

You are in a mountainous area which is represented by a String[] areaMap. The j-th character of the i-th element of the areaMap is a digit '0'-'9' representing the height of cell (i, j). You perform a walk in the area according to the following rules:

You start from cell (0, 0).

If you are in cell (i, j), you examine cells (i+1, j), (i, j-1), (i-1, j), (i, j+1) in this order. You go to the first of these cells you can enter. You can enter a cell if it is still on the map, you haven't been to it before and the difference between the heights of your current cell and the cell you want to enter is no bigger (in absolute value) than heightDifference.

You end your walk if you can not make another move, i.e., if you can not enter any neighboring cell.

You must return the number of cells that you visit while performing your walk. You visit cell (i, j) if and only if you enter cell (i, j) at some point during your walk (the starting cell (0, 0) also counts as entered, i.e., you definitely visit (0, 0)). Note that you will visit each cell at most once since you never enter the same cell twice.

Definition

Class:

MountainWalk

Method:

cellsVisited

Parameters:

String[], int

Returns:

int

Method signature:

int cellsVisited(String[] areaMap, int heightDifference)

(be sure your method is public)

Constraints

-

areaMap will contain between 1 and 50 elements, inclusive.

-

All the elements of areaMap will contain the same number of characters.

-

Each element of areaMap will contain between 1 and 50 digits ('0' - '9'), inclusive.

-

heightDifference will be between 0 and 9, inclusive.

Examples

0)

{"056",

"135",

"234"}

1

Returns: 5

Your path goes (0, 0) --> (1, 0) --> (2, 0) --> (2, 1) --> (1, 1) and so you visit 5 cells.

1)

{"056",

"195",

"234"}

1

Returns: 8

Now you can not enter the cell (1, 1) because of the cell difference so your path goes (0, 0) --> (1, 0) --> (2, 0) --> (2, 1) --> (2, 2) --> (1, 2) --> (0, 2) --> (0, 1).

2)

{"865",

"123",

"111"}

3

Returns: 9

Your path is (0, 0) --> (0, 1) --> (0, 2) --> (1, 2) --> (2, 2) --> (2, 1) --> (2, 0) --> (1, 0) --> (1, 1).

3)

{"00009876543210",

"00009876543210",

"00009876543210",

"00009876543210"}

8

Returns: 16

4)

{"0000",

"0000",

"0000",

"0000",

"9999",

"8888",

"7777",

"6666",

"5555",

"4444",

"3333",

"2222",

"1111",

"0000"}

3

Returns: 16

5)

{"173642855131893831828253420",

"126290035950506994475683704",

"381277675415026563959463393",

"019782700912864681764582260",

"496448425114634806770407597",

"049628433145840178727435051",

"117194708226266248973780562",

"398138380998246682323622510",

"408178777661559971959512111"}

8

Returns: 135

6)

{"9"}

0

Returns: 1

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

501-SRM 395 DIV2

Problem Statement

You enjoy working with numbers that contain only square digits (namely, 0, 1, 4 and 9). The sequence containing only these digits is 0, 1, 4, 9, 10, 11, 14... Return the n-th term (indexed from 0) in this sequence.

Definition

Class:

SquareDigitNumbers

Method:

getNumber

Parameters:

int

Returns:

int

Method signature:

int getNumber(int n)

(be sure your method is public)

Constraints

-

n will be between 0 and 1000, inclusive.

Examples

0)

0

Returns: 0

The first square digit number is 0.

1)

5

Returns: 11

2)

16

Returns: 100

3)

121

Returns: 1941

4)

123

Returns: 1949

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

503-SRM 396 DIV2

Problem Statement

You are developing an application for online transactions and you want to accept several credit cards. Card numbers, however, are usually long, so it's easy to make mistakes when typing them in. You want to create a method that will verify the numbers entered by users. You know that the Luhn formula applies for all the acceptable card numbers. The Luhn formula works as follows. First, separate the individual digits of the credit card number. For example: 21378 becomes 2 1 3 7 8 If there is an even number of digits, multiply each digit in an odd position by 2. Otherwise, multiply each digit in an even position by 2. Positions are 1-indexed, so the first digit is at position 1. The example number above contains an odd number of digits, so we multiply each digit in an even position by 2: 2 1 3 7 8 becomes 2 2 3 14 8 Note that the even positions refer to the original number so they don't change even when a 2-digit number appears. Finally, take the sum of all the digits (for 2-digit numbers insert both the digits separately into the sum): 2+2+3+1+4+8 = 20 If the sum is a multiple of 10, the number is valid. Otherwise, it is invalid. Given a String cardNumber containing the credit card number, return "VALID" if the card number is valid, or "INVALID" if it is invalid (all quotes for clarity).

Definition

Class:

VerifyCreditCard

Method:

checkDigits

Parameters:

String

Returns:

String

Method signature:

String checkDigits(String cardNumber)

(be sure your method is public)

Constraints

-

cardNumber will contain between 1 and 50 characters, inclusive.

-

Each character in cardNumber will be a digit ('0'-'9').

Examples

0)

"21378"

Returns: "VALID"

This number has 5 digits, which is an odd number, so we multiply the digits in even positions by 2 to get: 2 2 3 14 8 The sum of the digits is 20, which is a multiple of 10, meaning it's a valid number.

1)

"31378"

Returns: "INVALID"

When we apply the Luhn formula here, the sum of the digits is 21, so the number is invalid.

2)

"11111101"

Returns: "VALID"

We multiply the digits in odd positions by 2 to get: 2 1 2 1 2 1 0 1 The sum of the digits is 10, so it's a valid card.

3)

"50005"

Returns: "VALID"

All the digits in even positions are 0 so multiplying by 2 doesn't change the number. The sum of the digits is 10, so it's a valid card.

4)

"542987223412"

Returns: "INVALID"

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.

505-SRM 397 DIV2

Problem Statement

You have been given a secret mission where you must break the enemy's code. You have already figured out that they encode messages using the following method. Each letter between 'a' and 'z', inclusive, is assigned a distinct two-digit number between 01 and 26, inclusive. A message is encoded by simply replacing each letter with its assigned number. For example, if 't' is assigned 20, 'e' is assigned 05 and 's' is assigned 19, then the message "test" is encoded as "20051920". All original messages contain only lowercase letters.

You are given a String code containing the assignment of numbers to letters. The first letter of code is assigned 01, the second is assigned 02 and so on. You are also given a String message which is either an original unencoded message or an encoded message. If you are given an unencoded message, return the encoded version of that message, and if you are given an encoded message, return the original unencoded message.

Definition

Class:

BreakingTheCode

Method:

decodingEncoding

Parameters:

String, String

Returns:

String

Method signature:

String decodingEncoding(String code, String message)

(be sure your method is public)

Constraints

-

code will contain exactly 26 characters.

-

Each lowercase letter between 'a' and 'z', inclusive, will occur exactly once in code.

-

message will contain between 1 and 50 characters, inclusive.

-

message will either contain only lowercase letters ('a'-'z') or only digits ('0'-'9').

-

If message contains only digits, it will be a concatenation of two-digit numbers, each between 01 and 26, inclusive.

Examples

0)

"abcdefghijklmnopqrstuvwxyz"

"test"

Returns: "20051920"

Example from the problem statement. Here, the letters are coded in an alphabetical order.

1)

"abcdefghijklmnopqrstuvwxyz"

"20051920"

Returns: "test"

Now, we're decoding it.

2)

"qesdfvujrockgpthzymbnxawli"

"mwiizkelza"

Returns: "19242626171202251723"

3)

"faxmswrpnqdbygcthuvkojizle"

"02170308060416192402"

Returns: "ahxpwmtvza"

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2003, TopCoder, Inc. All rights reserved.