Stanford ProCo

May 15, 2010

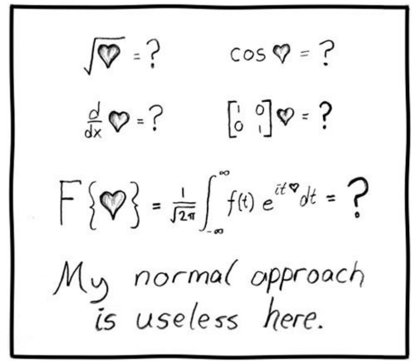
Problem Packet
Advanced Division

Sponsored by:

Advanced 2.1 **Abnormal Approach**

Overview: Add two base-10 numbers without carrying.

Description: Having fallen madly in love, you've lost all ability to do normal math. You find yourself unable to do even simple addition, as you always forget to carry. Fortunately, your programming skills are unimpaired, and you decide to make the most of this new style of false addition by writing a program to do it for you. Your normal approach is useless here.



http://xkcd.com/55/

Normal base-10 addition involves a carrying step whenever two digits sum to 10 or greater. For example, in 23 + 49 = 72, the 3 + 9 involves carrying a 1 to the tens unit. In false addition, any numbers that would be carried are simply dropped. So 23 + 49 = 62, since 3 + 9 = 12 (giving the 2 in the units place), and 2 + 4 = 6 (ignoring the carried 1).

Input:
Line 1: an integer *a*
Line 2: an integer *b*

Output:
Line 1: an integer *c*, the result of false addition performed on *a* and *b*.

Assumptions:
a and *b* will have the same number of digits in base 10.
$1 \le a < 1{,}000{,}000$
$1 \le b < 1{,}000{,}000$
$1 \le c < 2{,}000{,}000$
Leading zeros in *c* should not be printed. No leading zeros will appear in *a* and *b*.

Sample Input #1:
```
499
861
```

Sample Output #1:
```
250
```

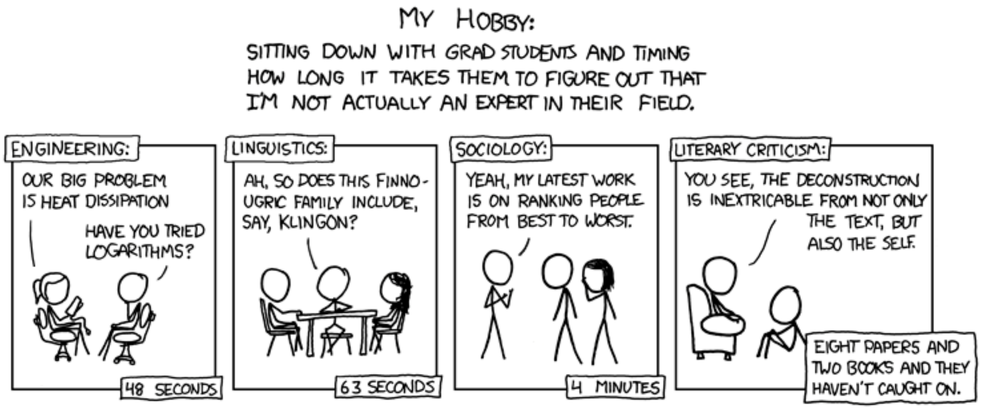
Sample Input #2:
```
19494
49494
```

Sample Output #2:
```
58888
```

Advanced 2.2 **My Hobby: Sociology**

Overview: Partition a list of numbers around a pivot

Description:



http://xkcd.com/451/

To further your research in Sociology, you decided to conduct a study ranking people from best to worst. Once you collected all your data, you decided that it would be simpler just to declare a threshold and rank people as either “good” or “bad.” The rigorous scientist that you are, you choose the “goodness” of the first person you interviewed to be the dividing line. Now all that remains is to rearrange the rest of your data.

Call the first element of a given list of n integers the pivot, with value p. Rearrange the list so that all numbers less than or equal to p are before it and all numbers greater than p are after it. The relative order of the numbers in each half of the new list must be retained; that is, for any b and c both before or both after the pivot in the result, if b was before c in the original list b should also be before c in the result.

Input:

Line 1: an integer n
Line 2: n space-separated integers, with the first integer as the pivot p

Output:

Line 1: n space-separated integers representing the rearranged list

Assumptions:

$1 \leq n < 1000$
All n integers, including the pivot p, will be ≥ 0 and $< 1{,}000{,}000$.
Trailing spaces after the n^{th} integer of output will be ignored.

Sample Input #1:

```
10
4 9 0 3 6 1 2 8 6 4
```

Sample Output #1:

```
0 3 1 2 4 4 9 6 8 6
```

Sample Input #2:

```
5
5 3 1 9 0
```

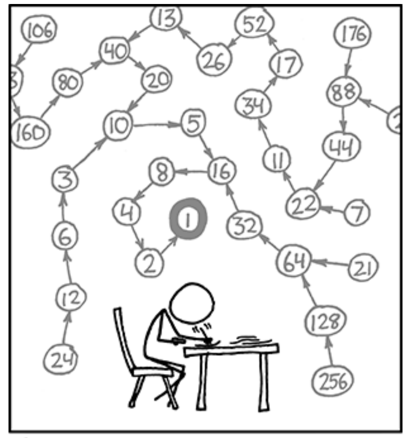
Sample Output #2:

```
3 1 0 5 9
```

Advanced 2.3 Happier than Hailstone

Overview: Check whether a given positive integer is a happy number.

Description: You've just spent the last three days in your room tracing the Hailstone Sequence. Sadly for you, you've made no progress in proving or disproving the Collatz Conjecture. To cheer yourself up, you decide to take on a different sequence of numbers, called happy/sad numbers. As in the Hailstone Sequence, you start at a certain number and start applying a given rule for generating the next number. If after some intense tracing, you end up at 1, then you're happy, and the number you started with is called a happy number. If you don't, you'll end up in a cycle, and having to do an infinite trace makes you sad.



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

http://xkcd.com/710/

Formally, a positive integer n is a happy number if and only if $f(\ldots f(f(n)))$ is eventually 1, where $f(n)$ is defined to be the sum of the squares of the digits of n. Positive integers that are not happy numbers are called sad numbers.

Input: Line 1: an integer n

Output: Line 1: a string, either `happy` or `sad`, denoting whether n makes you happy or sad

Assumptions: $1 \le n < 1{,}000{,}000{,}000$
All sequences either terminate at 1 or enter a cycle; no sequence grows without bound.
All intermediate numbers will be < 2,000,000,000.

Sample Input #1: `42`

Sample Output #1: `sad`

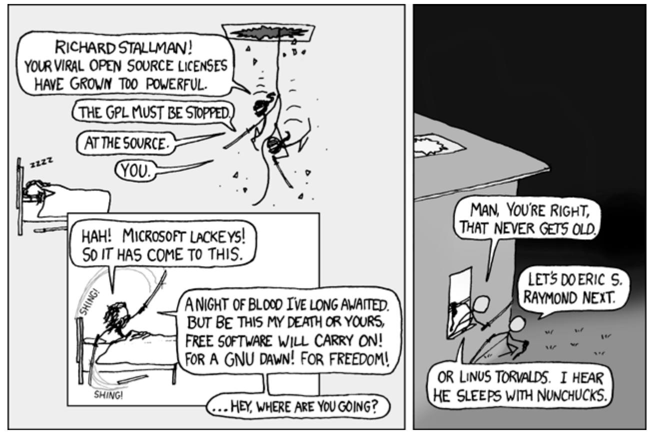
Sample Input #2: `1000`

Sample Output #2: `happy`

Advanced 2.4 **Magical**

Overview: Verify that a given grid is a normal magic square.

Description:



http://xkcd.com/225/

Richard Stallman did not appreciate your amateur ninja attack on his home! In revenge, he has locked you in an n x n cell. To alleviate your boredom you have begun hopping around the cell, but just as you enjoy walking on certain floor tile colors, you feel like you must hop around your cell such that the number of times you land on any tile in the cell leads to the formation of a magic square. A normal magic square is an n x n grid of unique integers 1 to n^2 such that every row, every column, and both principal diagonals have the same sum. For example,

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

is a 3 x 3 normal magic square because the rows, columns and principal diagonals (2-5-8 and 6-5-4) all sum to 15.

Input:

Line 1: an integer n, representing the width and height of the grid
Lines $2 \le i \le n + 1$: n space-separated integers that denote row i - 1

Output:

Line 1: a string, either `yes` or `no`, denoting whether the given grid is a magic square

Assumptions:

$1 \le n < 100$
All integers given will be > 0.

Sample Input #1:

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

Sample Output #1:

```
yes
```

Sample Input #2:

```
4
1 2 3 4
9 10 11 12
5 6 7 8
13 14 15 16
```

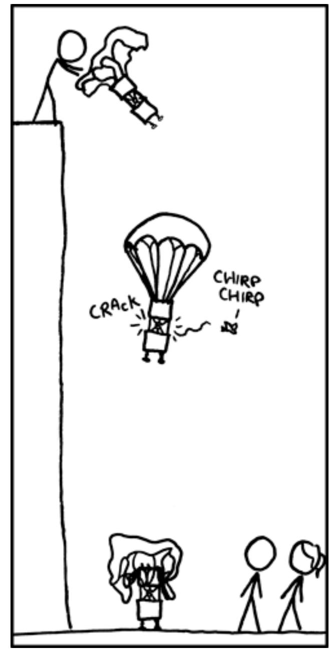
Sample Output #2:

```
no
```

Advanced 5.1 **Egg Drop Success**

Overview: Given two eggs, interactively find the height from which they will break

Description: In an alternate universe where all chickens are made of Osmium (the heaviest naturally occurring element, twice as dense as lead) and everyone is Randall Munroe's brother Randy, Mrs. Lenhart, the school science teacher, has just seen her first egg-dropping contest fail. Undeterred, she remodels the contest to consist of taking two same-mass eggs and dropping them from different stories of a 100-story building to see from which floor the egg will first break. Each team has two eggs.



http://xkcd.com/510/

Time to win this strange alternate universe contest! Both eggs have the same durability, and any floor in the building is equally likely to be the designated floor. Also, eggs dropped from below the magic floor are guaranteed not to break, no matter how many times they had been dropped previously. You are allowed a total of 20 drops, after which you must determine the magic floor. Note that an egg cannot be used again after it breaks, so after the second egg breaks you must submit what you think the floor number is.

Input/Output: This is an interactive problem. This means that your program will receive input from the grading environment based on the output your program produces. All input and output will be done through the console.

Rules of interaction:
1. Your program should output an integer *x*, which represents dropping an egg from floor *x*.
2. You MUST output a new line character and flush the output stream after each output! (See the sample contest problem)
3. Each query will result in an integer response *k*, which will be either `-1` or `1`, where `-1` indicates that the egg broke, and `1` indicates that the egg did not break.
4. Your program must submit a final guess immediately after one of the following occurs
   A) You submit 20 queries (20 drops)
   B) You break both eggs
You can make a final submission at any time by outputting a line of the form `G 51`
6. After you submit a floor, your program must immediately terminate.

Advanced 5.1 **Egg Drop Success**

Assumptions and Expectations:

1. If you drop an egg from any floor above and including the unknown floor, it will break.
2. If you drop an egg from a floor below the unknown floor, it will NOT break (no matter how many times you've already dropped the ball).
3. You are guaranteed that the unknown floor is between 1 and 100 inclusive.
4. All outputs from your program should be integers. If any output is invalid, your program is deemed incorrect.

Sample Run:
(Actual run consists only of second column; other words are shown for clarity)

Output:	50	Drop ball from 50th floor
Response:	-1	Ball #1 broke
Output:	9	Drop ball from 9th floor
Response:	1	Ball did not break
Output:	11	Drop ball from 11th floor
Response:	1	Ball did not break
Output:	12	Drop ball from 12th floor
Response:	-1	Ball #2 broke
Output:	G 12	Submit that the unknown floor is 12 (correct)

Advanced 5.2 The Difference

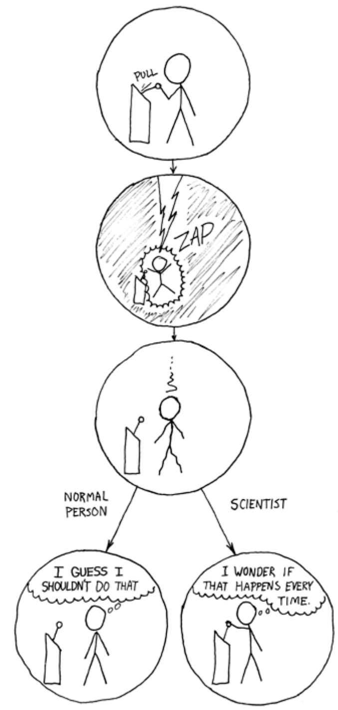
Overview: Find the elements that differ between two sets

Description:

Inspired by this comic, you decided to do an experiment to see how people will react to pulling a lever that zaps them. In the study, if the participant decides not to pull the lever again, you put them in a set, and if they do it again, they don't go in that set. But then your arch-nemesis comes and messes with your data by removing and adding people from your carefully determined set!

In order to save the study, we'll need your help: to purify the sets, write code to take in the mingled group of people and compute who will need to be added and removed to yield the original group.

Each person is represented as a single unique uppercase character `A-Z`, and a set of people is simply a string of uppercase characters in alphabetical order. The first set is the starting set, and the second set is the ending set.



http://xkcd.com/242/

Input:

Line 1: a string *s* representing the starting set
Line 2: a string *t* representing the ending set
Both strings will consist of unique uppercase characters in alphabetical order.

Output:

Line 1: a string *a* representing the elements to be added
Line 2: a string *r* representing the elements to be removed
Both strings should consist of unique uppercase characters in alphabetical order. Print `none` (in lowercase) if either set is empty.

Assumptions:

$1 \le |s| \le 26$
$1 \le |t| \le 26$

Sample Input #1:

```
AEIOU
BCEISTU
```

Sample Output #1:

```
BCST
AO
```

Sample Input #2:

```
WXYZ
Y
```

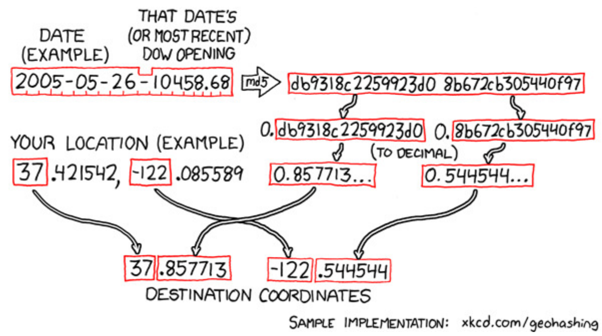
Sample Output #2:

```
none
WXZ
```

Advanced 5.3 **Palindromesemordnilap**

Overview: Output the length of the shortest palindrome containing the input as a contiguous substring.

Description: Despite your recent success with geohashing, you've decided that it's not quite enough. Rather than settle with taking the MD5 hashes straight from the Dow Jones opening, you want to find the shortest palindrome that contains that MD5 hash and perform some more complicated mangling. That's a lot of tedious calculations, so you decide to write a program instead.



http://xkcd.com/426/

A string *p* is a palindrome if and only if, when the letters in *p* are reversed to form the string *p'*, *p* = *p'*. You want to output the length of the shortest palindromic string that has the input *s* as a contiguous substring.

Input: Line 1: a string *s*, consisting of |*s*| characters

Output: Line 1: an integer |*p*|, indicating the number of characters in the shortest palindromic string *p* containing *s*

Assumptions: $1 \le |s| < 100$
s will only contain the uppercase characters `A-Z`.

Sample Input #1: `ILOVECS`

Sample Output #1: `13` (note: shortest strings are `ILOVECSCEVOLI` and `SCEVOLILOVECS`)

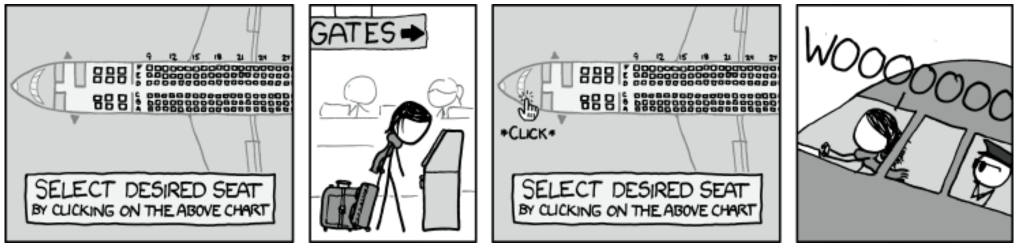
Sample Input #2: `SITONAPOTATO`

Sample Output #2: `19` (note: shortest string is `SITONAPOTATOPANOTIS`)

Advanced 5.4 **Seat Selection**

Overview: Given the number of people and a skip count, determine where you should start to sit where you want.

Description:



http://xkcd.com/726/

Sometimes seat selection is not as easy. One night, you are first to arrive at your friend's dinner party. Your friend has not finished setting up the *n* dinner table placeholders that he has made for you and your friends. He gives you the n placeholders to place around the round dinner table, and each seat at the table is already numbered *1* through *n*, increasing clockwise, in a circle. Because your friend likes to play games, he instructs you to put a placeholder at every m^{th} spot at the table moving clockwise, starting at whichever place you choose and ignoring any spot with a placeholder already. The last placeholder is yours, and you want to sit in the seat numbered *n*. At which spot should you start putting down placeholders?

For example, given 10 sets and instructions to put a placeholder at every 3^{rd} seat, you want to begin by putting a placeholder at seat 9. The placement order is as follows: 9, 2, 5, 8, 3, 7, 4, 1, 6, 10. (Note that you will place at 9 first, then skip 3.) The last seat, 10, is yours.

Input:

Line 1: an integer *n*
Line 2: an integer *m*

Output:

Line 1: an integer *f*, the prisoner to shoot first in order to spare the n^{th} prisoner

Assumptions:

$2 \le n < 1000$
$1 \le m \le n$

Sample Input #1:

```
10
3
```

Sample Output #1:

```
9
```

Sample Input #2:

```
16
8
```

Sample Output #2:

```
1
```

Advanced 9.1 **Autokey Dyvfitrhh**

Overview: Given the seed and ciphertext, decrypt an autokey cipher.

Description: After getting banned from the local cryptography contest once it became clear that your algorithms were all thinly disguised Lady Gaga songs, you decide that in order to regain your reputation as a cryptographer you must come up with a new code. You decide to call your new algorithm Can't Read My Message Text, which is simply an autokey cipher.

Encryption transforms an unencrypted string *p* (plaintext) into an encrypted string *c* (ciphertext) using a special string *k* (key). Given a key and plaintext of equal length, the ciphertext is derived one character at a time by doing a lookup in the table to the right. The autokey cipher is a special cipher that uses a seed *s* to generate the key *k* by appending the plaintext *p* to the seed and truncating to the length of *p*.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

For example, encryption using a seed of `PROCO` would give:

```
seed:        PROCO         (known)
plaintext:   THISISTOOEASY (known)
key:         PROCOTHISISTO (truncation of PROCOTHISISTOOEASY)
ciphertext:  IYWUWLAWGMSLM (output)
```

Your want to do the reverse: write a decryption algorithm that takes a seed *s* and the ciphertext *c* and outputs the plaintext *p*.

Input: Line 1: a string seed *s*, consisting of |*s*| uppercase characters
Line 2: a string ciphertext *c*, consisting of |*c*| uppercase characters

Output: Line 1: a string plaintext *p*, consisting of |*p*| uppercase characters

Assumptions: $1 \le |s| \le |c| = |p| \le 1000$

Sample Input #1:

```
PROCO
IYWUWLAWGMSLM
```

Sample Output #1:

```
THISISTOOEASY
```

Sample Input #2:

```
XKCD
PEFREUNSYEKWMRDOIPK
```

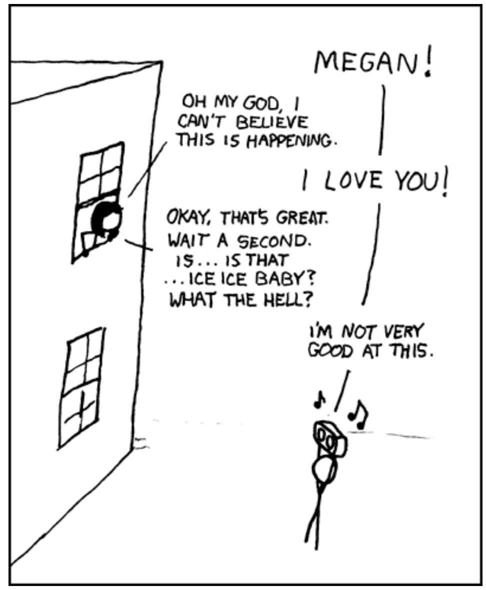
Sample Output #2:

```
SUDOMAKEMEASANDWICH
```

Advanced 9.2 **Some Digit Sum**

Overview: Determine how many numbers of a given length have digits summing to a given total.

Description: After your last failed attempt to impress Megan, you have decided to generate and memorize certain groups of numb ers. You are particularly fascinated by the *k*-digit numbers in base 10 (for example, the 4-digit numbers are 1000 through 9999). For each of these *k*-digit numbers, you can take the sum of all its digits (for example, 4503 has a digit sum of 12).



http://xkcd.com/159/

You have pondered the question: how many of these *k*-digit numbers have the same digit sum *s*? This will give you an idea of how many numbers there are per type of group to memorize. Get cracking!

Input: Line 1: an integer *k*, representing the length of the numbers
Line 2: an integer *s*, representing the sum of the digits

Output: Line 1: an integer *n*, equal to the number of *k*-digit base-10 numbers (the numbers from 10^{k-1} through $10^k - 1$ inclusive) whose digits sum to *s*

Assumptions: $1 \le k \le 9$
$1 \le s \le 9k$
$0 \le n < 10{,}000{,}000$

Sample Input #1:

```
2
4
```

Sample Output #1: `4` (these numbers are 13, 22, 31, 40)

Sample Input #2:

```
5
8
```

Sample Output #2:

```
330
```

Advanced 9.3 **A Matter of Life and Death**

Overview: Find the length of the shortest path through a maze, given a limited number of passes through walls.

Description: You were supposed to have a test today in your CS class, but your CS teacher was out sick and your substitute Mr. Monroe seems a little weird. As he hands out the tests, you read the first problem:

```
1. You are being chased through a maze by a group of hungry
   velociraptors! As you survey your apparently hopeless
   condition, you notice that the walls of the maze are low
   enough to climb over, but that climbing is substantially
   slower than fleeing on foot. If you spend too much time
   climbing, the velociraptors will surely catch you. Your only
   hope of survival is finding the shortest path through to the
   maze exit, where your velociraptor-proofed fortress awaits.
   Remember, raptors run at 10 m/s and they do not know fear.
```

http://xkcd.com/135/

The maze will consist of grid locations marked as either empty (`.`) or containing a wall (`#`). You may travel either up, down, left or right, but not diagonally. For each maze, you may travel through a specified maximum number of walls from the start square (`S`) to the finish square (`E`), though you need not travel through exactly that number. Your want to find the shortest path through the maze, where path length is calculated as follows: empty squares incur a distance penalty of 1 and walls incur a distance penalty of 3. The start square is not counted (consider it to be location 0), but the end square is. So a path that looks like `S#..E` will have a path length of 6.

Input: Line 1: three space-separated integers *r c w*, indicating the number of rows, the number of columns, and the maximum number of walls allowed, respectively
Lines $2 \le i \le r + 1$: a string of *c* characters that denotes row $i - 1$

Output: Line 1: an integer *p*, either the minimum path distance from start to finish, or `-1` if no valid path exists

Assumptions: $1 \le r \le 30$
$1 \le c \le 30$
$0 \le w \le 900$
All input characters will be either a `.` (empty square), a `#` (wall), an `S` (the start square), or an `E` (the end square).
There will be exactly one `S` and one `E` in the maze.

Advanced 9.3 **A Matter of Life and Death**

Sample Input #1:

```
5 5 2
S....
##.##
..#.E
.####
.....
```

Sample Output #1:

```
8
```

Sample Input #2:

```
3 4 1
S#.#
#.#E
..#.
```

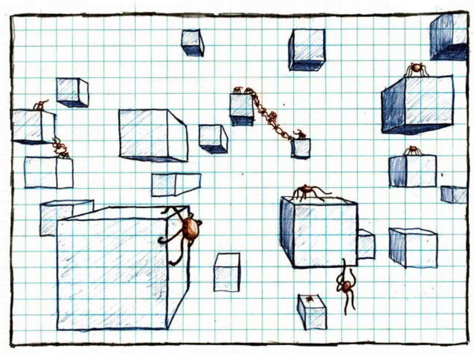
Sample Output #2:

```
-1
```

Advanced 9.4 **Red Superspiders**

Overview: Find the smallest number of red spiders whose climb shifts overlap with all others at some point.

Description: You and your fellow red spiders are cube climbing again. There are *n* spiders; each spider *i* will be cube climbing from time s_i inclusive to time e_i exclusive. The benevolent dictator of the Confederation of Red Spiders™ wants to make sure that no spiders die, and would like to form a spider committee of *m* members from the *n* spiders (leaving *n*-*m* spiders not on the committee) such that for each of the *n* spiders, at least one of the *m* spiders in the committee is present at some point when it is climbing cubes. What is the size of the smallest possible spider committee?



http://xkcd.com/8/

Input: Line 1: an integer *n*
Lines $2 \le i \le n + 1$: two space-separated integers s_{i-1} e_{i-1}

Output: Line 1: an integer *m*, the number of spiders in the smallest committee that can be formed

Assumptions: $1 \le n < 1000$
$0 \le s_i < e_i < 1{,}000{,}000$ for all *i*
Spider *i* stops climbing cubes instantly at time e_i. Thus a spider in the committee starting at time e_i does not count as being present while spider *i* is climbing.
Every climb shift overlaps with itself; that is, the set of all *n* spiders will satisfy the definition of a committee. This implies $m \le n$.

Sample Input #1:

```
4
1 2
3 4
2 3
1 3
```

Sample Output #1: `2` (spider #2 (overlaps #2) and #4 (overlaps #1, #3, and #4))

Sample Input #2:

```
3
0 2
5 6
1 7
```

Sample Output #2: `1` (spider #3 (overlaps #1, #2, and #3))