Daily Programming Challenges from Reddit

March 13, 2016

Contents

Chapter 1

Easy

Introduction

Easy challenges are meant to be fun and a great way to get into programming or a new language. For myself, I think of easy challenges as ones we can see the general idea of how to solve, but we may not know our way around a language yet to tackle it. The easy challenge gives us a concrete way to tackle it.

As you get more proficient, you'll find these can be tackled in just a few minutes in your language of choice. However they also make a great tool for learning a new language no matter how experienced you are. Many, but not all, can be solved in one or two functions.

Often in the /r/dailyprogrammer community we'll see more experienced programmers give advice to younger ones about doing things more clearly or in a more idiomatic style. This is great feedback.

1.1 Approximate PI

Description

The mathematical constant pi - the ratio of a circle's circumference to its radius - is an irrational number. Approximations have been made - first by hand and now by computers - for over 4000 years. The current record is to 12.1 trillion digits!

Approximations start to fail after various decimal places. For instance, the simple "25/8" approximation is good to only 2 decimal places, while "62832/20000" is correct to 4 decimal places. Various algorithms have been developed over the years that provide increasing accuracy.

For this challenge your task is to implement one or more of those algorithms and approximate pi correctly to a specific number of digits. You may *NOT* for this challenge use your programming language or system's built in definitions of pi (e.g. System.Math.PI from .Net, or M_PI from math.h), you should not solve it using

geometric functions like `sin()` or `tan()` or the like, or "find it" by downloading it from somewhere - that's straight up cheating.

Challenge Input

You'll be given *N*, a number of digits to which to correctly approximate pi.

```
4  (should be 3.1415 or 3.1416)
6  (should be 3.141592 or 3.141593)
10 (should be 3.1415926535)
```

Scala Solution

```
1  def factorial(n:Int):Int = if (n==0) 1 else n * factorial(n-1)
2
3  def Ramanujan_Series(k:Int, sofar:Double): Double = {
4      k match {
5          case -1 => 1/(12*sofar)
6          case _  => Ramanujan_Series(k - 1, sofar + ((math.pow(-1.0,
   ↪  k.toDouble) * factorial(6 * k) * (13591409 + 545140134 * k))/
7              (factorial(3 * k) * math.pow(factorial(k), 3.0) * math.pow(640320,
   ↪  (1.5 + 3 * k)))))
8          }
9  }
```

1.2 Atbash Cipher

Description

Atbash is a simple substitution cipher originally for the Hebrew alphabet, but possible with any known alphabet. It emerged around 500-600 BCE. It works by substituting the first letter of an alphabet for the last letter, the second letter for the second to last and so on, effectively reversing the alphabet. Here is the Atbash substitution table:

```
Plain: abcdefghijklmnopqrstuvwxyz
Cipher: ZYXWVUTSRQPONMLKJIHGFEDCBA
```

Amusingly, some English words Atbash into their own reverses, e.g., "wizard" = "draziw."

This is not considered a strong cipher but was at the time.

For more information on the cipher, please see the Wikipedia page on Atbash[1].

[1] `https://en.wikipedia.org/wiki/Atbash`

Input Description

For this challenge you'll be asked to implement the Atbash cipher and encode (or decode) some English language words. If the character is NOT part of the English alphabet (a-z), you can keep the symbol intact. Examples:

```
foobar
wizard
/r/dailyprogrammer
gsrh rh zm vcznkov lu gsv zgyzhs xrksvi
```

Output Description

Your program should emit the following strings as ciphertext or plaintext:

```
ullyzi
draziw
/i/wzrobkiltiznnvi
this is an example of the atbash cipher
```

Bonus

Preserve case.

Go Solution

```go
package main

import (
    "bytes"
    "fmt"
    "os"
    "strings"
)

func main() {
    input := "abcdefghijklmnopqrstuvwxyz"
    output := "zyxwvutsrqponmlkjihgfedcba"

    var ciphertext bytes.Buffer
    plaintext := strings.ToLower(os.Args[1])
    for i := 0; i < len(plaintext); i++ {
        j := strings.Index(input, string(plaintext[i]))
        if j >= 0 {
            ciphertext.WriteString(string(output[j]))
        } else {
            ciphertext.WriteString(string(plaintext[i]))
```

```
22          }
23      }
24
25      fmt.Println(ciphertext.String())
26  }
```

Fsharp Solution

```
1  let tbl = List.zip ['a'..'z'] ( [ 'a'..'z' ] |> List.rev) |> Map.ofList
2  let atbash (s:string) : string =
3      s.ToCharArray()
4      |> Array.map (fun x -> string(tbl.[x]) )
5      |> String.concat ""
```

1.3 Balancing Words

Description

Today we're going to balance words on one of the letters in them. We'll use the position and letter itself to calculate the weight around the balance point. A word can be balanced if the weight on either side of the balance point is equal. Not all words can be balanced, but those that can are interesting for this challenge.

The formula to calculate the weight of the word is to look at the letter position in the English alphabet (so A=1, B=2, C=3 ... Z=26) as the letter weight, then multiply that by the distance from the balance point, so the first letter away is multiplied by 1, the second away by 2, etc.

As an example:
STEAD balances at T: 1 * S(19) = 1 * E(5) + 2 * A(1) + 3 * D(4))
More info here[2] on the Questrel site.

Input Description

You'll be given a series of English words. Example:

```
STEAD
```

Output Description

Your program or function should emit the words split by their balance point and the weight on either side of the balance point. Example:

```
S T EAD - 19
```

This indicates that the T is the balance point and that the weight on either side is 19.

[2] http://www.questrel.com/records.html#spelling_alphabetical_order_entire_word_balance_points

Challenge Input

```
CONSUBSTANTIATION
WRONGHEADED
UNINTELLIGIBILITY
```

Challenge Output

```
CONSUBST A NTIATION - 456
WRO N GHEADED - 120
UNINTELL I GIBILITY - 521
```

Notes

This was found on a word games page suggested by /u/cDull, thanks! If you have your own idea for a challenge, submit it to /r/DailyProgrammer_Ideas, and there's a good chance we'll post it.

Scala Solution

```
1  def balance(word:String): (String, String, String, Int) = {
2    def loop(word:String, n:Int):(Int, Int) = {
3      n+word.length match {
4        case 1 =>       (0, -1)
5        case _ =>
6          val p = word.map(_.toInt-64).zip(n to
   ↪   (word.length+n-1)).map(x=>x._1*x._2).partition(_>0)
7          val lhs = p._1.sum
8          val rhs = p._2.sum
9          (lhs + rhs == 0) match {
10           case true  => (lhs, (n to (word.length+n-1)).indexOf(0))
11           case false => loop(word, n-1)
12         }
13     }
14   }
15   val b = loop(word.toUpperCase, 0)
16   b._1 match {
17     case 0 => ("", "", "", -1)
18     case _ => (word.substring(0, b._2), word(b._2).toString,
   ↪    word.substring(b._2+1, word.length), b._1)
19   }
20 }
21
22 // how many words can be balanced?
23 val bwords = scala.io.Source.
24               fromFile("/usr/share/dict/words").
25               getLines.
26               map(balance(_)).
```

```
27                filter(_._1 != "")
```

1.4 Broken Keyboard

Description

Help! My keyboard is broken, only a few keys work any more. If I tell you what keys work, can you tell me what words I can write?

(You should use the trusty enable1.txt file, or `/usr/share/dict/words` to chose your valid English words from.)

Input Description

You'll be given a line with a single integer on it, telling you how many lines to read. Then you'll be given that many lines, each line a list of letters representing the keys that work on my keyboard. Example:

```
3
abcd
qwer
hjklo
```

Output Description

Your program should emit the longest valid English language word you can make for each keyboard configuration.

```
abcd = bacaba
qwer = ewerer
hjklo = kolokolo
```

Challenge Input

```
4
edcf
bnik
poil
vybu
```

Challenge Output

```
edcf = deedeed
bnik = bikini
poil = pililloo
vybu = bubby
```

Scala Solution

Uses regexes

```
1  val words =
    ↪  io.Source.fromFile("/usr/share/dict/words").mkString.split("\n").toList
2  def typewriter(keys:String): String = words.filter(("[" + keys +
    ↪  "]+").r.replaceAllIn(_,"")=="").sortBy(x=>x.length).last
```

1.5 Cellular Automata: Rule 90

Description

The development of (cellular automata)[https://en.wikipedia.org/wiki/Cellular_automaton] (CA) systems is typically attributed to Stanisław Ulam and John von Neumann, who were both researchers at the Los Alamos National Laboratory in New Mexico in the 1940s. Ulam was studying the growth of crystals and von Neumann was imagining a world of self-replicating robots. That's right, robots that build copies of themselves. Once we see some examples of CA visualized, it'll be clear how one might imagine modeling crystal growth; the robots idea is perhaps less obvious. Consider the design of a robot as a pattern on a grid of cells (think of filling in some squares on a piece of graph paper). Now consider a set of simple rules that would allow that pattern to create copies of itself on that grid. This is essentially the process of a CA that exhibits behavior similar to biological reproduction and evolution. (Incidentally, von Neumann's cells had twenty-nine possible states.) Von Neumann's work in self-replication and CA is conceptually similar to what is probably the most famous cellular automaton: Conways "Game of Life," sometimes seen as a screen saver. CA has been pushed very hard by Stephen Wolfram (e.g. Mathematica, Worlram Alpha, and "A New Kind of Science").

CA has a number of simple "rules" that define system behavior, like "If my neighbors are both active, I am inactive" and the like. The rules are all given numbers, but they're not sequential for historical reasons.

The subject rule for this challenge, Rule 90, is one of the simplest, a simple neighbor XOR. That is, in a 1 dimensional CA system (e.g. a line), the next state for the cell in the middle of 3 is simply the result of the XOR of its left and right neighbors. E.g. "000" becomes "1" in the next state, "100" becomes "1" in the next state and so on. You traverse the given line in windows of 3 cells and calculate the rule for the next iteration of the following row's center cell based on the current one while the two outer cells are influenced by their respective neighbors. Here are the rules showing the conversion from one set of cells to another:

"111"	"101"	"010"	"000"	"110"	"100"	"011"	"001"
0	0	0	0	1	1	1	1

Input Description

You'll be given an input line as a series of 0s and 1s. Example:

```
1101010
```

Output Description

Your program should emit the states of the celular automata for 25 steps. Example from above, in this case I replaced 0 with a blank and a 1 with an X:

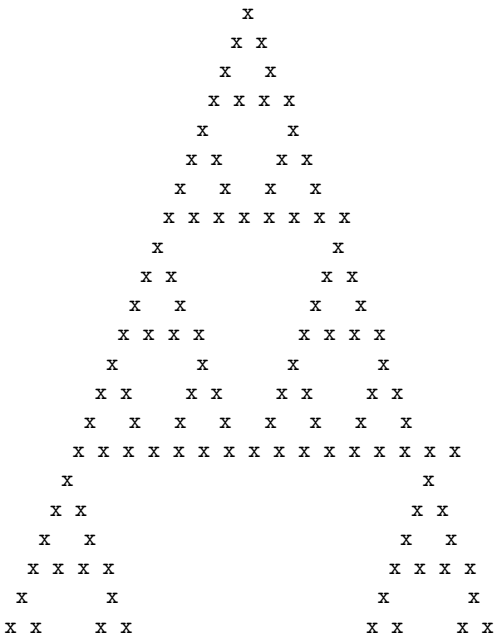
```
xx x x
xx   x
xxx  x
x xxx x
  x x
 x   x
```

Challenge Input

```
00000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000
```

Challenge Output

I chose this input because it's one of the most well known, it yields a Serpinski triangle, a well known fractcal.



Scala Solution

```
def rule90(row:String): String = {
    def loop(s:String): String = {
        s match {
            case "111" | "101" | "010" | "000" => "0"
            case "110" | "100" | "011" | "001" => "1"
        }
    }
    ("0" + row + "0").sliding(3).map(loop(_)).toList.mkString
}

def solution(s:String, n:Int) = {
    var row = s
    for (_ <- (0 to n)) {
        println(row.replace("0", " ").replace("1", "x"))
        row = rule90(row)
    }
}
```

1.6 Letters in Alphabetical Order

Description

A handful of words have their letters in alphabetical order, that is nowhere in the word do you change direction in the word if you were to scan along the English alphabet. An example is the word "almost", which has its letters in alphabetical order.

Your challenge today is to write a program that can determine if the letters in a word are in alphabetical order.

As a bonus, see if you can find words spelled in *reverse* alphebatical order.

Input Description

You'll be given one word per line, all in standard English. Examples:

```
almost
cereal
```

Output Description

Your program should emit the word and if it is in order or not. Examples:

```
almost IN ORDER
cereal NOT IN ORDER
```

Challenge Input

```
billowy
biopsy
chinos
defaced
chintz
sponged
bijoux
abhors
fiddle
begins
chimps
wronged
```

Challenge Output

```
billowy IN ORDER
biopsy IN ORDER
chinos IN ORDER
defaced NOT IN ORDER
chintz IN ORDER
sponged REVERSE ORDER
bijoux IN ORDER
abhors IN ORDER
fiddle NOT IN ORDER
begins IN ORDER
chimps IN ORDER
wronged REVERSE ORDER
```

Scala Solution

```
1  def alphabetical(word:String): Boolean = word.map(_.toInt).sorted ==
   ↪  word.map(_.toInt)
2  def rev_alphabetical(word:String): Boolean =
   ↪  word.map(_.toInt).sorted.reverse == word.map(_.toInt)
3
4  def main(word:String) = {
5      if (alphabetical(word) == true)
6          println(word + " IN ORDER")
7      else if (rev_alphabetical(word) == true)
8          println(word + " IN REVERSE ORDER")
```

```
9       else
10          println(word + " NOT IN ORDER")
11  }
```

1.7 In what year were most presidents alive?

Description

US presidents serve four year terms, with most presidents serving one or two terms. Unless a president dies in office, they then live after leaving office.

This challenge, then, given a list of presidents and their dates of birth and dates of death, is to figure out what *year* the most presidents - future, present, or previous - were alive.

Challenge Input

Below is a CSV input of presidential birthdates and death dates. Find what year in which the most number of people who would serve, are serving, or have served as presidents. The answer might be multiple years, or only a partial year.

```
PRESIDENT, BIRTH DATE, BIRTH PLACE, DEATH DATE, LOCATION OF DEATH
George Washington, Feb 22 1732, Westmoreland Co. Va., Dec 14 1799,
    Mount Vernon Va.
John Adams, Oct 30 1735, Quincy Mass., July 4 1826, Quincy Mass.
Thomas Jefferson, Apr 13 1743, Albemarle Co. Va., July 4 1826,
    Albemarle Co. Va.
James Madison, Mar 16 1751, Port Conway Va., June 28 1836, Orange Co.
    Va.
James Monroe, Apr 28 1758, Westmoreland Co. Va., July 4 1831, New York
     New York
John Quincy Adams, July 11 1767, Quincy Mass., Feb 23 1848, Washington
    D.C.
Andrew Jackson, Mar 15 1767, Lancaster Co. S.C., June 8 1845, Nashville
    Tennessee
Martin Van Buren, Dec 5 1782, Kinderhook New York, July 24 1862,
    Kinderhook New York
William Henry Harrison, Feb 9 1773, Charles City Co. Va., Apr 4 1841,
    Washington D.C.
John Tyler, Mar 29 1790, Charles City Co. Va., Jan 18 1862, Richmond Va
    .
James K. Polk, Nov 2 1795, Mecklenburg Co. N.C., June 15 1849, Nashville
    Tennessee
Zachary Taylor, Nov 24 1784, Orange County Va., July 9 1850, Washington
    D.C
Millard Fillmore, Jan 7 1800, Cayuga Co. New York, Mar 8 1874, Buffalo
    New York
Franklin Pierce, Nov 23 1804, Hillsborough N.H., Oct 8 1869, Concord
    New Hamp.
```

```
James Buchanan, Apr 23 1791, Cove Gap Pa., June 1 1868,  Lancaster Pa.
Abraham Lincoln,  Feb 12 1809,  LaRue Co. Kentucky, Apr 15 1865,
    Washington D.C.
Andrew Johnson, Dec 29 1808, Raleigh North Carolina, July 31 1875,
    Elizabethton Tenn.
Ulysses S. Grant, Apr 27 1822,  Point Pleasant Ohio,  July 23 1885,
    Wilton New York
Rutherford B. Hayes,  Oct 4 1822, Delaware Ohio, Jan 17 1893, Fremont
    Ohio
James A. Garfield, Nov 19 1831,  Cuyahoga Co. Ohio, Sep 19 1881,  Elberon
    New Jersey
Chester Arthur, Oct 5 1829, Fairfield Vermont, Nov 18 1886, New York New
    York
Grover Cleveland, Mar 18 1837,  Caldwell New Jersey,  June 24 1908,
    Princeton New Jersey
Benjamin Harrison, Aug 20 1833,  North Bend Ohio,  Mar 13 1901,
    Indianapolis Indiana
William McKinley, Jan 29 1843,  Niles Ohio, Sep 14 1901, Buffalo New
    York
Theodore Roosevelt, Oct 27 1858, New York New York, Jan 6 1919, Oyster
    Bay New York
William Howard Taft,  Sep 15 1857,  Cincinnati Ohio,  Mar 8 1930,
    Washington D.C.
Woodrow Wilson, Dec 28 1856, Staunton Virginia, Feb 3 1924, Washington D.
    C.
Warren G. Harding, Nov 2 1865, Morrow County Ohio, Aug 2 1923, San
    Francisco Cal.
Calvin Coolidge,  July 4 1872,  Plymouth Vermont, Jan 5 1933,
    Northampton Mass.
Herbert Hoover, Aug 10 1874, West Branch Iowa, Oct 20 1964,  New York
    New York
Franklin Roosevelt, Jan 30 1882, Hyde Park New York, Apr 12 1945, Warm
    Springs Georgia
Harry S. Truman,  May 8 1884, Lamar Missouri, Dec 26 1972, Kansas City
    Missouri
Dwight Eisenhower, Oct 14 1890,  Denison Texas, Mar 28 1969,  Washington
    D.C.
John F. Kennedy,  May 29 1917,  Brookline Mass.,  Nov 22 1963,  Dallas
    Texas
Lyndon B. Johnson, Aug 27 1908,  Gillespie Co. Texas,  Jan 22 1973,
    Gillespie Co. Texas
Richard Nixon, Jan 9 1913, Yorba Linda Cal., Apr 22 1994, New York New
    York
Gerald Ford,  July 14 1913,  Omaha Nebraska, Dec 26 2006, Rancho Mirage
    Cal.
Jimmy Carter,  Oct 1 1924, Plains Georgia, ,
Ronald Reagan, Feb 6 1911, Tampico Illinois, June 5 2004, Los Angeles Cal
    .
George Bush,  June 12 1924,  Milton Mass., ,
Bill Clinton,  Aug 19 1946,  Hope Arkansas, ,
```

```
George W. Bush, July 6 1946, New Haven Conn.,  ,
Barack Obama, Aug 4 1961, Honolulu Hawaii, ,
```

via U.S. Presidents Birth and Death Information[3].

Challenge Output

Any of the following years is valid: 1822, 1823, 1824, 1825, 1826, 1831, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1843, 1844, 1845 (each year had 18 presidents, current, former, or to be, alive).

Python Solution

```
import time
presidents = """PRESIDENT, BIRTH DATE, BIRTH PLACE, DEATH DATE, LOCATION
    ↪ OF DEATH
George Washington, Feb 22 1732, Westmoreland Co. Va., Dec 14 1799,
    ↪ Mount Vernon Va.
John Adams, Oct 30 1735, Quincy Mass., July 4 1826, Quincy Mass.
Thomas Jefferson, Apr 13 1743, Albemarle Co. Va., July 4 1826,
    ↪ Albemarle Co. Va.
James Madison, Mar 16 1751, Port Conway Va., June 28 1836, Orange
    ↪ Co. Va.
James Monroe, Apr 28 1758, Westmoreland Co. Va., July 4 1831, New
    ↪ York New York
John Quincy Adams, July 11 1767, Quincy Mass., Feb 23 1848,
    ↪ Washington D.C.
Andrew Jackson, Mar 15 1767, Lancaster Co. S.C., June 8 1845,
    ↪ Nashville Tennessee
Martin Van Buren, Dec 5 1782, Kinderhook New York, July 24 1862,
    ↪ Kinderhook New York
William Henry Harrison, Feb 9 1773, Charles City Co. Va., Apr 4 1841,
    ↪ Washington D.C.
John Tyler, Mar 29 1790, Charles City Co. Va., Jan 18 1862, Richmond
    ↪ Va.
James K. Polk, Nov 2 1795, Mecklenburg Co. N.C., June 15 1849,
    ↪ Nashville Tennessee
Zachary Taylor, Nov 24 1784, Orange County Va., July 9 1850,
    ↪ Washington D.C
Millard Fillmore, Jan 7 1800, Cayuga Co. New York, Mar 8 1874, Buffalo
    ↪ New York
Franklin Pierce, Nov 23 1804, Hillsborough N.H., Oct 8 1869, Concord
    ↪ New Hamp.
James Buchanan, Apr 23 1791, Cove Gap Pa., June 1 1868, Lancaster
    ↪ Pa.
```

[3] http://www.presidentsusa.net/birth.html

```
18 Abraham Lincoln,  Feb 12 1809,  LaRue Co. Kentucky, Apr 15 1865,
   ↪ Washington D.C.
19 Andrew Johnson, Dec 29 1808,  Raleigh North Carolina, July 31 1875,
   ↪ Elizabethton Tenn.
20 Ulysses S. Grant,  Apr 27 1822,  Point Pleasant Ohio,  July 23 1885,
   ↪ Wilton New York
21 Rutherford B. Hayes,  Oct 4 1822, Delaware Ohio,  Jan 17 1893,  Fremont
   ↪ Ohio
22 James A. Garfield,  Nov 19 1831,  Cuyahoga Co. Ohio,  Sep 19 1881,
   ↪ Elberon New Jersey
23 Chester Arthur, Oct 5 1829, Fairfield Vermont,  Nov 18 1886,  New York New
   ↪ York
24 Grover Cleveland,  Mar 18 1837,  Caldwell New Jersey,  June 24 1908,
   ↪ Princeton New Jersey
25 Benjamin Harrison,  Aug 20 1833,  North Bend Ohio,  Mar 13 1901,
   ↪ Indianapolis Indiana
26 William McKinley,  Jan 29 1843,  Niles Ohio, Sep 14 1901,  Buffalo New
   ↪ York
27 Theodore Roosevelt, Oct 27 1858,  New York New York,  Jan 6 1919, Oyster
   ↪ Bay New York
28 William Howard Taft,  Sep 15 1857,  Cincinnati Ohio,  Mar 8 1930,
   ↪ Washington D.C.
29 Woodrow Wilson, Dec 28 1856,  Staunton Virginia,  Feb 3 1924, Washington
   ↪ D.C.
30 Warren G. Harding,  Nov 2 1865, Morrow County Ohio, Aug 2 1923, San
   ↪ Francisco Cal.
31 Calvin Coolidge,  July 4 1872,  Plymouth Vermont,  Jan 5 1933,
   ↪ Northampton Mass.
32 Herbert Hoover, Aug 10 1874,  West Branch Iowa,  Oct 20 1964,  New York
   ↪ New York
33 Franklin Roosevelt, Jan 30 1882,  Hyde Park New York, Apr 12 1945,  Warm
   ↪ Springs Georgia
34 Harry S. Truman,  May 8 1884, Lamar Missouri, Dec 26 1972,  Kansas City
   ↪ Missouri
35 Dwight Eisenhower,  Oct 14 1890,  Denison Texas,  Mar 28 1969,
   ↪ Washington D.C.
36 John F. Kennedy,  May 29 1917,  Brookline Mass.,  Nov 22 1963,
   ↪ Dallas Texas
37 Lyndon B. Johnson,  Aug 27 1908,  Gillespie Co. Texas,  Jan 22 1973,
   ↪ Gillespie Co. Texas
38 Richard Nixon,  Jan 9 1913, Yorba Linda Cal.,  Apr 22 1994,  New York New
   ↪ York
39 Gerald Ford,  July 14 1913,  Omaha Nebraska, Dec 26 2006,  Rancho
   ↪ Mirage Cal.
40 Jimmy Carter,  Oct 1 1924, Plains Georgia, ,
41 Ronald Reagan,  Feb 6 1911, Tampico Illinois,  June 5 2004,  Los Angeles
   ↪ Cal.
```

```
42  George Bush,    June 12 1924,   Milton Mass.,   ,
43  Bill Clinton,   Aug 19 1946,    Hope Arkansas,  ,
44  George W. Bush, July 6 1946,    New Haven Conn.,    ,
45  Barack Obama,   Aug 4 1961, Honolulu Hawaii,    ,""".splitlines()
46
47  years = dict(map(lambda x: (x,0), xrange(1600, 2016)))
48  for line in presidents[1:]:
49      line = line.replace('July', 'Jul').replace('June', 'Jun')
50      gw = map(str.strip, line.split(','))
51      start = time.strptime(gw[1],  "%b %d %Y").tm_year
52      if len(gw[3]) > 1:
53          end = time.strptime(gw[3],  "%b %d %Y").tm_year
54      else:
55          end = time.gmtime().tm_year
56      for y in xrange(start, end+1):
57          years[y] = years.get(y, 0) + 1
58
59  years = [ (v,k) for k,v in years.iteritems() ]
60  n = max(years)[0]
61  print map(lambda x: x[1], filter(lambda x: x[0] == n, years))
```

1.8 Making numbers palindromic

Description

To covert nearly any number into a palindromic number you operate by reversing the digits and adding and then repeating the steps until you get a palindromic number. Some require many steps.

e.g. 24 gets palindromic after 1 steps: 66 -> 24 + 42 = 66

while 28 gets palindromic after 2 steps: 121 -> 28 + 82 = 110, so 110 + 11 (110 reversed) = 121.

Note that, as an example, 196 never gets palindromic (at least according to researchers, at least never in reasonable time). Several numbers never appear to approach being palindromic.

Input Description

You will be given a number, one per line. Example:

```
11
68
```

Output Description

You will describe how many steps it took to get it to be palindromic, and what the resulting palindrome is. Example:

```
11 gets palindromic after 0 steps: 11
68 gets palindromic after 3 steps: 1111
```

Challenge Input

```
123
286
196196871
```

Challenge Input Solution

```
123 gets palindromic after 1 steps: 444
286 gets palindromic after 23 steps: 8813200023188
196196871 gets palindromic after 45 steps: 44785554000069960000455587 44
```

Note

Bonus: see which input numbers, through 1000, yield identical palindromes.

Bonus 2: See which numbers don't get palindromic in under 10000 steps. Numbers that never converge are called Lychrel numbers.

Scala Solution

```
def reverse(n:Long): Long = n.toString.reverse.toLong

def palindrome(n:Long): Boolean =  n == reverse(n)

def loop(n:Long, steps:Int): (Long, Int) = {
    palindrome(n) match {
        case true  => (n, steps)
        case false => loop(reverse(n)+n, steps + 1)
    }
}
```

1.9 Pronouncing Hexidecimal

Description

The HBO network show "Silicon Valley" has introduced a way to pronounce hex.

```
Kid: Here it is: Bit... soup. It's like alphabet soup, BUT... it's ones
    and zeros instead of letters.
Bachman: {silence}
Kid: 'Cause it's binary? You know, binary's just ones and zeroes.
```

```
Bachman: Yeah, I know what binary is. Jesus Christ, I memorized the
    hexadecimal times tables when I was fourteen writing machine code.
    Okay? Ask me what nine times F is. It's fleventy-five. I don't need
    you to tell me what binary is.
```

Not "eff five", fleventy. `0xF0` is now fleventy. Awesome. Above a full byte you add "bitey" to the name. The hexidecimal pronunciation rules:

HEX PLACE VALUE	WORD
0xA0	"Atta"
0xB0	"Bibbity"
0xC0	"City"
0xD0	"Dickety"
0xE0	"Ebbity"
0xF0	"Fleventy"
0xA000	"Atta-bitey"
0xB000	"Bibbity-bitey"
0xC000	"City-bitey"
0xD000	"Dickety-bitey"
0xE000	"Ebbity-bitey"
0xF000	"Fleventy-bitey"

Combinations like `0xABCD` are then spelled out "atta-bee bitey city-dee".

For this challenge you'll be given some hex strings and asked to pronounce them.

Input Description

You'll be given a list of hex values, one per line. Examples:

```
0xF5
0xB3
0xE4
0xBBBB
0xA0C9
```

Output Description

Your program should emit the pronounced hex. Examples from above:

```
0xF5 "fleventy-five"
0xB3 "bibbity-three"
0xE4 "ebbity-four"
0xBBBB "bibbity-bee bitey bibbity-bee"
0xA0C9 "atta-bitey city-nine"
```

1.10 Ruth-Aaron Pairs

Description

In mathematics, a Ruth–Aaron pair consists of two consecutive integers (e.g. 714 and 715) for which the sums of the *distinct* prime factors of each integer are equal. For example, we know that (714, 715) is a valid Ruth-Aaron pair because its distinct prime factors are:

```
714 = 2 * 3 * 7 * 17
715 = 5 * 11 * 13
```

and the sum of those is both 29:

```
2 + 3 + 7 + 17 = 5 + 11 + 13 = 29
```

The name was given by Carl Pomerance, a mathematician at the University of Georgia, for Babe Ruth and Hank Aaron, as Ruth's career regular-season home run total was 714, a record which Aaron eclipsed on April 8, 1974, when he hit his 715th career home run. A student of one of Pomerance's colleagues noticed that the sums of the prime factors of 714 and 715 were equal.

For a little more on it, see MathWorld[4].

Your task today is to determine if a pair of numbers is indeed a valid Ruth-Aaron pair.

Input Description

You'll be given a single integer *N* on one line to tell you how many pairs to read, and then that many pairs as two-tuples. For example:

```
3
(714,715)
(77,78)
(20,21)
```

Output Description

Your program should emit if the numbers are valid Ruth-Aaron pairs. Example:

```
(714,715) VALID
(77,78) VALID
(20,21) NOT VALID
```

Chalenge Input

[4] http://mathworld.wolfram.com/Ruth-AaronPair.html

```
4
(5,6)
(2107,2108)
(492,493)
(128,129)
```

Challenge Output

```
(5,6) VALID
(2107,2108) VALID
(492,493) VALID
(128,129) NOT VALID
```

Scala Solution

```scala
def factorize(x: Int): List[Int] = {
  @tailrec
  def foo(x: Int, a: Int = 2, list: List[Int] = Nil): List[Int] = a*a > x
  ↪  match {
    case false if x % a == 0 => foo(x / a, a    , a :: list)
    case false               => foo(x    , a + 1, list)
    case true                => x :: list
  }
  foo(x)
}

def RA(a:Int, b:Int): Boolean = def RA(a:Int, b:Int): Boolean =
  ↪  factorize(a).toSet.sum == factorize(b).toSet.sum
```

CSharp Solution

```csharp
using System;
using System.Collections.Generic;
using System.Linq;

class Solution {
    public static void Main(string[] args) {
        int ra;
        bool t = int.TryParse(args[0], out ra);
        List<int> aints = PrimeFactors(ra);
        List<int> bints = PrimeFactors(ra+1);
        Console.WriteLine("{0} and {1} => {2}", ra, ra+1, aints.Sum() ==
  ↪  bints.Sum());
    }

    private static List<int> PrimeFactors(int n) {
        List<int> ints = new List<int>();
```

```
16          for (int i = 2; i <= n; i++) {
17              if (IsPrime(i) && (n%i == 0)) {
18                  ints.Add(i);
19              }
20          }
21          return ints;
22      }
23
24      private static bool IsPrime(int n)
25      {
26          bool prime = true;
27          int i = 2;
28          do {
29              prime &= (i == n || n%i != 0);
30              i += (i == 2 ? 1 : 2);
31          } while (i <= n);
32
33          return prime;
34      }
35  }
```

1.11 Shuffling a List

Description

We've had our fair share of sorting algorithms, now let's do a *shuffling* challenge. In this challenge, your challenge is to take a list of inputs and change around the order in random ways. Think about shuffling cards - can your program shuffle cards?

Input Description

You'll be given a list of values - integers, letters, words - in one order. The input list will be space separated. Example:

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

Output Description

Your program should emit the values in any non-sorted order; sequential runs of the program or function should yield different outputs. You should maximize the disorder if you can. From our example:

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

Challenge Input

```
apple blackberry cherry dragonfruit grapefruit kumquat mango nectarine
    persimmon raspberry raspberry
a e i o u
```

Challenge Output

Examples only, this is all about shuffling

```
raspberry blackberry nectarine kumquat grapefruit cherry raspberry apple
    mango persimmon dragonfruit
e a i o u
```

Bonus

Check out the Faro shuffle and the Fisher Yates shuffles, which are algorithms for specific shuffles. Shuffling has some interesting mathematical properties.

Scala Solution

```
1  def fischer_yates_shuffle(l:List[Int]): List[Int] = {
2      def loop(l:List[Int], n:Int): List[Int] = {
3          (l.length == n) match {
4              case true   => l
5              case false  => val i = (scala.math.random*l.length).toInt
6                             l.slice(0, n) ++ List(l(i)) ++ l.slice(n+1,i) ++
    ↪   List(l(n)) ++ l.slice(i+1,l.length)
7          }
8      }
9      loop(l, 0)
10 }
11
12 def faro_shuffle(l:List[Int], steps:Int): List[Int] = {
13     def loop(l:List[Int], n:Int): List[Int] = {
14         (n == 0) match {
15             case true  =>   l
16             case false =>   val (a,b) = (l.slice(0, l.length/2),
    ↪   l.slice(l.length/2, l.length))
17                             if (a.length != b.length) {
18                                 loop(a.zip(b).flatMap(x => List(x._1, x._2))
    ↪   ++ List(b.last), n-1)
19                             } else {
20                                 loop(a.zip(b).flatMap(x => List(x._1,
    ↪   x._2)), n-1)
21                             }
22         }
23     }
```

```
24      loop(l, steps)
25  }
```

1.12 Playing the Stock Market

Description

Let's assume I'm playing the stock market - buy low, sell high. I'm a day trader, so I want to get in and out of a stock before the day is done, and I want to time my trades so that I make the biggest gain possible.

The market has a rule that won't let me buy and sell in a pair of ticks - I have to wait for at least one tick to go buy. And obviously I can't buy in the future and sell in the past.

So, given a list of stock price ticks for the day, can you tell me what trades I should make to maximize my gain within the constraints of the market? Remember - buy low, sell high, and you can't sell before you buy.

Input Description

You'll be given a list of stock prices as a 2 decimal float (dollars and cents), listed in chronological order. Example:

```
19.35 19.30 18.88 18.93 18.95 19.03 19.00 18.97 18.97 18.98
```

Output Description

Your program should emit the two trades in chronological order - what you think I should buy at and sell at. Example:

```
18.88 19.03
```

Challenge Input

```
9.20 8.03 10.02 8.08 8.14 8.10 8.31 8.28 8.35 8.34 8.39 8.45 8.38 8.38
    8.32 8.36 8.28 8.28 8.38 8.48 8.49 8.54 8.73 8.72 8.76 8.74 8.87
    8.82 8.81 8.82 8.85 8.85 8.86 8.63 8.70 8.68 8.72 8.77 8.69 8.65
    8.70 8.98 8.98 8.87 8.71 9.17 9.34 9.28 8.98 9.02 9.16 9.15 9.07
    9.14 9.13 9.10 9.16 9.06 9.10 9.15 9.11 8.72 8.86 8.83 8.70 8.69
    8.73 8.73 8.67 8.70 8.69 8.81 8.82 8.83 8.91 8.80 8.97 8.86 8.81
    8.87 8.82 8.78 8.82 8.77 8.54 8.32 8.33 8.32 8.51 8.53 8.52 8.41
    8.55 8.31 8.38 8.34 8.34 8.19 8.17 8.16
```

Challenge Output

```
8.03 9.34
```

Python Quote Generator

```
import random

def stockprices(init, sofar=[]):
    if len(sofar) == 100: return sofar
    else:
        sofar.append(init + (random.paretovariate(init)-random.
            paretovariate(init))*0.5)
        return stockprices(sofar[-1], sofar)

print ' '.join(map(lambda x: '%.2f' % x, stockprices(8, [])))
```

Scala Solution

```
1  def pick(quotes:String) = {
2      def loop(quotes:List[Double], best:(Double, Double)): (Double, Double) =
   ↪  {
3          quotes.length match {
4              case 2 => best
5              case _ => {
6                  val biggest = quotes.tail.tail.map(x => ((quotes.head, x),
   ↪  x-quotes.head)).maxBy(_._2)
7                  if (biggest._2 > (best._2-best._1)) {
8                      loop(quotes.tail, biggest._1)
9                  } else {
10                     loop(quotes.tail, best)
11                 }
12             }
13         }
14     }
15     loop(quotes.split(" ").map(_.toDouble).toList, (0.0, 0.0))
16 }
```

1.13 Thue-Morse Sequence Generator

sequence

Description

The Thue-Morse sequence is a binary sequence (of 0s and 1s) that never repeats. It is obtained by starting with 0 and successively calculating the Boolean complement of the sequence so far. It turns out that doing this yields an infinite, non-repeating sequence. This procedure yields 0 then 01, 0110, 01101001, 0110100110010110, and so on.

See the Thue-Morse Wikipedia Article[5] for more information.

[5] http://en.wikipedia.org/wiki/Thue%E2%80%93Morse_sequence

Input

Nothing.

Output

Output the 0 to 6th order Thue-Morse Sequences.

Example

```
nth        Sequence
================================================================================

0          0
1          01
2          0110
3          01101001
4          0110100110010110
5          01101001100101101001011001101001
6
    0110100110010110100101100110100110010110011010010110100110010110
```

Extra Challenge

Be able to output any nth order sequence. Display the Thue-Morse Sequences for 100.

Note: Due to the size of the sequence it seems people are crashing beyond 25th order or the time it takes is very long. So how long until you crash. Experiment with it.

Fsharp Solution

```
let rec A3061 (L) =
    match (List.head L, List.tail L) with
    | (1, []) -> [ 0 ]
    | (0, []) -> [ 1 ]
    | (1, _ ) -> [ 0 ] @ A3061 (List.tail L)
    | (0, _ ) -> [ 1 ] @ A3061 (List.tail L)

let thuemorse (n:int) =
    let mutable L = [0]
    for i in [1..n] do
        L <- L @ A3061 L
    L
```

1.14 Vampire Numbers

Description

A vampire number v is a number $v{=}xy$ with an even number n of digits formed by multiplying a pair of $n/2$-digit numbers (where the digits are taken from the original number in any order) x and y together. Pairs of trailing zeros are not allowed. If v is a vampire number, then x and y are called its "fangs."

Additional information can be found here[6].

Input Description

Two digits on one line indicating n, the number of digits in the number to factor and find if it is a vampire number, and m, the number of fangs. Example:

```
4 2
```

Output Description

A list of all vampire numbers of n digits, you should emit the number and its factors (or "fangs"). Example:

```
1260=21*60
1395=15*93
1435=41*35
1530=51*30
1827=87*21
2187=27*81
6880=86*80
```

Challenge Input

```
6 3
```

Challenge Input Solution

```
114390=41*31*90
121695=21*61*95
127428=21*74*82
127680=21*76*80
127980=20*79*81
137640=31*74*60
139500=31*90*50
163680=66*31*80
178920=71*90*28
197925=91*75*29
```

[6] http://www.primepuzzles.net/puzzles/puzz_199.htm

```
198450=81*49*50
247680=40*72*86
294768=46*72*89
376680=73*60*86
397575=93*75*57
457968=56*94*87
479964=74*94*69
498960=99*84*60
```

Scala Solution

```
object VampireNumbers {
  def product(list: List[Int]): Int = list.foldLeft(1)(_*_)

  def vampire(n:Int, fangs:Int):List[(Int, List[Int])] ={
    n.
     toString.
     map(_.toString.toInt).
     permutations.
     map(_.grouped(2).map(_.mkString.toInt).toList).
     map(x=>(product(x),x)).
     filter(_._1==n).
     toList
  }

  def main(argc:Int, argv:Array[String]) = {
    val start = scala.math.pow(10, argv(1).toInt-1).toInt
    val end = scala.math.pow(10, argv(1).toInt).toInt-1
    val fangs = argv(2).toInt
    (start to end).map(x => vampire(x, fangs)).filter(_.length >
 ↪  0).foreach(println)
  }
}
```

1.15 Abundant and Deficient Numbers

Description

In number theory, a deficient or **deficient number** is a number n for which the sum of divisors *sigma(n)<2n*, or, equivalently, the sum of proper divisors (or aliquot sum) *s(n)<n*. The value *2n - sigma(n)* (or *n - s(n)*) is called the number's deficiency. In contrast, an **abundant number** or excessive number is a number for which the sum of its proper divisors is greater than the number itself

As an example, consider the number 21. Its divisors are 1, 3, 7 and 21, and their sum is 32. Because 32 is less than 2 x 21, the number 21 is *deficient*. Its deficiency is 2 x 21 - 32 = 10.

The integer 12 is the first *abundant* number. Its proper divisors are 1, 2, 3, 4 and 6 for a total of 16. The amount by which the sum exceeds the number is the abundance. The number 12 has an abundance of 4, for example.

Input Description

You'll be given an integer, one per line. Example:

```
18
21
9
```

Output Description

Your program should emit if the number if deficient, abundant (and its abundance), or neither. Example:

```
18 abundant by 3
21 deficient
9 neither
```

Challenge Input

```
111
112
220
69
134
85
```

Challenge Output

```
111 neither
112 abundant by 24
220 abundant by 64
69 deficient
134 deficient
85 deficient
```

Scala Solution

```
1  def divisors(n: Int): List[Int] = for (i <- List.range(1, n) if n % i == 0)
   ↪  yield i
2  def abundance(n:Int): Int = divisors(n).sum-n
3  def abundant(n:Int): Boolean = divisors(n).sum > n
4  def deficient(n:Int): Boolean = 2*divisors(n).sum < n
5  def main(n:Int): String = {
6      if (abundant(n)) {
```

```
7           return "abundant by " + abundance(n)
8       }
9       if (deficient(n)) {
10          return "deficient"
11      }
12      return "neither"
13  }
```

1.16 Detecting Alliteration

Description

Alliteration is defined as "the occurrence of the same letter or sound at the beginning of adjacent or closely connected words." It's a stylistic literary device identified by the repeated sound of the first consonant in a series of multiple words, or the repetition of the same sounds or of the same kinds of sounds at the beginning of words or in stressed syllables of a phrase. The first known use of the word to refer to a literary device occurred around 1624. A simple example is "Peter Piper Picked a Peck of Pickled Peppers".

Note on Stop Words

The following are some of the simplest English "stop words", words too common and uninformative to be of much use. In the case of Alliteration, they can come in between the words of interest (as in the Peter Piper example):

```
I
a
about
an
are
as
at
be
by
com
for
from
how
in
is
it
of
on
or
that
the
this
```

```
to
was
what
when
where
who
will
with
the
```

Sample Input

You'll be given an integer on a line, telling you how many lines follow. Then on the subsequent ines, you'll be given a sentence, one per line. Example:

```
3
Peter Piper Picked a Peck of Pickled Peppers
Bugs Bunny likes to dance the slow and simple shuffle
You'll never put a better bit of butter on your knife
```

Sample Output

Your program should emit the words from each sentence that form the group of alliteration. Example:

```
Peter Piper Picked Peck Pickled Peppers
Bugs Bunny    slow simple shuffle
better bit butter
```

Challenge Input

```
8
The daily diary of the American dream
For the sky and the sea, and the sea and the sky
Three grey geese in a green field grazing, Grey were the geese and green
    was the grazing.
But a better butter makes a batter better.
"His soul swooned slowly as he heard the snow falling faintly through the
     universe and faintly falling, like the descent of their last end,
    upon all the living and the dead."
Whisper words of wisdom, let it be.
They paved paradise and put up a parking lot.
So what we gonna have, dessert or disaster?
```

Challenge Output

```
daily diary
sky sea
grey geese green grazing
better butter batter better
soul swooned slowly
whisper words wisdom
paved paradise
dessert disaster
```

1.17 Anagram Detector

Description

An anagram is a form of word play, where you take a word (or set of words) and form a different word (or different set of words) that use the same letters, just rearranged. All words must be valid spelling, and shuffling words around doesn't count.

Some serious word play aficionados find that some anagrams can contain meaning, like "Clint Eastwood" and "Old West Action", or "silent" and "listen".

Someone once said, "All the life's wisdom can be found in anagrams. Anagrams never lie." How they don't lie is beyond me, but there you go.

Punctuation and capitalization don't matter.

Input Description

You'll be given two words or sets of words separated by a question mark. Your task is to replace the question mark with information about the validity of the anagram. Example:

```
"Clint Eastwood" ? "Old West Action"
"parliament" ? "partial man"
```

Output Description

You should replace the question mark with some marker about the validity of the anagram proposed. Example:

```
"Clint Eastwood" is an anagram of "Old West Action"
"parliament" is NOT an anagram of "partial man"
```

Challenge Input

```
"wisdom" ? "mid sow"
"Seth Rogan" ? "Gathers No"
"Reddit" ? "Eat Dirt"
"Schoolmaster" ? "The classroom"
"Astronomers" ? "Moon starer"
```

```
"Vacation Times" ? "I'm Not as Active"
"Dormitory" ? "Dirty Rooms"
```

Challenge Output

```
"wisdom" is an anagram of "mid sow"
"Seth Rogan" is an anagram of "Gathers No"
"Reddit" is NOT an anagram of "Eat Dirt"
"Schoolmaster" is an anagram of "The classroom"
"Astronomers" is NOT an anagram of "Moon starer"
"Vacation Times" is an anagram of "I'm Not as Active"
"Dormitory" is NOT an anagram of "Dirty Rooms"
```

Scala Solution

```
1  def anagram(s1:String, s2:String): Boolean =
   ↪  s1.toLowerCase().filter(_.isLetter).sorted ==
   ↪  s2.toLowerCase().filter(_.isLetter).sorted
```

1.18 Making Imgur-style Links

Description

Short links have been all the rage for several years now, spurred in part by Twitter's character limits. Imgur - Reddit's go-to image hosting site - uses a similar style for their links. Monotonically increasing IDs represented in Base62.

Your task today is to convert a number to its Base62 representation.

Input Description

You'll be given one number per line. Assume this is your alphabet:

```
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Example input:

```
15674
7026425611433322325
```

Output Description

Your program should emit the number represented in Base62 notation. Examples:

```
044
bDcRfbr63n8
```

Challenge Input

```
187621
237860461
2187521
18752
```

Challenge Output

```
9OM
3n26g
B4b9
sS4
```

Python Solution

```
def toBase62(n):
    alphabet =
 ↪  "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    basis = len(alphabet)
    ret = ''
    while (n > 0):
        tmp = n % basis
        ret += alphabet[tmp]
        n = (n/basis)
    return ret
```

1.19 Baum-Sweet Sequence

Description

In mathematics, the Baum–Sweet sequence[7] is an infinite automatic sequence of 0s and 1s defined by the rule:

- b_n = 1 if the binary representation of n contains no block of consecutive 0s of odd length;
- b_n = 0 otherwise;

for n >= 0.

For example, b_4 = 1 because the binary representation of 4 is 100, which only contains one block of consecutive 0s of length 2; whereas b_5 = 0 because the binary representation of 5 is 101, which contains a block of consecutive 0s of length 1. When n is 19611206, b_n is 0 because:

```
19611206 = 1001010110011111001000110 base 2
           00 0 0 00     00 000 0 runs of 0s
              ^ ^           ^^^    odd length sequences
```

[7] https://en.wikipedia.org/wiki/Baum%E2%80%93Sweet_sequence

Because we find an odd length sequence of 0s, b_n is 0.

Challenge Description

Your challenge today is to write a program that generates the Baum-Sweet sequence from 0 to some number *n*. For example, given "20" your program would emit:

```
1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0
```

Scala Solution

```
def b(n:Int): Int = {
    if (0 == n) {return 1}
    if (n.toBinaryString.split("1").filter(_!="").map(_.length%2 !=
    ↪  0).contains(true)) {return 0}
    else {return 1}
}

def baum_sweet(n:Int): IndexedSeq[Int] = (0 to n).map(b)
```

Go Solution

```
package main

import (
    "fmt"
    "os"
    "strconv"
    "strings"
)

func baumSweet(s string) int {
    zeroes := strings.Split(s, "1")
    for _, zero := range zeroes {
        if (len(zero) > 0) && ((len(zero) % 2) == 1) {
            return 1
        }
    }
    return 0
}

func main() {
    num, _ := strconv.ParseInt(os.Args[1], 10, 32)

    for n := 0; n <= int(num); n++ {
        s := strconv.FormatInt(int64(n), 2)
```

```
25          fmt.Printf("%d ", baumSweet(s))
26      }
27      fmt.Printf("\n")
28  }
```

1.20 Gold and Treasure: The Beale Cipher

Description

In 1885, an author named James B. Ward published a pamphlet telling of a long-lost treasure available to anyone clever enough to solve the puzzle associated with it. Ward reported that around 1817, a man named Thomas Jefferson Beale stumbled upon gold and silver deposits in what is now Colorado. Agreeing to keep it all a secret, Beale's team had spent the better part of two years quietly mining, then had taken the metals to Virginia by wagon and buried them in a vault underground between 1819 and 1821. Beale had written three notes explaining where the treasure was and who had legal rights to shares in it, encrypting each of these using a different text.

Eventually, the second of the three texts was deciphered using a slightly altered version of the Declaration of Independence. Each number in the text corresponded to a word in the U.S. Declaration of Independence. The first letter of each of those words spelled the plaintext—with a few modifications for errors and spelling.

Your mission today is to go treasure hunting and to write a program to decipher Beale's message.

DECLARATION OF INDEPENDENCE

When in the course of human events it becomes necessary for one people to dissolve the political bands which have connected them with another and to assume among the powers of the earth the separate and equal station to which the laws of nature and of nature's god entitle them a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation we hold these truths to be self evident that all men are created equal that they are endowed by their creator with certain unalienable rights that among these are life liberty and the pursuit of happiness that to secure these rights governments are instituted among men deriving their just powers from the consent of the governed that whenever any form of government becomes destructive of these ends it is the right of the people to alter or to abolish it and to institute new government laying its foundation on such principles and organizing its powers in such form as to them shall seem most likely to effect their safety and happiness prudence indeed will dictate that governments long established should not be changed for light and transient causes and accordingly all experience hath shown that mankind are more disposed to suffer while evils are sufferable than to right themselves by abolishing the forms to which they are accustomed but when a long train of abuses and usurpations pursuing invariably the same object evinces a design to reduce them under absolute

despotism it is their right it is their duty to throw off such government and to provide new guards for their future security such has been the patient sufferance of these colonies and such is now the necessity which constrains them to alter their former systems of government the history of the present king of great Britain is a history of repeated injuries and usurpations all having in direct object the establishment of an absolute tyranny over these states to prove this let facts be submitted to a candid world he has refused his assent to laws the most wholesome and necessary for the public good he has forbidden his governors to pass laws of immediate and pressing importance unless suspended in their operation till his assent should be obtained and when so suspended he has utterly neglected to attend to them he has refused to pass other laws for the accommodation of large districts of people unless those people would relinquish the right of representation in the legislature a right inestimable to them and formidable to tyrants only he has called together legislative bodies at places unusual uncomfortable and distant from the depository of their public records for the sole purpose of fatiguing them into compliance with his measures he has dissolved representative houses repeatedly for opposing with manly firmness his invasions on the rights of the people he has refused for a long time after such dissolutions to cause others to be elected whereby the legislative powers incapable of annihilation have returned to the people at large for their exercise the state remaining in the meantime exposed to all the dangers of invasion from without and convulsions within he has endeavored to prevent the population of these states for that purpose obstructing the laws for naturalization of foreigners refusing to pass others to encourage their migration hither and raising the conditions of new appropriations of lands he has obstructed the administration of justice by refusing his assent to laws for establishing judiciary powers he has made judges dependent on his will alone for the tenure of their offices and the amount and payment of their salaries he has erected a multitude of new offices and sent hither swarms of officers to harass our people and eat out their substance he has kept among us in times of peace standing armies without the consent of our legislatures he has affected to render the military independent of and superior to the civil power he has combined with others to subject us to a jurisdiction foreign to our constitution and unacknowledged by our laws giving his assent to their acts of pretended legislation for quartering large bodies of armed troops among us for protecting them by a mock trial from punishment for any murders which they should commit on the inhabitants of these states for cutting off our trade with all parts of the world for imposing taxes on us without our consent for depriving us in many cases of the benefits of trial by jury for transporting us beyond seas to be tried for pretended offenses for abolishing the free system of English laws in a neighboring province establishing therein an arbitrary government and enlarging its boundaries so as to render it at once an example and fit instrument for introducing the same absolute rule into these colonies for taking away our charters abolishing our most valuable laws and altering fundamentally the forms of our governments for suspending our own legislature and declaring themselves invested with power to legislate for us in all cases whatsoever he has abdicated government here by declaring us out of his protection and waging war against us he has plundered our seas ravaged our coasts burnt our towns and destroyed the lives of our people he is at this time transporting

large armies of foreign mercenaries to complete the works of death desolation and tyranny already begun with circumstances of cruelty and perfidy scarcely paralleled in the most barbarous ages and totally unworthy the head of a civilized nation he has constrained our fellow citizens taken captive on the high seas to bear arms against their country to become the executioners of their friends and brethren or to fall themselves by their hands he has excited domestic insurrections amongst us and has endeavored to bring on the inhabitants of our frontiers the merciless Indian savages whose known rule of warfare is an undistinguished destruction of all ages sexes and conditions in every stage of these oppressions we have petitioned for redress in the most humble terms our repeated petitions have been answered only by repeated injury a prince whole character is thus marked by every act which may define a tyrant is unfit to be the ruler of a free people nor have we been wanting in attention to our British brethren we have warned them from time to time of attempts by their legislature to extend an unwarrantable jurisdiction over us we have reminded them of the circumstances of our emigration and settlement here we have appealed to their native justice and magnanimity and we have conjured them by the ties of our common kindred to disavow these usurpations which would inevitably interrupt our connections and correspondence they too have been deaf to the voice of justice and of consanguinity we must therefore acquiesce in the necessity which denounces our separation and hold them as we hold the rest of mankind enemies in war in peace friends we therefore the representatives of the united states of America in general congress assembled appealing to the supreme judge of the world for the rectitude of our intentions do in the name and by authority of the good people of these colonies solemnly publish and declare that these united colonies are and of right ought to be free and independent states that they are absolved from all allegiance to the British crown and that all political connection between them and the state of great Britain is and ought to be totally dissolved and that as free and independent states they have full power to levy war conclude peace contract alliances establish commerce and to do all other acts and things which independent states may of right do and for the support of this declaration with a firm reliance on the protection of divine providence we mutually pledge to each other our lives our fortunes and our sacred honor .

Input Description

You'll be given a list of numbers, comma separated, representing the ciphertext given by Beale. Example:

115, 73, 24, 807, 37, 52, 49, 17, 31, 62, 647, 22, 7, 15, 140, 47, 29, 107, 79, 84, 56, 239, 10, 26, 811, 5, 196, 308, 85, 52, 160, 136, 59, 211, 36, 9, 46, 316, 554, 122, 106, 95, 53, 58, 2, 42, 7, 35, 122, 53, 31, 82, 77, 250, 196, 56, 96, 118, 71, 140, 287, 28, 353, 37, 1005, 65, 147, 807, 24, 3, 8, 12, 47, 43, 59, 807, 45, 316, 101, 41, 78, 154, 1005, 122, 138, 191, 16, 77, 49, 102, 57, 72, 34, 73, 85, 35, 371, 59, 196, 81, 92, 191, 106, 273, 60, 394, 620, 270, 220, 106, 388, 287, 63, 3, 6, 191, 122, 43, 234, 400, 106, 290, 314, 47, 48, 81, 96, 26, 115, 92, 158, 191, 110, 77, 85, 197, 46, 10, 113, 140, 353, 48, 120, 106, 2, 607, 61, 420, 811, 29, 125, 14, 20, 37, 105, 28, 248, 16, 159, 7, 35, 19, 301, 125, 110, 486, 287, 98, 117, 511, 62,

51, 220, 37, 113, 140, 807, 138, 540, 8, 44, 287, 388, 117, 18, 79, 344, 34, 20, 59, 511, 548, 107, 603, 220, 7, 66, 154, 41, 20, 50, 6, 575, 122, 154, 248, 110, 61, 52, 33, 30, 5, 38, 8, 14, 84, 57, 540, 217, 115, 71, 29, 84, 63, 43, 131, 29, 138, 47, 73, 239, 540, 52, 53, 79, 118, 51, 44, 63, 196, 12, 239, 112, 3, 49, 79, 353, 105, 56, 371, 557, 211, 505, 125, 360, 133, 143, 101, 15, 284, 540, 252, 14, 205, 140, 344, 26, 811, 138, 115, 48, 73, 34, 205, 316, 607, 63, 220, 7, 52, 150, 44, 52, 16, 40, 37, 158, 807, 37, 121, 12, 95, 10, 15, 35, 12, 131, 62, 115, 102, 807, 49, 53, 135, 138, 30, 31, 62, 67, 41, 85, 63, 10, 106, 807, 138, 8, 113, 20, 32, 33, 37, 353, 287, 140, 47, 85, 50, 37, 49, 47, 64, 6, 7, 71, 33, 4, 43, 47, 63, 1, 27, 600, 208, 230, 15, 191, 246, 85, 94, 511, 2, 270, 20, 39, 7, 33, 44, 22, 40, 7, 10, 3, 811, 106, 44, 486, 230, 353, 211, 200, 31, 10, 38, 140, 297, 61, 603, 320, 302, 666, 287, 2, 44, 33, 32, 511, 548, 10, 6, 250, 557, 246, 53, 37, 52, 83, 47, 320, 38, 33, 807, 7, 44, 30, 31, 250, 10, 15, 35, 106, 160, 113, 31, 102, 406, 230, 540, 320, 29, 66, 33, 101, 807, 138, 301, 316, 353, 320, 220, 37, 52, 28, 540, 320, 33, 8, 48, 107, 50, 811, 7, 2, 113, 73, 16, 125, 11, 110, 67, 102, 807, 33, 59, 81, 158, 38, 43, 581, 138, 19, 85, 400, 38, 43, 77, 14, 27, 8, 47, 138, 63, 140, 44, 35, 22, 177, 106, 250, 314, 217, 2, 10, 7, 1005, 4, 20, 25, 44, 48, 7, 26, 46, 110, 230, 807, 191, 34, 112, 147, 44, 110, 121, 125, 96, 41, 51, 50, 140, 56, 47, 152, 540, 63, 807, 28, 42, 250, 138, 582, 98, 643, 32, 107, 140, 112, 26, 85, 138, 540, 53, 20, 125, 371, 38, 36, 10, 52, 118, 136, 102, 420, 150, 112, 71, 14, 20, 7, 24, 18, 12, 807, 37, 67, 110, 62, 33, 21, 95, 220, 511, 102, 811, 30, 83, 84, 305, 620, 15, 2, 10, 8, 220, 106, 353, 105, 106, 60, 275, 72, 8, 50, 205, 185, 112, 125, 540, 65, 106, 807, 138, 96, 110, 16, 73, 33, 807, 150, 409, 400, 50, 154, 285, 96, 106, 316, 270, 205, 101, 811, 400, 8, 44, 37, 52, 40, 241, 34, 205, 38, 16, 46, 47, 85, 24, 44, 15, 64, 73, 138, 807, 85, 78, 110, 33, 420, 505, 53, 37, 38, 22, 31, 10, 110, 106, 101, 140, 15, 38, 3, 5, 44, 7, 98, 287, 135, 150, 96, 33, 84, 125, 807, 191, 96, 511, 118, 40, 370, 643, 466, 106, 41, 107, 603, 220, 275, 30, 150, 105, 49, 53, 287, 250, 208, 134, 7, 53, 12, 47, 85, 63, 138, 110, 21, 112, 140, 485, 486, 505, 14, 73, 84, 575, 1005, 150, 200, 16, 42, 5, 4, 25, 42, 8, 16, 811, 125, 160, 32, 205, 603, 807, 81, 96, 405, 41, 600, 136, 14, 20, 28, 26, 353, 302, 246, 8, 131, 160, 140, 84, 440, 42, 16, 811, 40, 67, 101, 102, 194, 138, 205, 51, 63, 241, 540, 122, 8, 10, 63, 140, 47, 48, 140, 288

Output Description

Your program should consume the input and decrypt it. Remember - the first letter of that word number from the US Declaration of Independence. Spacing, punctuation, capitalization, and fixing spelling is left as an exercise to the treasure seeker (as Beale intended). The above letter was intended to decrypt to:

I have deposited in the county of Bedford, about four miles from Buford's, in an excavation or vault, six feet below the surface of the ground, the following articles, belonging jointly to the parties whose names are given in number "3," herewith:

The first deposit consisted of one thousand and fourteen pounds of gold, and three thousand eight hundred and twelve pounds of silver, deposited November, 1819. The second was made December, 1821, and consisted of nineteen hundred and seven pounds of gold, and twelve hundred and eighty-eight pounds of silver; also jewels, obtained in St. Louis in exchange for silver to save transportation, and

valued at $13,000.

The above is securely packed in iron pots, with iron covers. The vault is roughly lined with stone, and the vessels rest on solid stone, and are covered with others. Paper number "1" describes the exact locality of the vault so that no difficulty will be had in finding it.

Challenge Input

71, 194, 38, 1701, 89, 76, 11, 83, 1629, 48, 94, 63, 132, 16, 111, 95, 84, 341, 975, 14, 40, 64, 27, 81, 139, 213, 63, 90, 1120, 8, 15, 3, 126, 2018, 40, 74, 758, 485, 604, 230, 436, 664, 582, 150, 251, 284, 308, 231, 124, 211, 486, 225, 401, 370, 11, 101, 305, 139, 189, 17, 33, 88, 208, 193, 145, 1, 94, 73, 416, 918, 263, 28, 500, 538, 356, 117, 136, 219, 27, 176, 130, 10, 460, 25, 485, 18, 436, 65, 84, 200, 283, 118, 320, 138, 36, 416, 280, 15, 71, 224, 961, 44, 16, 401, 39, 88, 61, 304, 12, 21, 24, 283, 134, 92, 63, 246, 486, 682, 7, 219, 184, 360, 780, 18, 64, 463, 474, 131, 160, 79, 73, 440, 95, 18, 64, 581, 34, 69, 128, 367, 460, 17, 81, 12, 103, 820, 62, 116, 97, 103, 862, 70, 60, 1317, 471, 540, 208, 121, 890, 346, 36, 150, 59, 568, 614, 13, 120, 63, 219, 812, 2160, 1780, 99, 35, 18, 21, 136, 872, 15, 28, 170, 88, 4, 30, 44, 112, 18, 147, 436, 195, 320, 37, 122, 113, 6, 140, 8, 120, 305, 42, 58, 461, 44, 106, 301, 13, 408, 680, 93, 86, 116, 530, 82, 568, 9, 102, 38, 416, 89, 71, 216, 728, 965, 818, 2, 38, 121, 195, 14, 326, 148, 234, 18, 55, 131, 234, 361, 824, 5, 81, 623, 48, 961, 19, 26, 33, 10, 1101, 365, 92, 88, 181, 275, 346, 201, 206, 86, 36, 219, 324, 829, 840, 64, 326, 19, 48, 122, 85, 216, 284, 919, 861, 326, 985, 233, 64, 68, 232, 431, 960, 50, 29, 81, 216, 321, 603, 14, 612, 81, 360, 36, 51, 62, 194, 78, 60, 200, 314, 676, 112, 4, 28, 18, 61, 136, 247, 819, 921, 1060, 464, 895, 10, 6, 66, 119, 38, 41, 49, 602, 423, 962, 302, 294, 875, 78, 14, 23, 111, 109, 62, 31, 501, 823, 216, 280, 34, 24, 150, 1000, 162, 286, 19, 21, 17, 340, 19, 242, 31, 86, 234, 140, 607, 115, 33, 191, 67, 104, 86, 52, 88, 16, 80, 121, 67, 95, 122, 216, 548, 96, 11, 201, 77, 364, 218, 65, 667, 890, 236, 154, 211, 10, 98, 34, 119, 56, 216, 119, 71, 218, 1164, 1496, 1817, 51, 39, 210, 36, 3, 19, 540, 232, 22, 141, 617, 84, 290, 80, 46, 207, 411, 150, 29, 38, 46, 172, 85, 194, 39, 261, 543, 897, 624, 18, 212, 416, 127, 931, 19, 4, 63, 96, 12, 101, 418, 16, 140, 230, 460, 538, 19, 27, 88, 612, 1431, 90, 716, 275, 74, 83, 11, 426, 89, 72, 84, 1300, 1706, 814, 221, 132, 40, 102, 34, 868, 975, 1101, 84, 16, 79, 23, 16, 81, 122, 324, 403, 912, 227, 936, 447, 55, 86, 34, 43, 212, 107, 96, 314, 264, 1065, 323, 428, 601, 203, 124, 95, 216, 814, 2906, 654, 820, 2, 301, 112, 176, 213, 71, 87, 96, 202, 35, 10, 2, 41, 17, 84, 221, 736, 820, 214, 11, 60, 760

Note

The inspiration for this challenge comes from the Damn Interesting website[8] and my love of the Nicholas Cage movie series "National Treasure". For more info see the Museum of Unnatural History[9].

[8] `http://www.damninteresting.com/89-263-201-500-337-480/`

[9] `http://www.unmuseum.org/bealepap.htm`

Python Solution

```
1  def solution(msg):
2      # msg is a comma-separated list of integers, just like beale wrote out
3      decl = "When in the course of human events ..." # omitted for brevity
4      msg = map(int, map(str.strip, msg1.split(',')))
5      return ''.join([ decl.split()[x-1][0] for x in msg])
```

1.21 Capitalize The First Letter of Every Word

Description

Given a sentence, can you capitalize the first letter of every word?

Yes this is a built-in in Python (`string.capwords`) and maybe some other languages, but the challenge is to implement your own capwords-like method.

Input Description

You'll be given an Enlish language sentence like this:

```
Now is the time for all great programmers to capitalize the correct words
    .
```

Output Description

You should emit a sentence with the first letter of every word capitalized.

```
Now Is The Time For All Great Programmers To Capitalize The Correct Words
    .
```

Challenge Input

```
Education is an admirable thing, but it is well to remember from time to
    time that nothing that is worth knowing can be taught.
An intelligent man is sometimes forced to be drunk to spend time with his
     fools.
The heart of a mother is a deep abyss at the bottom of which you will
    always find forgiveness.
All things are subject to interpretation whichever interpretation
    prevails at a given time is a function of power and not truth.
```

Challenge Output

```
Education Is An Admirable Thing, But It Is Well To Remember From Time To
    Time That Nothing That Is Worth Knowing Can Be Taught.
An Intelligent Man Is Sometimes Forced To Be Drunk To Spend Time With His
     Fools.
```

```
The Heart Of A Mother Is A Deep Abyss At The Bottom Of Which You Will
    Always Find Forgiveness.
All Things Are Subject To Interpretation Whichever Interpretation
    Prevails At A Given Time Is A Function Of Power And Not Truth.
```

Scala Solution

```
1  def capwords(s:String) = s.split(" ").map(_.capitalize).mkString(" ")
```

1.22 Collatz Conjecture

Description

The Collatz conjecture[10] is a conjecture in mathematics named after Lothar Collatz, who first proposed it in 1937. Take any natural number n. If n is even, divide it by 2 to get n / 2. If n is odd, multiply it by 3 and add 1 to obtain 3n + 1. Repeat the process (which has been called "Half Or Triple Plus One", or HOTPO[6]) indefinitely. The conjecture is that no matter what number you start with, you will always eventually reach 1.

For instance, starting with n = 6, one gets the sequence 6, 3, 10, 5, 16, 8, 4, 2, 1. n = 19, for example, takes longer to reach 1: 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Input

Your program should take a positive integer *N* as an argument.

Output Description

Your program should emit the number of steps it takes to reach 1 and the sequence emitted.

Bonus 1

Can you explain what's so interesting about the number 9232 in the context of the Collatz Conjecture?

Bonus 2

If you're feeling like it, throw in some unique visualizations of sequences or series from the Collatz Conjecture, using any imaging library you wish.

[10] https://en.wikipedia.org/wiki/Collatz_conjecture

Scala Solution

```
def collatz(N:Int): List[Int] = {
  def loop(n:Int, sofar:List[Int]): List[Int] = {
    n match {
      case 1 => 1::sofar
      case _ => (n%2 == 0) match {
          case true => loop(n/2, n::sofar)
          case false => loop(1 + 3*n, n::sofar)
      }
    }
  }
  loop(N, List()).reverse
}
```

1.23 Concatenated Integers

Description

Given a list of integers separated by a single space on standard input, print out the largest and smallest values that can be obtained by concatenating the integers together on their own line. This is from Five programming problems every Software Engineer should be able to solve in less than 1 hour[11], problem 4. Leading 0s are not allowed (e.g. 01234 is not a valid entry).

Sample Input

You'll be given a handful of integers per line. Example:

```
5 56 50
```

Sample Output

You should emit the smallest and largest integer you can make, per line. Example:

```
50556 56550
```

Challenge Input

```
79 82 34 83 69
420 34 19 71 341
17 32 91 7 46
```

Challenge Output

[11] https://blog.svpino.com/2015/05/07/five-programming-problems-every-software-engineer-should-be-able-to-solve-in-les

```
3469798283 8382796934
193413442071 714203434119
173246791 917463217
```

Scala Solution

```
1  // returns min, max
2  def intConcat(s:String): (Long, Long) = {
3      val l = s.split(" ").permutations.map(_.mkString.toLong).toList
4      (l.sorted.head, l.sorted.reverse.head)
5  }
```

1.24 Condensing Sentences

Description

Compression makes use of the fact that repeated structures are redundant, and it's more efficient to represent the pattern and the count or a reference to it. Siimilarly, we can *condense* a sentence by using the redundancy of overlapping letters from the end of one word and the start of the next. In this manner we can reduce the size of the sentence, even if we start to lose meaning.

For instance, the phrase "live verses" can be condensed to "liverses".

In this challenge you'll be asked to write a tool to condense sentences.

Input Description

You'll be given a sentence, one per line, to condense. Condense where you can, but know that you can't condense everywhere. Example:

```
I heard the pastor sing live verses easily.
```

Output Description

Your program should emit a sentence with the appropriate parts condensed away. Our example:

```
I heard the pastor sing liverses easily.
```

Challenge Input

```
Deep episodes of Deep Space Nine came on the television only after the
    news.
Digital alarm clocks scare area children.
```

Challenge Output

```
Deep episodes of Deep Space Nine came on the televisionly after the news.
Digitalarm clockscarea children.
```

1.25 Double (or More) Knots

Description

In knot tying, the double overhand knot is a common, basic knot that serves fishermen (and fisher-women) and sourgeons well. We can represent a single order double knot like this:

```
  __
 /  \
| /\ |
| \/ |
 \ \/
 /\ \
| /\ |
| \/ |
 \__/
```

And a double knot of 2 this way:

```
  __  __
 /  \/  \
| /\/ /\ |
| \/ /\/ |
 \ \/\ \/
 /\ \/\ \
| /\/ /\ |
| \/ /\/ |
 \__/\__/
```

Challenge Description

In this challenge, you'll be asked to draw various orders of ASCII double knots given an integer.

Challenge Input

```
2
6
```

Challenge Output

```
  __  __
 /  \/  \
| /\/ /\ |
| \/ /\/ |
```

```
 \ \/\ \/
 /\ \/\ \
| /\/ /\ |
| \/ /\/ |
 \__/\__/
```

```
  __  __  __  __  __  __
 /  \/  \/  \/  \/  \/  \
| /\/ /\/ /\/ /\/ /\/ /\ |
| \/ /\/ /\/ /\/ /\/ /\/ |
 \ \/\ \/\ \/\ \/\ \/\ \/
 /\ \/\ \/\ \/\ \/\ \/\ \
| /\/ /\/ /\/ /\/ /\/ /\ |
| \/ /\/ /\/ /\/ /\/ /\/ |
 \__/\__/\__/\__/\__/\__/
```

1.26 Emirp Numbers

Description

We all know what prime numbers are - numbers only divisible by themselves and one. Math grad student Matthew Scroggs came up with *emirp* numbers. An emirp number[12] is a prime number which is a different prime number when the digits are reversed.

Your task today is to write a program that can discover emirp numbers over a range.

Input Description

You'll be given two numbers, *a* and *b*, that represent the range *inclusive* of the numbers to screen for emirp numbers. Example:

```
10 100
```

Output Description

Your program should emit the list of valid emirp numbers for that range. Example:

```
[11; 13; 17; 31; 37; 71; 73; 79; 97]
```

Challenge Input

```
10000 10100
999810 999999
```

Challenge Output

[12] https://en.wikipedia.org/wiki/Emirp

```
[10007; 10009; 10039; 10061; 10067; 10069; 10079; 10091]
[999853; 999931; 999983]
```

FSharp Solution

```
let isprime (n:int) =
    let rec check i =
        i > n/2 || (n % i <> 0 && check (i + 1))
    check 2;;

let revnum(n:int) =
    string(n).ToCharArray() |> Array.rev |> Array.map(fun x -> string(x)) |>
    String.concat "" |> int


[ 1 .. 100 ] |> List.filter isprime |> List.filter (fun x -> revnum x |>
    isprime)
```

1.27 Extravagant Numbers

Description

An extravagant number[13] (also known as a wasteful number) is a natural number that has fewer digits than the number of digits in its prime factorization (including exponents). Trivial examples include 6 = 2*3, 8 = 2^3, and 9 = 3^2, all extravagant numbers.

Your challenge today is to write a program to determine is numbers are extravagant or not.

Input Description

You'll be given a single integer per line. Examples:

```
6
16
32
99
```

Output Description

Your program should emit if the number is extravagant or not:

```
6 EXTRAVAGANT
16 NOT EXTRAVAGANT
32 NOT EXTRAVAGANT
99 EXTRAVAGANT
```

[13] https://en.wikipedia.org/wiki/Extravagant_number

Challenge Input

```
90
30
74
141
782
938
```

Challenge Output

```
90 EXTRAVAGANT
30 EXTRAVAGANT
74 EXTRAVAGANT
141 NOT EXTRAVAGANT
782 EXTRAVAGANT
938 EXTRAVAGANT
```

Scala Solution

```
1  def factorize(x: Int): List[Int] = {
2      @tailrec
3      def foo(x: Int, a: Int = 2, list: List[Int] = Nil): List[Int] = a*a > x
   ↪   match {
4        case false if x % a == 0 => foo(x / a, a    , a :: list)
5        case false               => foo(x    , a + 1, list)
6        case true                => x :: list
7      }
8      foo(x)
9    }
10
11 def power(n:Int): String = {
12     if (n > 1) {return n.toString}
13     else {return ""}
14 }
15
16 def extravagant(n:Int): Boolean =
17     factorize(n).groupBy(x => x).mapValues(_.length).map(x => x._1 + "" +
   ↪   power(x._2)).mkString("").length > n.toString.length
```

1.28 Finding the Freshest Eggs

Description

The fresher the egg, the better the flavor. Because the sell-by date for eggs in a supermarket (with U.S.D.A. inspection) can be up to 45 days after the packing date, there is a quick and easy way to check for freshness: the Julian date. Every egg carton has a code printed on its side, and the last 3 digits of this code are called

the Julian date. The code uses a number from 001 to 365 to correspond to a day of the year and indicate when they were packaged. For example, 001 is January 1st and 365 is December 31st.

For this challenge, your goal is to read the ISO calendar date (e.g. 2015-03-28) and then the Julian date code from a number of egg cartons and to pick the freshest one, the one that was packaged the fewest days ago.

Input Description

For each scenario you'll be given an integer *N* which tells you how many egg cartons to check, then on the next line an ISO date format of the current date. Then you'll be given *N* egg cartons' worth of Julian dates.

You'll be given 3 sets of scenarios that match these parameters. Solve them all independently.

You should work with the assumption that time travel is impossible, so keep that in mind working with Julian dates.

Output Description

Your program should emit the scenario's date and the Julian date of the freshest egg carton and the oldest egg carton (maybe ripe for removal from the shelf!).

Challenge Input

Scenario 1:

```
5
2015-03-28
019
026
017
041
063
```

Scenario 2:

```
5
2014-01-01
311
163
270
229
162
```

Scenario 3:

```
5
2015-01-10
321
004
```

```
354
009
337
```

Challenge Output

```
2015-03-28 063 017
2014-01-01 311 162
2015-01-10 009 321
```

1.29 Basic Graph Statistics: Node Degrees

Description

In graph theory, the *degree* of a node is the number of edges coming into it or going out of it - how connected it is. For this challenge you'll be calculating the degree of every node.

Input Description

First you'll be given an integer, *N*, on one line showing you how many nodes to account for. Next you'll be given an undirected graph as a series of number pairs, *a* and *b*, showing that those two nodes are connected - an edge. Example:

```
3
1 2
1 3
```

Output Description

Your program should emit the degree for each node. Example:

```
Node 1 has a degree of 2
Node 2 has a degree of 1
Node 3 has a degree of 1
```

Challenge Input

This data set is an social network of tribes of the Gahuku-Gama alliance structure of the Eastern Central Highlands of New Guinea, from Kenneth Read (1954). The dataset contains a list of all of links, where a link represents signed friendships between tribes. It was downloaded from the network repository[14].

[14] http://networkrepository.com/soc_tribes.php

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

```
1 16
2 16
5 16
6 16
11 16
12 16
13 16
14 16
15 16
```

Challenge Output

```
Node 1 has a degree of 8
Node 2 has a degree of 8
Node 3 has a degree of 6
Node 4 has a degree of 3
Node 5 has a degree of 7
Node 6 has a degree of 10
Node 7 has a degree of 7
Node 8 has a degree of 7
Node 9 has a degree of 7
Node 10 has a degree of 5
Node 11 has a degree of 9
Node 12 has a degree of 8
Node 13 has a degree of 8
Node 14 has a degree of 5
Node 15 has a degree of 9
Node 16 has a degree of 9
```

Bonus: Adjascency Matrix

Another tool used in graph theory is an *adjacency matrix*, which is an *N* by *N* matrix where each *(i,j)* cell is filled out with the degree of connection between nodes *i* and *j*. For our example graph above the adjacency matrix would look like this:

```
0 1 1
1 0 0
1 0 0
```

Indicating that node 1 is connected to nodes 2 and 3, but nodes 2 and 3 do not connect. For a bonus, create the adjacency matrix for the challenge graph.

Scala Solution

```
1  def degree(edges:String) =
2      edges.
3        split("\n").
4        map(_.split(" ").filter(_.length>0)).
```

```
      toSet.
      toList.
      flatten.
      groupBy(_.toString).
      mapValues(_.size)

def adj_matrix(edges:String, n:Int):String = {
    val m = Array.ofDim[Int](n,n)
    val es = edges.
               split("\n").
               map(_.split(" ").filter(_.length>0)).
               map(_.map(_.toInt))
    for (e <- es) { m(e(0)-1)(e(1)-1) = 1; m(e(1)-1)(e(0)-1) = 1 }
    m.map(_.mkString(" ")).mkString("\n")
}

def challenge(edges:String) =
    degree(edges).foreach { kv => println(kv._1 + " has a degree of " +
 ↪  kv._2) }

def bonus(edges:String, n:Int) = {
    challenge(edges)
    println(adj_matrix(edges, n))
}
```

Go Solution

```
package main

import (
    "fmt"
    "io/ioutil"
    "os"
    "strconv"
    "strings"
)

func check(e error) {
    if e != nil {
        panic(e)
    }
}

func main() {
    bdata, err := ioutil.ReadFile(os.Args[1])
    check(err)
```

```
20
21      data := string(bdata)
22      var nodes map[string]int
23      nodes = make(map[string]int)
24
25      // calcuate node degree
26      lines := strings.Split(data, "\n")
27      for _, line := range lines {
28          vals := strings.Split(line, " ")
29          if len(vals) == 2 {
30              nodes[vals[0]] = nodes[vals[0]] + 1
31              nodes[vals[1]] = nodes[vals[1]] + 1
32          }
33      }
34      i := 0
35      for k, v := range nodes {
36          fmt.Printf("Node %s has a degree of %d\n", k, v)
37          i = i + 1
38      }
39
40      // bonus adjacency matrix
41      adjm := make([][]string, i)
42      for n := range adjm {
43          adjm[n] = make([]string, i)
44          for m := range adjm[n] {
45              adjm[n][m] = "0"
46          }
47      }
48      for _, line := range lines {
49          vals := strings.Split(line, " ")
50          if len(vals) == 2 {
51              x, err := strconv.ParseUint(vals[0], 10, 32)
52              check(err)
53              y, err := strconv.ParseUint(vals[1], 10, 32)
54              check(err)
55              adjm[x-1][y-1] = "1"
56              adjm[y-1][x-1] = "1"
57              adjm[x-1][x-1] = "1"
58          }
59      }
60
61      for n := 0; n < i; n++ {
62          fmt.Printf("%q\n", strings.Join(adjm[n], " "))
63      }
64  }
```

1.30 Harshad Number

Description

In recreational mathematics, a Harshad number[15] (or Niven number) in a given number base, is an integer that is divisible by the sum of its digits when written in that base. Harshad numbers in base n are also known as *n*-Harshad (or *n*-Niven) numbers. Harshad numbers were defined by D. R. Kaprekar, a mathematician from India. The word "Harshad" comes from the Sanskrit *harsa* (joy) + *da* (give), meaning joy-giver. The term "Niven number" arose from a paper delivered by Ivan M. Niven at a conference on number theory in 1977.

Thus, you can observe that number 21 is a Harshad number - 21 / (2 + 1) = 7.

Today's challenge is to determine if an integer is a valid Harshad number.

Input Description

You'll be given an integer, one per line. Example:

```
21
22
```

Output Description

Your program should emit if the number is a Harshad number or not. Example:

```
21 HARSHAD
22 NOT HARSHAD
```

Challenge Input

```
21
49
62
63
```

Challenge Output

```
21 HARSHAD
49 NOT HARSHAD
62 NOT HARSHAD
63 HARSHAD
```

Scala Solution

[15] https://en.wikipedia.org/wiki/Harshad_number

```
def harshad(n:Int): Boolean =
  ↪  (n/n.toString.map(_.toString.toInt).sum)*(n.toString.map(_.toString.toInt).sum)
  ↪  == n
```

1.31 Integer Sequence Search Part 1

Description

In mathematics, an integer sequence is a sequence (i.e., an ordered list) of integers. Not all sequences are computable (e.g. not all have a formula that can express them), but unique sequences have interesting properties and can be quite fun to watch play out.

The On-Line Encyclopedia of Integer Sequences (OEIS)[16] website has an interesting feature where you can search for sequences by name, ID, or even just subsequences.

For this challenge you'll be replicating that subsequence search feature.

Input Description

You'll be given two integers, *N* and *M*, which tell you how many sequences to read to form your database and then how many search queries to process, respectively. Then you'll be given the database as *N* pairs of *name* and *first 10 terms of the sequence* pair. Then you'll be given *M* queries of a series of integers. Note that the overlap of the query and the sequence database will be unambiguous but is not guaranteed to overlap completely. All sequences to search will be contiguous (e.g. no gaps). Sequence names will use the OEIS naming convention

Example:

```
1 2
A000055 1, 1, 1, 1, 2, 3, 6, 11, 23, 47
11, 23, 47, 106
3, 14, 15, 92, 65
```

Output Description

For each of the search terms you should emit the query and the sequence name.

Example:

```
11, 23, 47, 106 A000055
3, 14, 15, 92, 65 NOMATCH
```

[16] https://oeis.org/

Challenge Input

```
9 6
A000055 1, 1, 1, 1, 2, 3, 6, 11, 23, 47
A000045 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
A050278 1023456789, 1023456798, 1023456879, 1023456897, 1023456978,
    1023456987, 1023457689, 1023457698, 1023457869, 1023457896
A000010 1, 1, 2, 2, 4, 2, 6, 4, 6, 4
A194508 -1, 1, 0, 2, 1, 0, 2, 1, 3, 2
A000111 1, 1, 1, 2, 5, 16, 61, 272, 1385, 7936
A233586 1, 6, 12, 19, 63, 263, 856, 2632, 7714, 9683
A000391 1, 6, 21, 71, 216, 672, 1982, 5817, 16582, 46633
A000713 1, 3, 8, 18, 38, 74, 139, 249, 434, 734
1, 3, 8, 18
263, 856, 2632, 7714, 9683
1, 1, 2, 3, 5, 8
1, 6, 12, 19, 63
20, 22, 24, 28, 30, 34
434, 734, 1215, 1967, 3132, 4902
```

Challenge Output

```
1, 3, 8, 18    A000713
263, 856, 2632, 7714, 9683 A233586
1, 1, 2, 3, 5, 8  A000045
1, 6, 12, 19, 63  A233586
20, 22, 24, 28, 30, 34 NOMATCH
434, 734, 1215, 1967, 3132, 4902 A000713
```

1.32 Jolly Jumper

Description

A sequence of n $>$ 0 integers is called a jolly jumper if the absolute values of the differences between successive elements take on all possible values 1 through n - 1. For instance,

```
1 4 2 3
```

is a jolly jumper, because the absolute differences are 3, 2, and 1, respectively. The definition implies that any sequence of a single integer is a jolly jumper. Write a program to determine whether each of a number of sequences is a jolly jumper.

Input Description

You'll be given a row of numbers. The first number tells you the number of integers to calculate over, *N*, followed by *N* integers to calculate the differences. Example:

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

Output Description

Your program should emit some indication if the sequence is a jolly jumper or not. Example:

```
4 1 4 2 3 JOLLY
8 1 6 -1 8 9 5 2 7 NOT JOLLY
```

Challenge Input

```
4 1 4 2 3
5 1 4 2 -1 6
4 19 22 24 21
4 19 22 24 25
4 2 -1 0 2
```

Challenge Output

```
4 1 4 2 3 JOLLY
5 1 4 2 -1 6 NOT JOLLY
4 19 22 24 21 NOT JOLLY
4 19 22 24 25 JOLLY
4 2 -1 0 2 JOLLY
```

1.33 L33tspeak Translator

Description

L33tspeak - the act of speaking like a computer hacker (or hax0r) - was popularized in the late 1990s as a mechanism of abusing ASCII art and character mappings to confuse outsiders. It was a lot of fun. One popuar comic strip[17] in 2000 showed just how far the joke ran.

In L33Tspeak you substitute letters for their rough outlines in ASCII characters, e.g. symbols or numbers. You can have 1:1 mappings (like E -> 3) or 1:many mappings (like W -> '//). So then you wind up with words like this:

```
BASIC => 6451C
ELEET => 31337 (pronounced elite)
WOW => '//0'//
MOM => (V)0(V)
```

Mappings

For this challenge we'll be using a subset of American Standard Leetspeak:

[17] http://megatokyo.com/strip/9

```
A -> 4
B -> 6
E -> 3
I -> 1
L -> 1
M -> (V)
N -> (\)
O -> 0
S -> 5
T -> 7
V -> \/
W -> `//
```

Your challenge, should you choose to accept it, is to translate to and from L33T.

Input Description

You'll be given a word or a short phrase, one per line, and asked to convert it from L33T or to L33T. Examples:

```
31337
storm
```

Output Description

You should emit the translated words: Examples:

```
31337 -> eleet
storm -> 570R(V)
```

Challenge Input

```
I am elite.
Da pain!
Eye need help!
3Y3 (\)33d j00 t0 g37 d4 d0c70r.
1 n33d m4 p1llz!
```

Challenge Output

```
I am elite. -> 1 4m 37173
Da pain! -> D4 P41(\)!
Eye need help! -> 3Y3 (\)33D H31P!
3Y3 (\)33d j00 t0 g37 d4 d0c70r. -> Eye need j00 to get da doctor.
1 n33d m4 p1llz! -> I need ma pillz!
```

1.34 Lipogram Detector

Description

A lipogram is a kind of constrained writing or word game consisting in writing paragraphs or longer works in which a particular letter or group of letters is avoided. Writing a lipogram may be a trivial task when avoiding uncommon letters like Z, J, Q, or X, but it is much more difficult to avoid common letters like E, T or A, as the author must omit many ordinary words. A famous example is Poe's poem *The Raven* contains no Z, but there is no evidence that this was intentional. Pangrammatic lipograms use all letters except one.

Your challenge today is to detect what letter is missing from the given text.

Input Description

You'll be given a short piece of text. For example:

```
A jovial swain should not complain
Of any buxom fair
Who mocks his pain and thinks it gain
To quiz his awkward air.
```

Output Description

Your program should emit what letter is missing. From ths above example:

```
E
```

Challenge Input

```
Bold Nassan quits his caravan,
A hazy mountain grot to scan;
Climbs jaggy rocks to find his way,
Doth tax his sight, but far doth stray.

Not work of man, nor sport of child
Finds Nassan on this mazy wild;
Lax grow his joints, limbs toil in vain-
Poor wight! why didst thou quit that plain?

Vainly for succour Nassan calls;
Know, Zillah, that thy Nassan falls;
But prowling wolf and fox may joy
To quarry on thy Arab boy.
```

Challenge Output

```
E
```

Scala Solution

```
def lipogram(text: String) : Set[Char] =

  ↪  "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toSet--text.toCharArray.map(_.toUpper).toSet
```

Python Solution

```
def lipogram(text):
    return set(string.lowercase) - ( { ch.lower() for ch in text } -
 ↪  set(string.punctuation))
```

Go Solution

```
package main

import (
    "fmt"
    "gopkg.in/fatih/set.v0"
    "os"
    "strings"
)

func main() {
    const alphabet = "abcdefghijklmnopqrstuvwxyz"
    text := os.Args[1]

    characters := set.New()
    for _, ch := range strings.ToLower(text) {
        characters.Add(string(ch))
    }

    alpha := set.New()
    for _, ch := range alphabet {
        alpha.Add(string(ch))
    }

    fmt.Println(set.Difference(alpha, characters))
}
```

1.35 Pandigital Roman Numbers

Description

1474 is a pandigital in Roman numerals (MCDLXXIV). It uses each of the symbols I, V, X, L, C, and M at least once. Your challenge today is to find the small handful

of pandigital Roman numbers up to 2000.

Output Description

A list of numbers. Example:

```
1 (I), 2 (II), 3 (III), 8 (VIII) (Examples only, these are not pandigital
     Roman numbers)
```

Challenge Input

Find all numbers that are pandigital in Roman numerals using each of the symbols I, V, X, L, C, D and M *exactly* once.

Challenge Input Solution

1444, 1446, 1464, 1466, 1644, 1646, 1664, 1666

See OEIS sequence A105416[18] for more information.

1.36 Pell Numbers

Description

In mathematics, the Pell numbers[19] are an infinite sequence of integers, known since ancient times, that comprise the denominators of the closest rational approximations to the square root of 2. This sequence of approximations begins 1/1, 3/2, 7/5, 17/12, and 41/29, so the sequence of Pell numbers begins with 0, 1, 2, 5, 12, and 29 (each Pell number is the sum of twice the previous Pell number and the Pell number before that).

Your challenge today is to generate this sequence and pick out specific elements in the seqence.

If you're feeling especially brave, try applying memoization and recursion in your answer.

Sample Input

You'll be given number *N*, one per line. This is the position in the sequence of Pell numbers to yield. Examples:

```
1 3
2 5
```

[18] http://oeis.org/A105416

[19] https://en.wikipedia.org/wiki/Pell_number

Sample Output

2
12

Challenge Input

10
17

Challenge Output

985
470832

Bonus

What is the 100th Pell number? (Answer: 27749033099085295754434173207717704165)

F# Solution

```
let pell n =
    let addPell a b = (2I*a + b)
    let rec loop n sofar =
        printfn "%A" sofar
        match ((List.length sofar) = n) with
        | true  -> List.head sofar
        | false -> loop n ((addPell (List.head sofar) (Seq.nth 1
 ↪  sofar))::sofar)
    match n with
    | 0 -> 0I
    | 1 -> 1I
    | _ -> loop n [1I; 0I]
```

1.37 Perfect Numbers

Description

In number theory, a perfect number[20] is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself (also known as its aliquot sum). The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and $1 + 2 + 3 = 6$. Equivalently, the number 6 is equal to half the sum of all its positive divisors: $(1 + 2 + 3 + 6) / 2 = 6$. The next perfect number is $28 = 1 + 2 + 4 + 7 + 14$. This is followed by the perfect numbers 496 and 8128.

In this challenge you'll be asked to calculate an arbitrary number of perfect numbers.

[20] http://en.wikipedia.org/wiki/Perfect_number

Input Description

You'll be given a single integer, *N*, which tells you how many perfect numbers to emit. Example:

```
4
```

Output Description

Your program should emit a list of perfect numbers up to that point. For our example:

```
6 28 496 8128
```

Challenge Input

```
10
```

Challenge Output

```
6 28 496 8128 33550336 8589869056 137438691328 2305843008139952128
    2658455991569831744654692615953842176
    191561942608236107294793378084303638130997321548169216
```

Scala Solution

a naive solution that doesn't use the Euclid-Euler theorem

```
1  def factors(n:Int): List[Int] = (2 to n/2).filter(x => n%x == 0).toList
2
3  def perfect(n:Int): Boolean = n == factors(n).sum + 1
4
5  (1 to 10000).filter(perfect(_))
```

getting closer to using Euclid-Euler

```
def isprime(n:Int) : Boolean = {
    def check(i:Int) : Boolean = (i > n/2) || ((n % i != 0) && (check (i
        +1)))
    check(2)
}

def perfect(n:Int): Long = (scala.math.pow(2, (n-1))*(scala.math.pow(2, n
    )-1)).toLong

(2 to 50000).filter(isprime(_)).map(perfect(_)).distinct
```

1.38 Primes in Grids

Description

This puzzle was first proposed (1989) by Gordon Lee: given a grid of numbers, how many *distinct* primes can you find embedded in the matrix, regarding that you can read the lines or part of them, in form vertical, horizontal or diagonal orientation, in both directions.

Note that you can't change direction once you start moving (e.g. this isn't Boggle).

Input Description

You'll be given a single number on a line which tells you how many rows and columns to read (all grids will be square). Example:

```
3
113
754
937
```

Output Description

Your program should emit the number of distinct primes it finds in the grid. Optionally list them. Example:

```
30
113, 311, 179, 971, 157, 751 359, ...
```

Challenge Input

```
5
11933
99563
89417
33731
32939

6
317333
995639
118142
136373
349199
379379
```

Challenge Output

116

187

1.39 Reverse Factorial

Description

Nearly everyone is familiar with the factorial operator in math. 5! yields 120 because factorial means "multiply successive terms where each are one less than the previous":

```
5! -> 5 * 4 * 3 * 2 * 1 -> 120
```

Simple enough.

Now let's reverse it. Could you write a function that tells us that "120" is "5!"?

Hint: The strategy is pretty straightforward, just divide the term by successively larger terms until you get to "1" as the resultant:

```
120 -> 120/2 -> 60/3 -> 20/4 -> 5/5 -> 1 => 5!
```

Sample Input

You'll be given a single integer, one per line. Examples:

```
120
150
```

Sample Output

Your program should report what each number is as a factorial, or "NONE" if it's not legitimately a factorial. Examples:

```
120 = 5!
150  NONE
```

Challenge Input

```
3628800
479001600
6
18
```

Challenge Output

```
3628800 = 10!
479001600 = 12!
6 = 3!
18  NONE
```

Fsharp Solution

```
let rec tcaf(n: int) (sofar: int) =
    match (n%sofar) with
    | 0 ->  match (n/sofar) with
            | 1 -> sofar
            | _ -> tcaf (n/sofar) (sofar+1)
    | _ -> -1

let solution (n: int) =
    let res = tcaf n 2
    match res with
    | -1 -> "NONE"
    | _  -> (string res) + "!"
```

1.40 Roller Coaster Words

Description

A roller coaster word[21] is a word with letters that alternate between going forward and backward in alphabet. One such word is "decriminalization". Can you find other examples of roller coaster words in the English dictionary?

Output

Your program should emit any and all roller coaster words it finds in a standard English language dictionary longer (or enable1.txt[22]) than 4 letters.

Notes

If you have your own idea for a challenge, submit it to /r/DailyProgrammer_Ideas, and there's a good chance we'll post it.

1.41 Safe Prime Numbers

Description

A safe prime[23] is a prime number of the form $2p + 1$, where p is also a prime. Safe primes are also important in cryptography because of their use in discrete logarithm-based techniques like Diffie-Hellman key exchange.

[21] http://www.questrel.com/records.html#spelling_alphabetical_order_entire_word_roller-coaster

[22] https://github.com/dolph/dictionary/blob/master/enable1.txt

[23] https://en.wikipedia.org/wiki/Safe_prime

Input Description

You will be given a single number that is the maximum value of safe prime to search for. Example:

```
100
```

Output Description

A list of numbers, one on each line, showing numbers that solve the safe prime definition. Example:

```
5
7
11
23
47
59
83
```

Challenge Input

```
1000
```

Challenge Input Solution (not visible by default)

```
5
7
11
23
47
59
83
107
167
179
227
263
347
359
383
467
479
503
563
587
719
839
863
```

887
983

FSharp Solution

```
> let isprime (n:int) =
-     let n = bigint(n)
-     let rec check i =
-         i > n/2I || (n % i <> 0I && check (i + 1I))
-     check 2I
-- ;;


> let safeprimes(n:int) =
-     [ 2..n ] |> List.filter (fun x ->isprime(x) && isprime(1+2*x) )
- ;;

val safeprimes : n:int -> int list

> safeprimes 100 ;;
val it : int list = [2; 3; 5; 11; 23; 29; 41; 53; 83; 89]
> safeprimes 1000 ;;
val it : int list =
  [2; 3; 5; 11; 23; 29; 41; 53; 83; 89; 113; 131; 173; 179; 191; 233; 239;
   251;
   281; 293; 359; 419; 431; 443; 491; 509; 593; 641; 653; 659; 683; 719;
   743;
   761; 809; 911; 953]
```

1.42 Calculating Shannon Entropy of a String

Description

Shannon entropy was introduced by Claude E. Shannon in his 1948 paper "A Mathematical Theory of Communication". Somewhat related to the physical and chemical concept entropy, the Shannon entropy measures the uncertainty associated with a random variable, i.e. the expected value of the information in the message (in classical informatics it is measured in bits). This is a key concept in information theory and has consequences for things like compression, cryptography and privacy, and more.

The Shannon entropy H is calculated as -1 times the sum of the frequency of the symbol times the log base 2 of the frequency:

$$H(X) = -1 * \sum_{i=1}^{n} \frac{count(i)}{N} * \log_2 \left(\frac{count(i)}{N}\right)$$

For more, see Wikipedia for Entropy in information theory[24].

Input Description

You'll be given a string, one per line, for which you should calculate the Shannon entropy. Examples:

```
1223334444
Hello, world!
```

Output Description

Your program should emit the calculated entropy values for the strings to at least five decimal places. Examples:

```
1.84644
3.18083
```

Challenge Input

```
122333444455555666666777777788888888
563881467447538846567288767728553786
https://www.reddit.com/r/dailyprogrammer
int main(int argc, char *argv[])
```

Challenge Output

```
2.794208683
2.794208683
4.056198332
3.866729296
```

Python Solution

```python
import math

from collections import Counter

def entropy(s):
    p, lns = Counter(s), float(len(s))
    return -sum( count/lns * math.log(count/lns, 2) for count in p.values())
```

FSharp Solution

[24] https://en.wikipedia.org/wiki/Entropy_(information_theory)

```
let entropy (s) : float =
    let p = string(s).ToCharArray()
            |> Seq.groupBy (fun x -> x)
            |> Seq.map (fun (x,y) -> Seq.length y)
    -1.0 * ([ for count in p ->
                float(count)/float(String.length(s)) *
                System.Math.Log(float(count)/float(String.length(s)), 2.0) ]
            |> Seq.sum )
```

1.43 Typo Maker

Description

Typos are great fun, and often follow a common pattern - keys next to eachother, doubled or omitted characters, and transpositions. Can you write a program to generate common typos? If so, you could be on your way to typo-squatting in DNS!

Common typos fall into a few different categories:

- Skip letter
- Double letters
- Reverse (transpose) letters
- Skip spaces
- Missed key
- Inserted key

For this challege, when you think about neighbor keys, assume a [QWERTY keyboard layout] (http://en.wikipedia.org/wiki/QWERTY).

Input Description

You'll be given a word, one per line, and asked to generate typos for it. Example:

```
typo
```

Output Description

Your program should emit the word mangled into its various formats using the above categories. Our example:

```
tpo
ypo
typ
ttypo
tyypo
typpo
typoo
tyop
ytpo
toyp
rypo
yypo
ttpo
tupo
ty[o
tyoo
typi
typp
trypo
rtypo
... (omitted for brevity)
```

Input Description

```
Facebook
Google
Global thermonuclear war
Dead as a doornail
Britney Spears
Cappuccino
Everybody to the limit
```

1.44 Wedderburn-Etherington Sequence

Description

The Wedderburn-Etherington numbers are an integer sequence named for Ivor Malcolm Haddon Etherington and Joseph Wedderburn that can be used to count certain kinds of binary trees. The first few numbers in the sequence are

```
0, 1, 1, 1, 2, 3, 6, 11, 23, 46, 98, 207, 451 ...
```

The Wedderburnâ–Etherington numbers may be calculated using the recurrence relation (in LaTeX notation)

```
a_{2n-1} = \sum\limits{i=1}^n-1 a_ia_{2n-i-1}

a_{2n} = a_n(a_n+1)/2 + \sum\limits{i=1}^n-1 a_ia_{2n-i}
```

See Wikipedia for more on the Wedderburn-Etherington number[25] and its uses.

Input Description

You'll be given a number *n*, the number in the sequence to generate.

Output Description

A sequence of integers in the Wedderburn-Etherington sequence up to position *n*.

1.45 XOR Multiplication

Description

One way to think about bitwise *addition* (using the symbol ^) as binary addition without carrying the extra bits:

```
  101   5
^ 1001  9
  ----
  1100  12

  5^9=12
```

So let's define XOR multiplcation (we'll use the symbol @) in the same way, the addition step doesn't carry:

```
    1110  14
  @ 1101  13
   -----
    1110
      0
   1110
^ 1110
  ------
  1000110 70

  14@13=70
```

For this challenge you'll get two non-negative integers as input and output or print their XOR-product, using both binary and decimal notation.

Input Description

You'll be given two integers per line. Example:

```
5 9
```

[25] http://en.wikipedia.org/wiki/Wedderburn%E2%80%93Etherington_number

Output Description

You should emit the equation showing the XOR multiplcation result:

```
5@9=12
```

Challenge Input

```
1 2
9 0
6 1
3 3
2 5
7 9
13 11
5 17
14 13
19 1
63 63
```

Challenge Output

```
1@2=2
9@0=0
6@1=6
3@3=5
2@5=10
7@9=63
13@11=127
5@17=85
14@13=70
19@1=19
63@63=1365
```

Chapter 2

Intermediate

Introduction

Intermediate challenges are meant to help you stretch your chops in your language of choice. You may have to dig around a few esoteric corners of your language, but are still just a programming exercise - they have a clear path to a solution in whatever language you choose.

2.1 Where Should Grandma's House Go?

Description

My grandmother and I are moving to a new neighborhood. The houses haven't yet been built, but the map has been drawn. We'd like to live as close together as possible. She makes some outstanding cookies, and I love visiting her house on the weekend for delicious meals - my grandmother is probably my favorite cook!

Please help us find the two lots that are closest together so we can build our houses as soon as possible.

Example Input

You'll be given a single integer, *N*, on a line, then *N* lines of Cartesian coordinates of (x,y) pairs. Example:

```
16
(6.422011725438139, 5.833206713226367)
(3.154480546252892, 4.063265532639129)
(8.894562467908552, 0.3522346393034437)
(6.004788746281089, 7.071213090379764)
(8.104623252768594, 9.194871763484924)
(9.634479418727688, 4.005338324547684)
(6.743779037952768, 0.7913485528735764)
```

```
(5.560341970499806, 9.270388445393506)
(4.67281620242621, 8.459931892672067)
(0.30104230919622, 9.406899285442249)
(6.625930036636377, 6.084986606308885)
(9.03069534561186, 2.3737246966612515)
(9.3632392904531, 1.8014711293897012)
(2.6739636897837915, 1.6220708577223641)
(4.766674944433654, 1.9455404764480477)
(7.438388978141802, 6.053689746381798)
```

Example Output

Your program should emit the two points of (x,y) pairs that are closest together. Example:

```
(6.625930036636377,6.084986606308885)
    (6.422011725438139,5.833206713226367)
```

Challenge Input

```
100
(5.558305599411531, 4.8600305440370475)
(7.817278884196744, 0.8355602049697197)
(0.9124479406145247, 9.989524754727917)
(8.30121530830896, 5.0088455259181615)
(3.8676289528099304, 2.7265254619302493)
(8.312363982415834, 6.428977658434681)
(2.0716308507467573, 4.39709962385545)
(4.121324567374094, 2.7272406843892005)
(9.545656436023116, 2.874375810978397)
(2.331392166597921, 0.7611494627499826)
(4.241235371900736, 5.54066919094827)
(3.521595862125549, 6.799892867281735)
(7.496600142701988, 9.617336260521792)
(2.5292596863427796, 4.6514954819640035)
(8.9365560770944, 8.089768281770253)
(8.342815293157892, 1.3117716484643926)
(6.358587371849396, 0.7548433481891659)
(1.9085858694489566, 1.2548184477302327)
(4.104650644200331, 5.1772760616934645)
(6.532092345214275, 8.25365480511137)
(1.4484096875115393, 4.389832854018496)
(9.685268864302843, 5.7247619715577915)
(7.277982280818066, 3.268128640986726)
(2.1556558331381104, 7.440500993648994)
(5.594320635675139, 6.636750073337665)
(2.960669091428545, 5.113509430176043)
(4.568135934707252, 8.89014754737183)
```

```
(4.911111477474849, 2.1025489963335673)
(8.756483469153423, 1.8018956531996244)
(1.2275680076218365, 4.523940697190396)
(4.290558055568554, 5.400885500781402)
(8.732488819663526, 8.356454134269345)
(6.180496817849347, 6.679672206972223)
(1.0980556346150605, 9.200474664842345)
(6.98003484966205, 8.22081445865494)
(1.3008030292739836, 2.391081348654746 6)
(0.8176167873315643, 3.664910265751047)
(4.707575761419376, 8.48393210654012)
(2.574624846075059, 6.638825467263861)
(0.5055608733353167, 8.040212389937379)
(3.905281319431256, 6.158362777150526)
(6.517523776426172, 6.758027776767626)
(6.946135743246488, 2.245153765579998)
(6.797442280386309, 7.70803829544593)
(0.5188505776214936, 0.1909838711203915)
(7.896980640851306, 4.366680008699691)
(1.2404651962738256, 5.963706923183244)
(7.9085889544911945, 3.501907219426883)
(4.829123686370425, 6.116328436853205)
(8.703429477346157, 2.494600359615746)
(6.9851545945688684, 9.241431992924019)
(1.8865556630758573, 0.14671871143506765)
(4.237855680926536, 1.4775578026826663)
(3.8562761635286913, 6.487067768929168)
(5.8278084663109375, 5.98913080157908)
(8.744913811001137, 8.208176389217819)
(1.1945941254992176, 5.832127086137903)
(4.311291521846311, 7.670993787538297)
(4.403231327756983, 6.027425952358197)
(8.496020365319831, 5.059922514308242)
(5.333978668303457, 5.698128530439982)
(9.098629270413424, 6.8347773139334675)
(7.031840521893548, 6.705327830885423)
(9.409904685404713, 6.884659612909266)
(4.750529413428252, 7.393395242301189)
(6.502387440286758, 7.5351527902895965)
(7.511382341946669, 6.768903823121008)
(7.508240643932754, 6.556840482703067)
(6.997352867756065, 0.9269648538573272)
(0.9422251775272161, 5.103590106844054)
(0.5527353428303805, 8.586911807313664)
(9.631339754852618, 2.6552168069445736)
(5.226984134025007, 2.8741061109013555)
(2.9325669592417802, 5.951638270812146)
(9.589378643660075, 3.2262646648108895)
(1.090723228724918, 1.3998921986217283)
(8.364721356909339, 3.2254754023019148)
```

```
(0.7334897173512944, 3.8345650175295143)
(9.715154631802577, 2.153901162825511)
(8.737338862432715, 0.9353297864316323)
(3.9069371008200218, 7.486556673108142)
(7.088972421888375, 9.338974320116852)
(0.5043493283135492, 5.676095496775785)
(8.987516578950164, 2.500145166324793)
(2.1882275188267752, 6.703167722044271)
(8.563374867122342, 0.0034374051899066504)
(7.22673935541426, 0.7821487848811326)
(5.305665745194435, 5.6162850431000875)
(3.7993107636948267, 1.3471479136817943)
(2.0126321055951077, 1.6452950898125662)
(7.370179253675236, 3.631316127256432)
(1.9031447730739726, 8.674383934440593)
(8.415067672112773, 1.6727089997072297)
(6.013170692981694, 7.931049747961199)
(0.9207317960126238, 0.17671002743311348)
(3.534715814303925, 5.890641491546489)
(0.611360975385955, 2.9432460366653213)
(3.94890493411447, 6.248368129219131)
(8.358501795899047, 4.655648268959565)
(3.597211873999991, 7.184515265663337)
```

Challenge Output

```
(5.305665745194435,5.6162850431000875)
    (5.333978668303457,5.698128530439982)
```

Scala Solution

```
1  def parseInput(text:String): List[(Double, Double)] = {
2      val pat = """\d\.\d+""".r
3      def loop(text:List[String], sofar:List[(Double, Double)]): List[(Double,
   ↪   Double)] = {
4          text match {
5              case Nil => sofar
6              case x::xs => { val m = pat.findAllMatchIn(x).toList
7                              loop(xs, (m(0).toString.toDouble,
   ↪   m(1).toString.toDouble)::sofar)
8              }
9          }
10     }
11     loop(text.split("\n").toList, List())
12 }
13
14 def solution(text:String): List[(Double, Double)] = {
15     val points = parseInput(text)
16     def distance(p1:(Double, Double), p2:(Double, Double)): Double =
```

```
17          scala.math.sqrt((p1._1-p2._1)*(p1._1-p2._1) +
    ↪  (p1._2-p2._2)*(p1._2-p2._2))
18      points.combinations(2).map(x => (distance(x(0), x(1)),
    ↪  x)).toList.sortBy(_._1).head._2
19  }
```

2.2 Connect Four

Description

Connect Four is a two-player connection game in which the players first choose a color and then take turns dropping colored discs (like checkers) from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before your opponent.

A fun discourse on winning strategies at Connect Four is found [here] (http://www.pomakis.com/c4/expert_play.html).

In this challenge you'll be given a set of game moves and then be asked to figure out who won and when (there are more moves than needed).

For sake of consistency, this is how we'll organize the board, rows as numbers *1-6* descending and columns as letters *a-g*. This was chosen to make the first moves in row 1.

```
   a b c d e f g
6  . . . . . . .
5  . . . . . . .
4  . . . . . . .
3  . . . . . . .
2  . . . . . . .
1  . . . . . . .
```

Input Description

You'll be given a game with a list of moves. Moves will be given by *column only* (gotta make this challenging somehow). We'll call the players *X* and *O*, with *X* going first using columns designated with an uppercase letter and *O* going second and moves designated with the lowercase letter of the column they chose.

```
C  d
D  d
D  b
C  f
C  c
B  a
A  d
G  e
E  g
```

Output Description

Your program should output the player ID who won, what move they won, and what final position (column and row) won. Optionally list the four pieces they used to win.

```
X won at move 7 (with A2 B2 C2 D2)
```

Challenge Input

```
D  d
D  c
C  c
C  c
G  f
F  d
F  f
D  f
A  a
E  b
E  e
B  g
G  g
B  a
```

Challenge Output

```
X won at move 13 (with D3 E3 F3 G3)
```

2.3 Detecting Four Sided Figures

Description

I got this idea from the Mensa quiz[1], specifically question 17. It's a basic scanning challenge: can your program detect and count intersecting bounding boxes from an ASCII art input? A four-sided figure is an ASCII art rectangle. Note that it can overlap another one, as long as the four corners are fully connected.

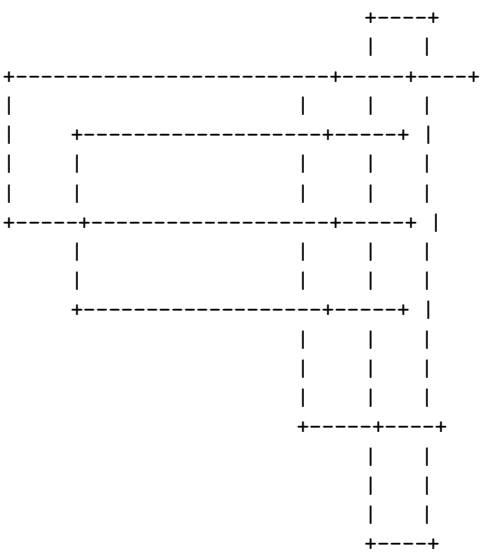
Formal Inputs & Outputs

Your program will be given an ASCII art chart showing boxes and lines. - and | characters indicate horizontal and vertical lines, respectively, while "+" characters show intersections.

Your program should emit an integer, *N*, of how many unique four sided figures it found. Rectangles and squares both count.

[1] https://www.mensa.org/workout/questions

Example Input

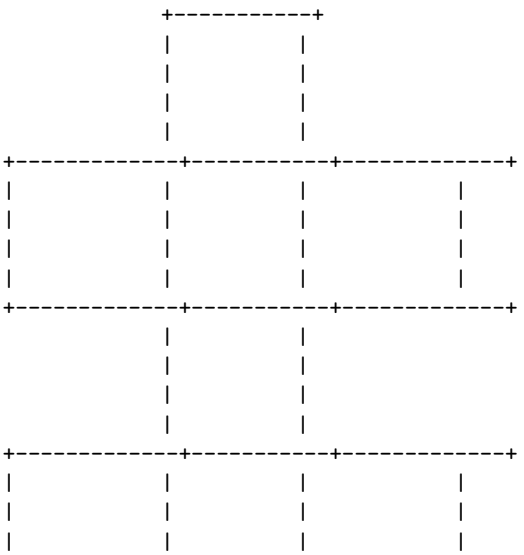


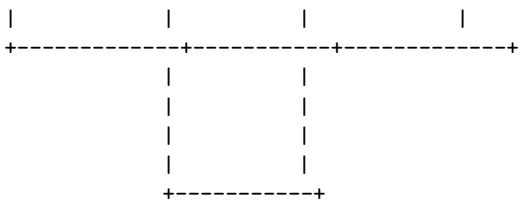
Example Output

For the above diagram your program should find 25 four sided figures.

Challenge Input

This one adds a bit to the complexity by throwing in some three sided figures. This should catch more naive implementations.





Challenge Output

For the challenge diagram your program should find 25 four sided figures.

2.4 Bioinformatics 2: DNA Restriction Enzymes

Description

Continuing with our bioinformatics theme today. If you like these sorts of problems, I encourage you to check out Project Rosalind[2] (their site seems back up).

Restriction enzymes are DNA-cutting enzymes found in bacteria (and harvested from them for use). Because they cut within the molecule, they are often called restriction endonucleases. In order to be able to sequence DNA, it is first necessary to cut it into smaller fragments. For precise molecular biology work, what is needed is a way to cleave the DNA molecule at a few specifically-located sites so that a small set of homogeneous fragments are produced. The tools for this are the restriction endonucleases. The rarer the site it recognizes, the smaller the number of pieces produced by a given restriction endonuclease.

For more information on how these enzymes work, including a great visualization of how they cut, have a look here[3].

These enzymes can cleave the DNA at a site that leaves both strands the same length. This is called a "blunt" end because of this and can be visualized like this:

```
5'-GG +CC-3'
3'-CC  GG-5'
```

Other DNA restriction enzymes cleave the ends at different lengths, although it's symmetrical about the central axis. These are called "sticky" ends, and here's a simple visualization of one of those cuts:

```
5'-ATCTGACT    + GATGCGTATGCT-3'
3'-TAGACTGACTACG      CATACGA-5'
```

In both cases the two strands are cut at a point of symmetry (the upper and lower strands are symmetrical if rotated).

Today your challenge is to write a program that can recognize the locations where various enzymes will cut DNA.

[2] http://rosalind.info/

[3] http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/R/RestrictionEnzymes.html

Input

You'll be given a list of DNA restriction enzymes and their recognition site and where the cut occurs. The input will be structured as enzyme name, if the enzyme makes a "sticky" or "blunt" end cut, the DNA recognition sequence and the position of the cut marked with a caret ("^"). For the sticky ends, you should assume the mirror image of the complementary strand gets the same cut, leaving one of the strands to overhang (hence it's "sticky").

```
BamHI sticky G^GATCC
HaeIII blunt GG^CC
HindIII sticky A^AGCTT
```

Then you'll be given a DNA sequence and be asked to cut it with the listed enzymes. For sake of convenience, the DNA sequence is broken into blocks of 10 bases at a time and in lengths of 6 blocks per row. You should merge these together and drop the first column of digits.

This sequence was taken from the genome of *Enterobacteria phage phiX174 sensu lato*[4] and modified for this challenge.

```
  1 gagttttatc gcttccatga cgcagaagtt aacactttcg gatatttctg atgagtcgaa
 61 aaattatctt gataaagcag gaattactac tgcttgttta cgaattaaat cgaagtggac
121 tgctggcgga aaatgagaaa attcgaccta tccttgcgca gctcgagaag ctcttacttt
181 gcgacctttc gccatcaact aacgattctg tcaaaaactg acgcgttgga tgaggagaag
241 tggcttaata tgcttggcac gttcgtcaag gactggttta gatatgagtc acattttgtt
301 catggtagag attctcttgt tgacatttta aaagagcgtg gattactatc tgagtccgat
361 gctgttcaac cactaatagg taagaaatca tgagtcaagt tactgaacaa tccgtacgtt
421 tccagaccgc tttggcctct attaagctta ttcaggcttc tgccgttttg gatttaaccg
481 aagatgattt cgattttctg acgagtaaca aagtttggat ccctactgac cgctctcgtg
541 ctcgtcgctg cgttgaggct tgcgtttatg gtacgctgga ctttgtggga taccctcgct
601 ttcctgctcc tgttgagttt attgctgccg tcaaagctta ttatgttcat cccgtcaaca
661 ttcaaacggc ctgtctcatc atggaaggcg ctgaatttac ggaaaacatt attaatggcg
721 tcgagcgtcc ggttaaagcc gctgaattgt tcgcgtttac cttgcgtgta cgcgcaggaa
781 acactgacgt tcttactgac gcagaagaaa acgtgcgtca aaaattacgt gcggaaggag
841 tgatgtaatg tctaaaggta aaaaacgttc tggcgctcgc cctggtcgtc cgcagccgtt
```

Output

Your program should emit the name of the enzyme, the cut positions for that enzyme, and the contextualized cut. For the above the solution would be:

```
BamHI 517 agttt[g gatcc]ctactg
HaeIII 435 gcttt[gg cc]tctattaa
HaeIII 669 caaac[gg cc]tgtctcat
HindIII 444 ctatt[a agctt]attcag
HindIII 634 cgtca[a agctt]attatg
```

[4] http://www.genome.jp/dbget-bin/www_bget?refseq+NC_001422

Bonus

Write some code that identifies any and all symmetrical points along the DNA sequence where an enzyme (not just the three listed) could cut. These should be even-length palindromes between 4 and 10 bases long.

Scala Solution

```
object Intermediate207 {
  def main(argc:Int, argv:Array[String]) = {
    val gene = """  1 gagttttatc gcttccatga cgcagaagtt aacactttcg gatatttctg
 ↪  atgagtcgaa
     61 aaattatctt gataaagcag gaattactac tgcttgttta cgaattaaat cgaagtggac
    121 tgctggcgga aaatgagaaa attcgaccta tccttgcgca gctcgagaag ctcttacttt
    181 gcgacctttc gccatcaact aacgattctg tcaaaaactg acgcgttgga tgaggagaag
    241 tggcttaata tgcttggcac gttcgtcaag gactggttta gatatgagtc acattttgtt
    301 catggtagag attctcttgt tgacatttta aaagagcgtg gattactatc tgagtccgat
    361 gctgttcaac cactaatagg taagaaatca tgagtcaagt tactgaacaa tccgtacgtt
    421 tccagaccgc tttggcctct attaagctta ttcaggcttc tgccgttttg gatttaaccg
    481 aagatgattt cgattttctg acgagtaaca aagtttggat ccctactgac cgctctcgtg
    541 ctcgtcgctg cgttgaggct tgcgtttatg gtacgctgga ctttgtggga taccctcgct
    601 ttcctgctcc tgttgagttt attgctgccg tcaaagctta ttatgttcat cccgtcaaca
    661 ttcaaacggc ctgtctcatc atggaaggcg ctgaatttac ggaaaacatt attaatggcg
    721 tcgagcgtcc ggttaaagcc gctgaattgt tcgcgtttac cttgcgtgta cgcgcaggaa
    781 acactgacgt tcttactgac gcagaagaaa acgtgcgtca aaaattacgt gcggaaggag
    841 tgatgtaatg tctaaaggta aaaaacgttc tggcgctcgc cctggtcgtc
 ↪  cgcagccgtt""".replaceAll(" ", "").replaceAll("[0-9]",
 ↪  "").replaceAll("\n", "")

    val enzymes = List(("BamHI", "ggatcc", 1),
                       ("HaeIII", "ggcc", 2),
                       ("HindIII", "aagctt", 1))

    for (e <- enzymes) {
      val (name, pat, pos) = e
      def loop(off:Int, name:String, pat:String, pos:Int): String = {
        gene.indexOf(pat, off) match {
          case -1 => ""
          case _  => val mark = "[" + pat.substring(0,pos) + " " +
 ↪  pat.substring(pos, pat.length) + "]"
                     val seq = gene.substring(gene.indexOf(pat, off)-5,
 ↪  gene.indexOf(pat, off)+12).replace(pat, mark)
                     val offset = gene.indexOf(pat, off)+pos
                     println(name + " " + offset + " " + seq)
                     loop(offset+2, name, pat, pos)
        }
      }
```

```
35          loop(0, name, pat, pos)
36        }
37      }
38    }
```

2.5 Maximizing Crop Irrigation

Description

You run a farm which isn't doing so well. Your crops that you planted aren't coming up, and your bills are bigger than your expected proceeds. So, you have to conserve water and focus instead on the plants that are growing. You have a center pivot watering system which has a rotating sprinkler around a central pivot, creating a circular watered area. For this challenge, you just have to decide where to locate it based on this year's crops.

Some notes:

- Because this is a simple grid, we're only dealing with integers in this challenge.
- For partially covered squares, round down: the sprinkler covers the crop if the distance from the sprinkler is less than or equal to the sprinklers radius.
- If you place the sprinkler on a square with a crop, you destroy the crop so handle accordingly (e.g. deduct 1 from the calculation).
- If in the event you find two or more placements that yield identical scores, pick any one of them (or even emit them all if you so choose), this is entirely possible.

Input Description

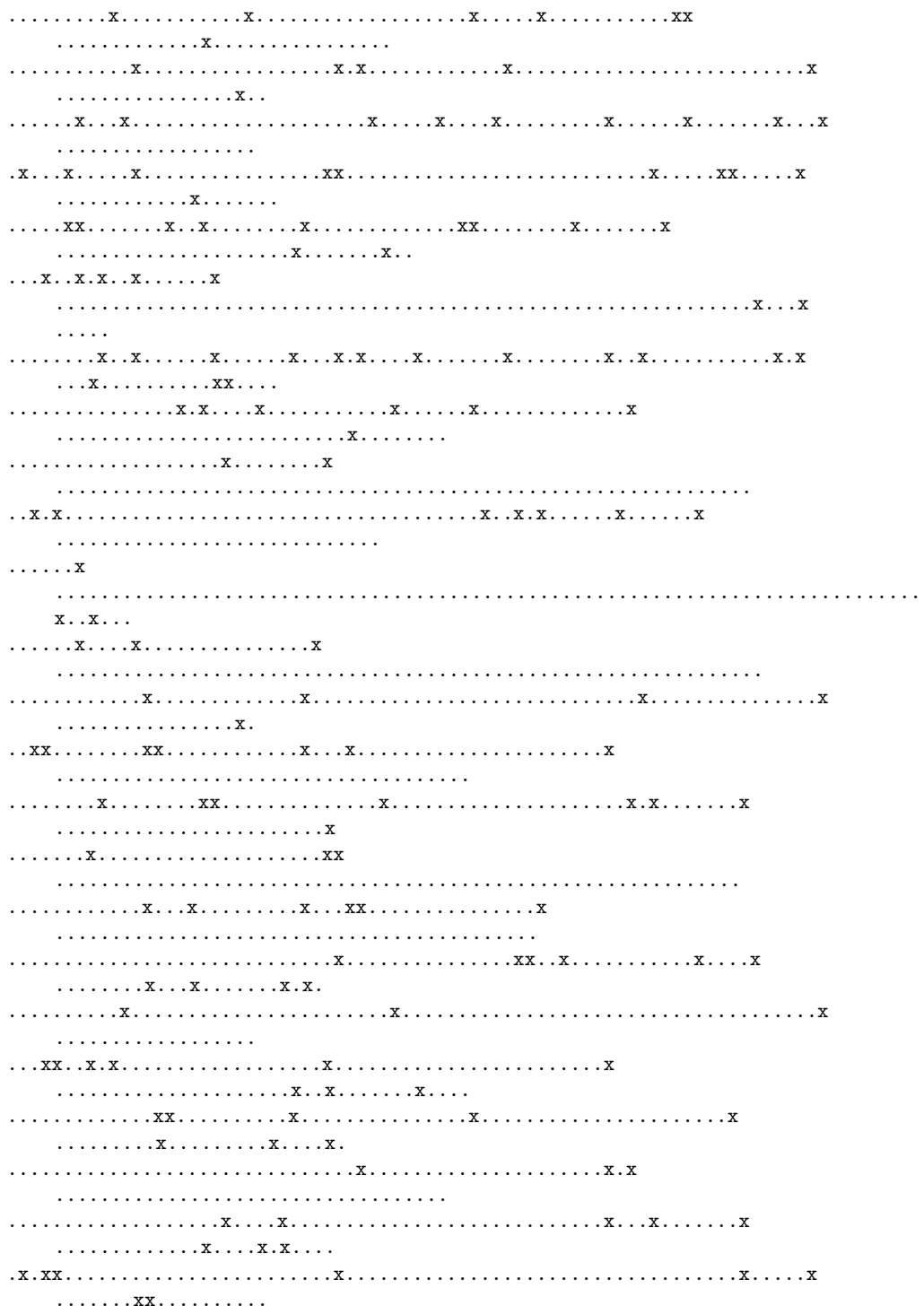
You'll be given three integers (h w r) which correspond to the number of rows (h) and columns (w) for the ASCII map (respectively) and then the radius (r) of the watering sprinkler. The ASCII map will have a "." for no crop planted and an "x" for a growing crop.

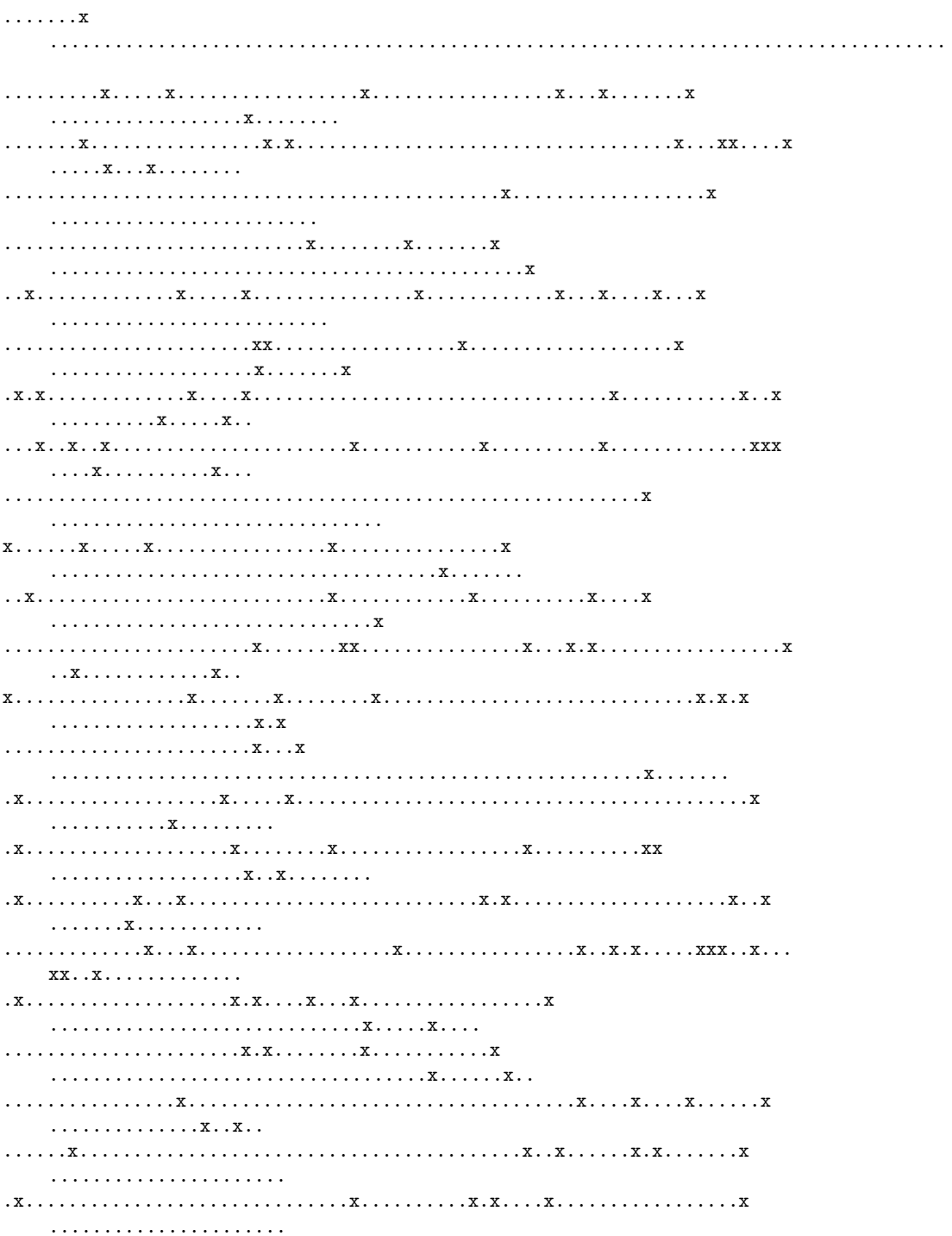
Output Description

You should emit the coordinates (0-indexed) of the row and column showing where to place the center of the sprinkler. Your coordinates should be integers.

Challenge Input

```
51 91 9
......x...x....x...........x...........x
    ................................x..............
```





```
x..x..........x..x.x...x.....................................x
    .............xx.......
..xx......x.......x.x................x
    ....................................x.............
```

Bonus

Emit the map with the circle your program calculated drawn.

Credit

This challenge was inspired by a question on IRC from user whatiswronghere.

2.6 Packing a Sentence in a Box

Description

You're moving, and you have a bunch of sentences to pack up. To accomplish this, you'll be using a small program you should write to pack these sentences efficiently into a box for shipping. Leave no unused space, you have a lot of sentences to pack and you don't want to waste precious shipping space.

For this challenge you're free to choose any legal dimensions of a rectangle, and you're free to start in any position you wish. Your program (and thus your output) should walk the grid to adjacent squares using only left, right, up, down (no diagonal moves allowed).

Input

You'll be given a sentence to pack into a box

```
EVERYWHERE IS WITHIN WALKING DISTANCE IF YOU HAVE THE TIME
```

Output

Your program should emit the starting position (column and row, 1-indexed) for the sentence, and then the box with the sentence packed into it. You can chose your own box dimensions. The above example is a 49 character sentence (minus spaces), so that's a 7x7 box. Here's one possible solution:

```
4 4
E      T      I      M      E      D      I
H      W      S      I      E      G      S
T      I      E      V      R      N      T
E      T      R      E      E      I      A
V      H      Y      W      H      K      N
A      I      N      W      A      L      C
H      U      O      Y      F      I      E
```

Challenge Input

```
IT IS RAINING CATS AND DOGS OUT THERE
```

Challenge Output

Here's one possible solution

```
1 1
I      T      I      N      I
E      I      A      G      N
R      S      R      C      A
E      G      O      D      T
H      S      O      D      S
T      T      U      N      A
```

2.7 Generating Polyominoes

Description

A polyomino is a collection of cells (equal-sized squares) which are connected, that is, each cell shares a border with another one. Think about tetris pieces, those would be tetrominoes - they each have four squares, and there are 5 unique combinations of their squares into unique shapes. Polyominoes are considered equivalent if they can be made to look identical if they are rotated or flipped. For additional background on polyominoes see this link[5].

Input Description

You will be given a single integer, this is the polyomino order to calculate and draw. Example:

```
4
```

Formal Output Description

Draw the complete set of unique polyominoes in ASCII art. Example output:

```
##
##

##
 ##

#
#
#
```

[5] http://home.adelphi.edu/~stemkoski/mathematrix/polys.html

```
#

#
#
##

#
##
#
```

Challenge Input

6

Challenge Input Solution

```
######

#
#####

 #
#####

  #
#####

##
 ####

##
####

# #
####

#  #
####

 ##
####

#
#
####

 #
```

```
79
80  ##
81  ##
82  ##
83
84    #
85  ###
86    #
87    #
88
89  ###
90    ##
91    #
92
93    #
94    ##
95  ###
96    #
97
98    #
99  ###
100 #
101 #
102
103   ##
104 ##
105 #
106 #
107
108 ###
109 # #
110 #
111
112 # #
113 ###
114 #
115
116 # #
117 ###
118   #
119
120   ##
121   #
122 ##
123 #
124
125 #
```

```
126   ##
127   ###
128
129    #
130   ###
131   ##
132
133     #
134   ###
135   ##
136
137     #
138    ##
139   ##
140   #
```

2.8 Finding Legal Reversi Moves

Description

The game of Reversi (or Othello) is a color flipping strategy game played between two players. It's played on an 8x8 uncheckered board. In each turn, the player must place a new chip on the game board. The chip must be placed in a currently empty square. The other requirement is that it be placed so that one or more of their opponent's chips lie between the empty square and another chip of the player's color. That is, the player placing a black chip must place it on an empty square with one or more white chips in a row - vertical, horizontal, or diagonal - between it and another black chip.

The object of the game is to have the majority of disks turned to display your color when the last playable empty square is filled.

Today's challenge is to review a game in progress and indicate legal moves for the next player.

Input Description

You'll be given a row with a single letter, `X` or `O`, denoting the player whose move it is. Then you'll be given the board as an 8x8 grid, with a dash `-` for an open square and an `X` or an `O` for a space occupied by that piece. Example:

```
X
--------
--------
--------
---OX---
---XO---
--------
--------
```

```
--------
```

Output Description

Your program should indicate the quantity of moves for that piece and then draw where they could be, indicated using a star *. Example

```
4 legal moves for X
--------
--------
---*----
--*OX---
---XO*--
----*---
--------
--------
```

Challenge Input

```
O
--------
--------
---O----
--XXOX--
---XOO--
----X---
--------
--------

X
--------
--------
---OX---
--XXXO--
--XOO---
---O----
--------
--------
```

Challenge Output

```
11 legal moves for O
--------
--------
--*O-**-
-*XXOX*-
-**XOO--
--**X---
```

```
---**---
--------

12 legal moves for X
--------
--***---
--*OX---
--XXXO*-
--XOO**-
--*O**--
---**---
--------
```

Note

For an interesting discussion of such algorithms, see the Wikipedia page on computer Othello[6]. An 8x8 board has nearly 10**28 legal moves in a game tree possible! One of the first computer Othello programs was published in 1977, written in FORTRAN.

2.9 Set Game Solver

Description

Set is a card game where each card is defined by a combination of four attributes: shape (diamond, oval, or squiggle), color (red, purple, green), number (one, two, or three elements), and shading (open, hatched, or filled). The object of the game is to find sets in the 12 cards drawn at a time that are distinct in every way or identical in just one way (e.g. all of the same color). The rules of Set are summarized by: If you can sort a group of three cards into "Two of ____ and one of _____," then it is not a set.

See the Wikipedia entry[7] for more background.

Input Description

A game will present 12 cards described with four characters for shape, color, number, and shading: (D)iamond, (O)val, (S)quiggle; (R)ed, (P)urple, (G)reen; (1), (2), or (3); and (O)pen, (H)atched, (F)illed.

Output Description

Your program should list all of the possible sets in the game of 12 cards in sets of triplets.

[6] https://en.wikipedia.org/wiki/Computer_Othello
[7] http://en.wikipedia.org/wiki/Set_(game)

Example Input

```
SP3F
DP3O
DR2F
SP3H
DG3O
SR1H
SG2O
SP1F
SP3O
OR3O
OR3H
OR2H
```

Example Output

```
SP3F SR1H SG2O
SP3F DG3O OR3H
SP3F SP3H SP3O
DR2F SR1H OR3O
```

Challenge Input

```
DP2H
DP1F
SR2F
SP1O
OG3F
SP3H
OR2O
SG3O
DG2H
DR2H
DR1O
DR3O
```

Challenge Output

```
DP1F SR2F OG3F
DP2H DG2H DR2H
DP1F DG2H DR3O
SR2F OR2O DR2H
SP1O OG3F DR2H
OG3F SP3H DR3O
```

Scala Solution

```
// solves the game "set" when given a list of cards (e.g. the board)
// cards look like "OR3O" for shape, color, count, and style

var sets = Source.
              fromFile("cards").
              getLines().
              map( x => x.toCharArray.toList ).
              toList.
              combinations(3).
              toList

def compare3(a:Char,b:Char,c:Char): Boolean = {
  (a == b && b == c && a == c) ||
  ( a != b && b != c && a != c)
}

def look(set:List[List[Char]]): Boolean = {
  for(pos <- Range(0,4)) {
    var a = set.apply(0).apply(pos)
    var b = set.apply(1).apply(pos)
    var c = set.apply(2).apply(pos)
    compare3(a,b,c) match {
      case false => return false
      case _ =>
    }
  }
  return true
}

println(sets.
  filter( x => look(x)).
  map(x => x.map(y => y.mkString))
)
```

2.10 Red Squiggles

Description

Many of us are familiar with real-time spell checkers in our text editors. Two of the more popular editors Microsoft Word or Google Docs will insert a red squiggly line under a word as it's typed incorrectly to indicate you have a problem. (Back in my day you had to run spell check *after* the fact, and that was an extra feature you paid for. Real time was just a dream.) The lookup in a dictionary is dynamic. At some point, the error occurs and the number of possible words that it could be goes to zero.

For example, take the word `foobar`. Up until `foo` it could be words like `foot`,

`fool`, `food`, etc. But once I type the `b` it's appearant that no words could possibly match, and Word throws a red squiggly line.

Your challenge today is to implement a real time spell checker and indicate where you would throw the red squiggle. For your dictionary use `/usr/share/dict/words` or the always useful `enable1.txt`.

Input Description

You'll be given words, one per line. Examples:

```
foobar
garbgae
```

Output Description

Your program should emit an indicator for where you would flag the word as mispelled. Examples:

```
foob<ar
garbg<ae
```

Here the $<$ indicates "This is the start of the mispelling". If the word is spelled correctly, indicate so.

Challenge Input

```
accomodate
acknowlegement
arguemint
comitmment
deductabel
depindant
existanse
forworde
herrass
inadvartent
judgemant
ocurrance
parogative
suparseed
```

Challenge Output

```
accomo<date
acknowleg<ement
arguem<int
comitm<ment
deducta<bel
depin<dant
```

```
existan<se
forworde<
herra<ss
inadva<rtent
judgem<ant
ocur<rance
parog<ative
supars<eed
```

Bonus

Include some suggested replacement words using any strategy you wish (edit distance, for example, or where you are in your data structure if you're using a trie).

Scala Solution

```
def spellcheck(word:String):String = {
    def loop(word:String, sofar:Int, matches:List[String]): String = {
        if ((word.length == sofar) && (matches.contains(word))) {return word + " is spelled correctly"}
        matches match {
            case Nil => word.slice(0, sofar-1) + "<" + word.slice(sofar-1, word.length)
            case _   => loop(word, sofar+1, matches.filter(_.startsWith(word.slice(0, sofar))))
        }
    }
    loop(word, 1, scala.io.Source.fromFile("/usr/share/dict/words").getLines.toList.filter(_.startsWith(word.slice(0, 1))))
}
```

CSharp Solution

```
using System;
using System.IO;

class RedSquiggles {
    public static void Main() {
        string[] words = File.ReadAllLines("/usr/share/dict/words");
        string[] input = File.ReadAllLines("red_squiggles.txt");
        HashSet<string> wordlist = new HashSet<string>();
        foreach (string w in words) {
            for (int i = 1; i <= w.Length; i++) {
                wordlist.Add(w.Substring(0, i));
            }
        }
```

```
14          foreach (string word in input) {
15              for (int i = 1; i < word.Length; i++) {
16                  if (!wordlist.Contains(word.Substring(0, i))) {
17                      Console.WriteLine(word.Substring(0, i) + "<" +
    ↪  word.Substring(i, word.Length-i));
18                      break;
19                  }
20              }
21          }
22      }
23  }
```

2.11 Simple Stream Cipher

Description

Stream ciphers like RC4 operate very simply: they have a strong psuedo-random number generator that takes a key and produces a sequence of psuedo-random bytes, which is then XORed against the plaintext to provide the cipher text. The strength of the cipher then depends on the strength of the generated stream of bytes - its randomness (or lack thereof) can lead to the text being recoverable.

Challenge Description

Your program should have the following components:

- A psuedo-random number generator which takes a key and produces a consistent stream of psuedo-random bytes. A very simple one to implement is the linear congruential generator (LCG)[8]
- An "encrypt" function (or method) that takes a key and a plaintext and returns a ciphertext.
- A "decrypt" function (or method) that takes a key and the ciphertext and returns the plaintext.

Python Solution

```
1  import sys
2
3  # def xor(b, s): return "".join(map(lambda x: chr(x^b), map(lambda x: ord(x),
   ↪  s)))
4  def xor(b, s): return map(lambda x: x^b, map(lambda x: ord(x), s))
5
6  M = sys.maxsize
```

[8] https://en.wikipedia.org/wiki/Linear_congruential_generator

```
M = 128

def lcg(m, a, c, x): return (a*x + c) % m

def enc(msg, seed):
    res = []
    for ch in msg:
        res.extend(xor(lcg(M, 1664525, 1013904223, seed), ch))
        seed = lcg(M, 1664525, 1013904223, seed)
    return res

def dec(msg, seed):
    res = []
    for ch in msg:
        res.append(lcg(M, 1664525, 1013904223, seed)^ch)
        seed = lcg(M, 1664525, 1013904223, seed)
    return ''.join(map(chr, res))
```

Scala Solution

```
def lcg(m:Int, a:Int, c:Int, x:Int)=  (a*x + c) % m

def enc(s:String, key:Int): List[Int] =
    (0 to s.length).toList.foldLeft[List[Int]](List()){(acc, x) => if
    ↪  (acc.isEmpty) {List(lcg(128,664525, 1013904223,key))} else
    ↪  {lcg(128,664525, 1013904223,acc.head)::acc}}.zip(s.toCharArray).map(x
    ↪  => x._1^x._2)

def dec(msg:List[Int], key:Int): String =
    (0 to msg.length).toList.foldLeft[List[Int]](List()){(acc, x) => if
    ↪  (acc.isEmpty) {List(lcg(128,664525, 1013904223,key))} else
    ↪  {lcg(128,664525, 1013904223,acc.head)::acc}}.zip(msg).map(x =>
    ↪  x._1^x._2).map(_.toChar).mkString
```

2.12 Use a Web Service to Find Bitcoin Prices

Desciption

Modern web services are the core of the net. One website can leverage 1 or more other sites for rich data and mashups. Some notable examples include the Google maps API which has been layered with crime data, bus schedule apps, and more.

For this challenge, you'll be asked to implement a call to a simple RESTful web API for Bitcoin pricing. This API was chosen because it's freely available and doesn't require any signup or an API key. Other APIs work in much the same way but often require API keys for use.

The Bitcoin API we're using is documented[9] Specifically we're interested in the `/v1/trades.csv` endpoint.

Your native code API (e.g. the code you write and run locally) should take the following parameters:

- The short name of the bitcoin market. Legitimate values are:

  bitfinex bitstamp btce itbit anxhk hitbtc kraken bitkonan bitbay rock cbx cotr vcx

- The short name of the currency you wish to see the price for Bitcoin in. Legitimate values are:

  KRW NMC IDR RON ARS AUD BGN BRL BTC CAD CHF CLP CNY CZK DKK EUR GAU GBP HKD HUF ILS INR JPY LTC MXN NOK NZD PEN PLN RUB SAR SEK SGD SLL THB UAH USD XRP ZAR

The API call you make to the bitcoincharts.com site will yield a plain text response of the most recent trades, formatted as CSV with the following fields: UNIX timestamp, price in that currency, and amount of the trade. For example:

```
1438015468,349.250000000000,0.001356620000
```

Your API should return the current value of Bitcoin according to that exchange in that currency. For example, your API might look like this (in F# notation to show types and args):

```
val getCurrentBitcoinPrice : exchange:string -> currency:string -> float
```

Which basically says take two string args to describe the exchange by name and the currency I want the price in and return the latest price as a floating point value. In the above example my code would return `349.25`.

Part of today's challenge is in understanding the API documentation, such as the format of the URL and what endpoint to contact.

Note

Many thanks to /u/adrian17 for finding this API for this challenge.

2.13 Word Squares Part 1

Description

A word square is a type of acrostic, a word puzzle. In a word square you are given a grid with letters arranged that spell valid English language words when you read from left to right or from top to bottom, with the requirement that the words you spell in each column and row of the same number are the same word. For example, the first row and the first column spell the same word, the second row and second

[9] `http://bitcoincharts.com/about/markets-api/`

column do, too, and so on. The challenge is that in arranging those letters that you spell valid words that meet those requirements.

One variant is where you're given an n*n grid and asked to place a set of letters inside to meet these rules. That's today's challenge: given the grid dimensions and a list of letters, can you produce a valid word square.

Input Description

You'll be given an integer telling you how many rows and columns (it's a square) to use and then n^2 letters to populate the grid with. Example:

```
4 eeeeddoonnnsssrv
```

Output Description

Your program should emit a valid word square with the letters placed to form valid English language words. Example:

```
rose
oven
send
ends
```

Challenge Input

```
4 aaceeeedmmoonnn
5 aaaeeeefhhmoonssrrrrttttw
5 aabbeeeeeeeehmosrrrruttvv
7 aaaaaaaaabbeeeeeeeddddddggmmlloooonnssssrrrruvvyyy
```

Challenge Output

```
moan
once
acme
need

feast
earth
armor
stone
threw

heart
ember
above
revue
trees
```

```
bravado
renamed
analogy
valuers
amoebas
degrade
odyssey
```

2.14 Anagram Maker

Description

Anagrams, where you take the letters from one or more words and rearrange them to spell something else, are a fun word game.

In this challenge you'll be asked to create anagrams from specific inputs. You should ignore capitalization as needed, and use only English language words. Note that because there are so many possibilities, there are no "right" answers so long as they're valid English language words and proper anagrams.

Example Input

First you'll be given an integer on a single line, this tells you how many lines to read. Then you'll be given a word (or words) on *N* lines to make anagrams for. Example:

```
1
Field of dreams
```

Example Output

Your program should emit the original word and one or more anagrams it developed. Example:

```
Field of dreams -> Dads Offer Lime
Field of dreams -> Deaf Fold Miser
```

Challenge Input

```
6
Desperate
Redditor
Dailyprogrammer
Sam likes to swim
The Morse Code
Help, someone stole my purse
```

2.15 Finding Ancestors

Description

Given a body of facts, can you answer a question about who is someone's grandparent?

Input Description

You'll be given a set of names, all unique. The parentage and lineage of people is shown; for instance, A+B=C means A and B had a child C. Multiple children will be separated by a comma. For Example:

```
1
Beth+Dave=Tom,Dick,Harry
```

Then you'll be given a person's name followed by one or more question marks or exclamation points. Each question mark means go back one level of ancestery, each exclamation mark means go *forward* one level of ancestery. For example:

```
2
Beth!
Harry?
```

Output Description

Your task is to find one or more ancestors (or descendants) of that person at the correct level. From our example:

```
Beth->Tom,Dick,Harry
Harry<-Beth,Dave
```

Challenge Input

```
8
Beth+Dave=Tom,Dick,Harry
Jan+John=Mary,Michael,Amy
Amy+Harry=Jim,John,Joe
Mary+Dick=Bill
Joan+Tom=Sam
Sam+Jane=Larry,Peter
Peter+Pauline=Francine
Francine+Frank=Thomas
4
Peter??
Sam!!
Thomas??
Francine??
Beth!!!
```

Challenge Output

```
Peter<-Joan,Tom,Tom,Jane
```

2.16 Encoding with the Beale Cipher

Description

You may recall the story of Thomas J. Beale[10], who (legend has it) worked with his cohorts to mine treasure over two years, then secretly bring it back to Virginia and hid it. He described the location of the treasure in a document encoded using the US Declaration of Independence as a key. Each number in the ciphertext corresponded to a word from the Declaration of Independence from which you take the first letter.

Today's challenge is to *encode* a message using this Beale cipher. There will be multiple solutions to any phrase, but your message should be able to decrypt your message using the solution from your Beale decoder.

Remember, Beale's ciphers discard anything not in the A-Z alphabet, so you'll have to drop whitespace and punctuation. Yes it makes decryption a bit harder.

Declaration of Independence

When in the course of human events it becomes necessary for one people to dissolve the political bands which have connected them with another and to assume among the powers of the earth the separate and equal station to which the laws of nature and of nature's god entitle them a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation we hold these truths to be self evident that all men are created equal that they are endowed by their creator with certain unalienable rights that among these are life liberty and the pursuit of happiness that to secure these rights governments are instituted among men deriving their just powers from the consent of the governed that whenever any form of government becomes destructive of these ends it is the right of the people to alter or to abolish it and to institute new government laying its foundation on such principles and organizing its powers in such form as to them shall seem most likely to effect their safety and happiness prudence indeed will dictate that governments long established should not be changed for light and transient causes and accordingly all experience hath shown that mankind are more disposed to suffer while evils are sufferable than to right themselves by abolishing the forms to which they are accustomed but when a long train of abuses and usurpations pursuing invariably the same object evinces a design to reduce them under absolute despotism it is their right it is their duty to throw off such government and to provide new guards for their future security such has been the patient sufferance of these colonies and such is now the necessity which constrains them to alter their former systems of government the history of the present king of great Britain is a history of repeated injuries and usurpations all having in direct object the establishment of an absolute tyranny over these states to prove this let facts be submitted to a

[10] https://en.wikipedia.org/wiki/Beale_ciphers

candid world he has refused his assent to laws the most wholesome and necessary for the public good he has forbidden his governors to pass laws of immediate and pressing importance unless suspended in their operation till his assent should be obtained and when so suspended he has utterly neglected to attend to them he has refused to pass other laws for the accommodation of large districts of people unless those people would relinquish the right of representation in the legislature a right inestimable to them and formidable to tyrants only he has called together legislative bodies at places unusual uncomfortable and distant from the depository of their public records for the sole purpose of fatiguing them into compliance with his measures he has dissolved representative houses repeatedly for opposing with manly firmness his invasions on the rights of the people he has refused for a long time after such dissolutions to cause others to be elected whereby the legislative powers incapable of annihilation have returned to the people at large for their exercise the state remaining in the meantime exposed to all the dangers of invasion from without and convulsions within he has endeavored to prevent the population of these states for that purpose obstructing the laws for naturalization of foreigners refusing to pass others to encourage their migration hither and raising the conditions of new appropriations of lands he has obstructed the administration of justice by refusing his assent to laws for establishing judiciary powers he has made judges dependent on his will alone for the tenure of their offices and the amount and payment of their salaries he has erected a multitude of new offices and sent hither swarms of officers to harass our people and eat out their substance he has kept among us in times of peace standing armies without the consent of our legislatures he has affected to render the military independent of and superior to the civil power he has combined with others to subject us to a jurisdiction foreign to our constitution and unacknowledged by our laws giving his assent to their acts of pretended legislation for quartering large bodies of armed troops among us for protecting them by a mock trial from punishment for any murders which they should commit on the inhabitants of these states for cutting off our trade with all parts of the world for imposing taxes on us without our consent for depriving us in many cases of the benefits of trial by jury for transporting us beyond seas to be tried for pretended offenses for abolishing the free system of English laws in a neighboring province establishing therein an arbitrary government and enlarging its boundaries so as to render it at once an example and fit instrument for introducing the same absolute rule into these colonies for taking away our charters abolishing our most valuable laws and altering fundamentally the forms of our governments for suspending our own legislature and declaring themselves invested with power to legislate for us in all cases whatsoever he has abdicated government here by declaring us out of his protection and waging war against us he has plundered our seas ravaged our coasts burnt our towns and destroyed the lives of our people he is at this time transporting large armies of foreign mercenaries to complete the works of death desolation and tyranny already begun with circumstances of cruelty and perfidy scarcely paralleled in the most barbarous ages and totally unworthy the head of a civilized nation he has constrained our fellow citizens taken captive on the high seas to bear arms against their country to become the executioners of their friends and brethren or to fall themselves by their hands he has excited domestic insurrections amongst us and has

endeavored to bring on the inhabitants of our frontiers the merciless Indian savages whose known rule of warfare is an undistinguished destruction of all ages sexes and conditions in every stage of these oppressions we have petitioned for redress in the most humble terms our repeated petitions have been answered only by repeated injury a prince whole character is thus marked by every act which may define a tyrant is unfit to be the ruler of a free people nor have we been wanting in attention to our British brethren we have warned them from time to time of attempts by their legislature to extend an unwarrantable jurisdiction over us we have reminded them of the circumstances of our emigration and settlement here we have appealed to their native justice and magnanimity and we have conjured them by the ties of our common kindred to disavow these usurpations which would inevitably interrupt our connections and correspondence they too have been deaf to the voice of justice and of consanguinity we must therefore acquiesce in the necessity which denounces our separation and hold them as we hold the rest of mankind enemies in war in peace friends we therefore the representatives of the united states of America in general congress assembled appealing to the supreme judge of the world for the rectitude of our intentions do in the name and by authority of the good people of these colonies solemnly publish and declare that these united colonies are and of right ought to be free and independent states that they are absolved from all allegiance to the British crown and that all political connection between them and the state of great Britain is and ought to be totally dissolved and that as free and independent states they have full power to levy war conclude peace contract alliances establish commerce and to do all other acts and things which independent states may of right do and for the support of this declaration with a firm reliance on the protection of divine providence we mutually pledge to each other our lives our fortunes and our sacred honor .

Input Description

You'll be given a sentence, one per line, that serve as your plaintext. Example:

```
Hello, world!
```

Output Description

Your program should emit a string of numbers that encode the plaintext in Beale's cipher. Example:

1268, 777, 881, 1319, 496, 718, 610, 987, 337, 138

That's one valid output for it. Another might be:

1052, 950, 652, 652, 1150, 415, 369, 1159, 423, 1117

And so on.

Challenge Input

```
I have deposited in the county of Bedford, about four miles from Buford's
    ,
```

```
in an excavation or vault, six feet below the surface of the ground, the
following articles
```

Challenge Output

641, 623, 823, 1135, 499, 527, 950, 1018, 402, 38, 3, 794, 777, 527, 1011, 552, 1297, 484, 1278, 68, 528, 640, 781, 214, 238, 198, 1201, 587, 1196, 475, 1187, 115, 1259, 878, 280, 1284, 374, 765, 679, 1284, 845, 1084, 562, 370, 489, 777, 1093, 1009, 1159, 770, 751, 580, 412, 1009, 109, 872, 249, 720, 724, 1199, 1097, 1051, 965, 495, 1048, 821, 1322, 237, 641, 496, 392, 128, 1293, 821, 1231, 845, 236, 842, 455, 118, 1049, 587, 777, 921, 700, 1278, 351, 266, 985, 1188, 1325, 193, 518, 319, 795, 404, 943, 649, 1034, 858, 1263, 459, 962, 777, 688, 399, 528, 1078, 928, 901, 538, 1052, 1001, 1167, 1047, 778, 1074, 109, 1273, 520, 272, 116, 903, 316, 1004, 460, 533, 489, 1034, 38

Python Solution

```python
def beale_cipher(plaintext, decl):
    def _startswith(ch, ideclrange):
        for i in ideclrange:
            if i[1].lower().startswith(ch.lower()):
                return i[0]
    idecl = map(lambda x: x[0], zip(enumerate(decl.split(), 1)))
    res = []
    for ch in plaintext.replace(" ", ""):
        i = random.randint(1, len(idecl))
        app = _startswith(ch, idecl[i:])
        if app:
            res.append(app)
        else:
            # maybe we went too far, start from beginning
            res.append(_startswith(ch, idecl))
    return res
```

2.17 Sturdy Brick Walls

Description

A brick wall is a rectangle made of horizontal 1-by-n bricks stacked into rows. A wall is sturdy if no two vertical edges of bricks appear in the same line. Here's an example of a sturdy wall:

```
[___][___]
_][___][__
[___][___]
```

And here's an example of a wall with a fault (it is not sturdy):

```
[______][______]
[__][____][__][]
[][______][____]
[____][______][]
```

In this challenge your task is to create walls at least three bricks high that are sturdy.

Input Description

You'll be given a list of integers that correspond to brick widths, one wall per line. Draw your bricks like this (e.g. the number tells you how many underscores to draw):

```
1: [_]
2: [__]
3: [___]
4: [____]
...
```

Output Description

Your program should emit a sturdy brick wall as ASCII art. If you have more than one possible solution, draw one or more.

Challenge Input

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

Challenge Output

```
[][__][__]
[__][__][]
[][__][__]
[__][__][]
[][__][__]

[][______________]
[__][____________]
[________________]
[____][__________]
[______][________]

[][__][____]
[__][____][]
[____][____]
```

2.18 Calkin-Wilf Tree

Description

In number theory, the Calkin–Wilf tree is a tree in which the vertices correspond 1-for-1 to the positive rational numbers. The tree is rooted at the number 1, and any rational number expressed in simplest terms as the fraction a/b has as its two children the numbers a/(a + b) and (a + b)/b. Every positive rational number appears exactly once in the tree.

The children of any vertex in the Calkin–Wilf tree may be computed by inverting the formula for the parents of a vertex. Each vertex a/b has one child whose value is less than 1, a/(a + b), because this is the only value less than 1 whose parent formula leads back to a/b. Similarly, each vertex a/b has one child whose value is greater than 1, (a + b)/b.

Input Description

You'll be given an integer on a line, *n*. This is the depth of the tree to which you should generate (with the root as depth 1). For example:

```
3
```

Output Description

Your program should emit the sequence of rational numbers at that depth. For example:

```
3 -> 1/3, 3/2, 2/3, 3/1
```

Challenge Input

```
5
```

Challenge Output

```
5 -> 1/9, 9/8, 8/15, 15/7, 7/20, 20/13, 13/19, 19/6, 6/23, 23/17, 17/28,
    28/11, 11/27, 27/16, 16/21, 21/5,
5/24, 24/19, 19/33, 33/14, 14/37, 37/23, 23/32, 32/9, 9/31, 31/22, 22/35,
     35/13, 13/30, 30/17, 17/21,
21/4, 4/23, 23/19, 19/34, 34/15, 15/41, 41/26, 26/37, 37/11, 11/40,
    40/29, 29/47, 47/18, 18/43, 43/25,
25/32, 32/7, 7/31, 31/24, 24/41, 41/17, 17/44, 44/27, 27/37, 37/10,
    10/33, 33/23, 23/36, 36/13, 13/29,
29/16, 16/19, 19/3, 3/20, 20/17, 17/31, 31/14, 14/39, 39/25, 25/36,
    36/11, 11/41, 41/30, 30/49, 49/19,
19/46, 46/27, 27/35, 35/8, 8/37, 37/29, 29/50, 50/21, 21/55, 55/34,
    34/47, 47/13, 13/44, 44/31, 31/49,
```

```
49/18, 18/41, 41/23, 23/28, 28/5, 5/27, 27/22, 22/39, 39/17, 17/46,
    46/29, 29/41, 41/12, 12/43, 43/31,
31/50, 50/19, 19/45, 45/26, 26/33, 33/7, 7/30, 30/23, 23/39, 39/16,
    16/41, 41/25, 25/34, 34/9, 9/29,
29/20, 20/31, 31/11, 11/24, 24/13, 13/15, 15/2, 2/15, 15/13, 13/24,
    24/11, 11/31, 31/20, 20/29, 29/9,
9/34, 34/25, 25/41, 41/16, 16/39, 39/23, 23/30, 30/7, 7/33, 33/26, 26/45,
     45/19, 19/50, 50/31, 31/43,
43/12, 12/41, 41/29, 29/46, 46/17, 17/39, 39/22, 22/27, 27/5, 5/28,
    28/23, 23/41, 41/18, 18/49, 49/31,
31/44, 44/13, 13/47, 47/34, 34/55, 55/21, 21/50, 50/29, 29/37, 37/8,
    8/35, 35/27, 27/46, 46/19, 19/49,
49/30, 30/41, 41/11, 11/36, 36/25, 25/39, 39/14, 14/31, 31/17, 17/20,
    20/3, 3/19, 19/16, 16/29, 29/13,
13/36, 36/23, 23/33, 33/10, 10/37, 37/27, 27/44, 44/17, 17/41, 41/24,
    24/31, 31/7, 7/32, 32/25, 25/43,
43/18, 18/47, 47/29, 29/40, 40/11, 11/37, 37/26, 26/41, 41/15, 15/34,
    34/19, 19/23, 23/4, 4/21, 21/17,
17/30, 30/13, 13/35, 35/22, 22/31, 31/9, 9/32, 32/23, 23/37, 37/14,
    14/33, 33/19, 19/24, 24/5, 5/21,
21/16, 16/27, 27/11, 11/28, 28/17, 17/23, 23/6, 6/19, 19/13, 13/20, 20/7,
     7/15, 15/8, 8/9, 9/1
```

https://en.wikipedia.org/wiki/Calkin%E2%80%93Wilf_tree

Scala Solution

```
1  class Rational(numerator: Int, denominator: Int) {
2
3    require(denominator != 0)
4
5    private val gcd = greatestCommonDivisor(numerator.abs,
6      denominator.abs)
7    val n = numerator / gcd
8    val d = denominator / gcd
9
10   def this(n: Int) = this(n, 1)
11
12   private def greatestCommonDivisor(a: Int, b: Int): Int =
13   if (b == 0) a else greatestCommonDivisor(b, a % b)
14
15   def + (that: Rational): Rational =
16   new Rational(n * that.d + d * that.n, d * that.d)
17
18   def - (that: Rational): Rational =
19   new Rational(n * that.d - d * that.n, d * that.d)
20
21   def * (that: Rational): Rational =
22   new Rational(n * that.n, d * that.d)
```

```
23
24    def / (that: Rational): Rational =
25    new Rational(n * that.d, d * that.n)
26
27    override def toString = n + "/" + d
28  }
29
30
31  def leaf(r:Rational): (Rational, Rational) = (new Rational(r.n, r.n+r.d),
    ↪  new Rational(r.n+r.d,r.d))
32  def leafToList(rr:(Rational,Rational)): List[Rational] = List(rr._1, rr._2)
33
34  def calkin_wilf(n:Int): List[Rational] = {
35      val root = new Rational(1,1)
36      def loop(r:Rational): List[Rational] = {
37          leafToList(leaf(r)).flatMap(x => leafToList(leaf(x)))
38      }
39      var res = loop(root)
40      for (_ <- (2 to n-1)) {
41          res = res.flatMap(loop)
42      }
43      res
44  }
```

2.19 Change Ringing

http://en.wikipedia.org/wiki/Change_ringing

Description

Change ringing is the art of ringing a set of tuned bells in a series of mathematical patterns called "changes". Change ringing differs from many other forms of campanology (such as carillon ringing) in that no attempt is made to produce a conventional melody. Developed in the 17th century, change ringing remains popular in England.

Permutations of the bells are done a pair at a time. Each row of bells is then rung once per row in that order before the next permutation and ringing. An example of this method is this set of rows, where 1 pair of neighbors is swapped for each permutation. This method yields the following pattern:

1,2,3,4,5,6 1,3,2,4,5,6 1,3,2,5,4,6 1,3,5,2,4,6 3,1,5,2,4,6 ...

For some people, the ultimate goal of this system is to ring all the permutations, to ring a tower's bells in every possible order without repeating — what is called an "extent" (or sometimes, formerly, a "full peal"). This pattern continues until the original ordering is returned. There are *N!* permutations.

Input Description

For this challenge you'll be given a single integer, *N*, which is the number of bells to organize.

Output Description

Your program should emit the complete list of steps to take in the permutation, the "full peal".

Challenge Input

6

Challenge Output

2.20 Constructing Cyclic Numbers

Description

A cyclic number[11] is an integer in which cyclic permutations of the digits are successive multiples of the number. The most widely known in base 10 is 142857:

```
142857 * 1 = 142857
142857 * 2 = 285714
142857 * 3 = 428571
142857 * 4 = 571428
142857 * 5 = 714285
142857 * 6 = 857142
```

You can watch the 714 loop through the number again and again, showing you the cycle.

Cyclic numbers can be constructed using the following steps:

1. Let b be the number base (e.g. 10 for decimal)
2. Let p be a prime that does not divide b.
3. Let $t = 0$.
4. Let $r = 1$.
5. Let $n = 0$.
6. Loop over the following steps:
    1. Let $t = t + 1$
    2. Let $x = r * b$

[11] https://en.wikipedia.org/wiki/Cyclic_number

3. Let $d = \text{int}(x/p)$
4. Let $r = x \bmod p$
5. Let $n = n * b + d$
6. If r != 1 then repeat the loop.

7. if $t = p$ - 1 then n is a cyclic number.

Your challenge today is a bit different than a typical challenge. Instead of creating or hunting for an algorithm, your challenge is to *implement* the above algorithm and create some cyclic numbers.

Note

Implementing an algorithm from a description is a valuable skill to have, just as is designing an algorithm. That's why this is a programming challenge - not solving a puzzle, but applying your language knowledge to a problem.

Scala Solution

```
def makeCyclic(b:Int, p:Int): Int = {
    var t = 0
    var r = 1
    var n = 0
    def loop(n: Int, t:Int, r:Int, b:Int): Int = {
        val tt = t + 1
        val x = r * b
        val d = x/p
        var nn = n * b * d
        r match {
            case 1 => t
            case _ => loop(nn, t, r, b)
        }
    }
    t = loop(n, t, r, b)
    if (t == p - 1) {
        return t
    } else {
        return -1
    }
}
```

2.21 Cyclic Words

Description

A cyclic word is a word that can be cleaved somewhere in the middle of it, the two halves swapped and then rejoined to form another legitimate word. For example the

word `slaughter` is one with `laughters` - cleave after the `s` and move `laughter` from the end to the beginning.

For this challenge you can use the famous `enable1.txt` file or `/usr/share/dict/words`, as long as it's an English dictionary.

Input Description

You'll be given an integer on one line telling you how many words to read, then a list of *N* words words to read. Example:

```
2
slaughter
calculate
```

Output Description

Your program should emit if the words are cyclic in some fashion. Example:

```
slaughter CYCLIC
calculate NOT CYCLIC
```

Challenge Input

```
10
large
easel
steak
believe
spot
words
overt
respite
ranger
neurectomy
```

Challenge Output

```
large NOT CYCLIC
easel CYCLIC
steak CYCLIC
believe NOT CYCLIC
spot CYCLIC
words CYCLIC
overt NOT CYCLIC
respite NOT CYCLIC
ranger NOT CYCLIC
neurectomy NOT CYCLIC
```

2.22 Calculating De Bruijn sequences

Description

In combinatorial mathematics, a *k-ary* De Bruijn sequence *B(k, n)* of order *n*, named after the Dutch mathematician Nicolaas Govert de Bruijn, is a cyclic sequence of a given alphabet *A* with size *k* for which every possible subsequence of length *n* in *A* appears as a sequence of consecutive characters exactly once. At the terminus, you "wrap" the end of the sequence around to the beginning to get any remaining subsequences.

Each *B(k, n)* has length *k^n*.

A De Bruijn sequence *B(2, 3)* (with alphabet *0* and *1*) is therefore:

```
00010111
```

Similarly, *B("abcd", 2)* (with alphabet "a", "b", "c", and "d") is therefore:

```
aabacadbbcbdccdd
```

For those sequences of length, every trigram (for the former case) or bigram (for the latter case) is represented in the result.

De Bruijn sequences have various applications, including in PIN pad testing and rotor angle calculation.

Input Description

You'll be given two inputs *k* and *n*, the first is an integer or a a string of unique characters, the second is the length of the subsequences to ensure are encoded.

Output Description

Your program should emit a string that encodes the De Bruijn sequence.

Input

```
5 3
2 4
abcde 4
```

Output

The outputs expected for those (in order) are:

```
00010020030040110120130140210220230240310320330340410420430441112113114122123124132133134142143144222322423
```

```
0000100110101111
aaaabaaacaaadaaaeaabbaabcaabdaabeaacbaaccaacdaaceaadbaadcaaddaadeaaebaaecaaedaaeeababacabadabaeabbbabbcabbda
```

Notes

Have an idea for a challenge? Please share it on /r/dailyprogrammer_ideas.

2.23 Tallest Tower from a List of Digits

Description

Today we're building towers of digits. A digit can be on top of another one:

```
2
6
```

or can be supported by two other diagonally below it:

```
 9
5 8
```

The bottom one(s) have to support the weight the upper one supports (if there is any), plus the upper one's weight which is always 1. If there are two supporters, they split the upper one's total weight evenly (50%-50%).

The weight of every digit is 1 independent of it's value. If one digit supports two others it has to be able to support the sum of their corresponding weight. A digit can support at most its numerical value.

Example valid towers look like this:

```
7
5
2
3


  1
 5 4
5 9 5
```

Input Description

You'll be given a list of space-separated integers, one list per line. Example:

```
1 2 3 4 5
```

Output Description

Your program should emit the maximal hieght of the tower you calculated and draw it, as well. Example:

```
1 2 3 4 5 -> 5
1
2
3
```

```
4
5
```

Challenge Input

```
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9
0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6 6 6 6 7 7
    7 7 7 8 8 8 8 8 9 9 9 9 9
```

Challenge Output

```
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 -> 9
  1
  2
  2
  3
  4
  5
 3 3
4 4 4
5 5 5 5

0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 -> 12
 0
 1
 2
 3
 4
 5
 6
 7
4 5
6 7
8 8
9 9

0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6 6 6 6 6 7 7
    7 7 7 8 8 8 8 8 9 9 9 9 9 -> 18
    0
    1
    2
    3
    4
    5
    6
    7
    8
    9
```

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

2.24 Elggob - Make a Boggle Layout

Description

The game of Boggle is familiar to many - a set of 36 6-sided dice with letters on them, which then turns into a 6x6 layout of letters, and then you work on finding properly spelled words by moving from one die to another. We've done a Boggle challenge in the past. In Boggle rules you may move left, right, up, down, and diagonally, but you may not use the same die twice as you spell a word.

Now can you do it backwards? Given a list of target words, can you create a Boggle layout using any other letters you wish that could yield those words and more?

Input Description

First you'll be given an integer on a line telling you how many words to target. Then you'll be given a list of *N* target words, one per line. Example:

```
3
CATCHER
MOUSE
AIRY
```

Output Description

Your program should emit a possible Boggle layout that could yield those words and any other ones that happen to be possible. For example:

```
L  W  D  J  M  Q
L  A  H  E  R  J
K  C  I  E  S  O
N  A  T  R  U  E
P  C  Y  M  O  W
T  E  O  H  T  C
```

Notice that you can spell words like COW, TOW, TOE, TOURS, and more in the above, in addition to the 3 words we had to target.

Challenge Input

```
9
MID
RANA
GRANT
BOCCA
CILIA
WAIVE
OSSAL
SALMO
FICE
```

Credit

This challenge as inspired by a question[12] from /u/JRhapsodus.

2.25 Math Snake

Description

There's a math puzzle making the rounds of the net as of May 2015. According to the VN Express, it was set as a problem for third graders in the town of Bao Loc in the Vietnamese highlands. You need to fill in the gaps with the digits from 1 to 9 so that the equation makes sense, following the order of operations - multiply first, then division, addition and subtraction last.

Input Description

Rather than making you parse an ASCII snake, you can parse the input as a series of blanks and values. You'll be given oprands (=, +, −, ∗ and /) and numbers together with blanks (as underscores, or _) that you need to fill in.

Output Description

Your program should emit a solution (multiple solutions may exist) as a valid equation.

Challenge Input

```
_ + 13 * _ / _ + _ + 12 * _ - _ - 11 + _ * _ / _ - 10 = 66
```

Challenge Output

```
3 + 13 * 2 / 1 + 5 + 12 * 4 - 7 - 11 + 9 * 8 / 6 - 10 = 66
```

[12] https://www.reddit.com/r/compsci/comments/3zjt44/filling_a_grid_with_words_using_boggle_rules/

Notes

This puzzle comes from Alex Bellos at the Guardian's May 20, 2015 column http://www.theguardian.com/scier
adventures-in-numberland/2015/may/20/can-you-do-the-maths-puzzle-for-vietnamese-
eight-year-olds-that-has-stumped-parents-and-teachers?CMP=share_btn_tw

2.26 Friedman numbers

Description

A Friedman number is an integer, which in a given base, is the result of an expression using all its own digits in combination with any of the four basic arithmetic operators (+, -, *, /), additive inverses, parentheses, and exponentiation. For example, 347 is a Friedman number, since 347 = 7^3 + 4. Parentheses can be used in the expressions, but only to override the default operator precedence, for example, in 1024 = (4 - 2)10. Allowing parentheses without operators would result in trivial Friedman numbers such as 24 = (24). Leading zeros cannot be used, since that would also result in trivial Friedman numbers, such as 001729 = 1700 + 29.

Two special cases of Friedman numbers worth noting: A **nice or "orderly" Friedman number** is a Friedman number where the digits in the expression can be arranged to be in the same order as in the number itself. For example, we can arrange 127 = 2^7 - 1 as 127 = -1 + 2^7. **Vampire numbers** are a type of Friedman numbers where the only operation is a multiplication of two numbers with the same number of digits, for example 1260 = 21 * 60.

Input Description

You'll be given a number, which may or may not be a Friedman number, one per line. Examples:

```
127
2502
576
```

Output Description

You'll be asked to output the formula that results in the value according to Friedman number rules, or a statement that it is not:

```
127 = 2^7+1
2502 = 50^2+2
576 NOT VALID
```

Challenge Input

```
343
1285
8147
123456789
```

Challenge Output

```
343 = (3+4)^3
1285 = (1+2^8)+5
8147 NOT VALID
123456789 = ((86 + 2 * 7)5 - 91) / 34
```

Notes

The August 2000 Math Magic problem was around Friedman numbers, here are many of them decomposed. http://www2.stetson.edu/~efriedma/mathmagic/0800.html

2.27 Graph Radius and Diameter

Description

Graph theory has a relatively straightforward way to calculate the *size* of a graph, using a few definitions:

- The eccentricity *ecc(v)* of *v* in *G* is the greatest distance from *v* to any other node.
- The radius *rad(G)* of *G* is the value of the smallest eccentricity.
- The diameter *diam(G)* of *G* is the value of the greatest eccentricity.
- The center of *G* is the set of nodes *v* such that *ecc(v)=rad(G)*

So, given a graph, we can calculate its size.

Input Description

You'll be given a single integer on a line telling you how many lines to read, then a list of *n* lines telling you nodes of a *directed* graph as a pair of integers. Each integer pair is the source and destination of an edge. The node IDs will be stable. Example:

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

Output Description

Your program should emit the radius and diameter of the graph. Example:

```
Radius: 1
Diameter: 2
```

Challenge Input

```
146
10 2
28 2
2 10
2 4
2 29
2 15
23 24
23 29
15 29
15 14
15 34
7 4
7 24
14 2
14 7
14 29
14 11
14 9
14 15
34 15
34 14
34 29
34 24
34 11
34 33
34 20
29 23
29 7
29 2
29 18
29 27
29 4
29 13
29 24
29 11
29 20
29 9
29 34
29 14
```

```
29 15
18 27
18 13
18 11
18 29
27 18
27 4
27 24
4 2
4 27
4 13
4 35
4 24
4 20
4 29
13 18
13 16
13 30
13 20
13 29
13 4
13 2
24 4
24 30
24 5
24 19
24 21
24 20
24 11
24 29
24 7
11 18
11 24
11 30
11 33
11 20
11 34
11 14
20 29
20 11
20 4
20 24
20 13
20 33
20 21
20 26
20 22
20 34
22 34
22 11
```

```
22 20
9 29
9 20
21 9
21 20
21 19
21 6
33 24
33 35
33 20
33 34
33 14
33 11
35 33
35 4
35 30
35 16
35 19
35 12
35 26
30 13
30 19
30 35
30 11
30 24
16 36
16 19
16 35
16 13
36 16
31 16
31 19
5 19
19 30
19 16
19 5
19 35
19 33
19 24
12 33
12 35
12 3
12 26
26 21
26 35
6 21
6 19
1 6
8 3
8 6
```

```
3 8
3 6
3 12
3 35
33 29
29 33
14 33
29 21
```

Challenge Output

```
Radius: 3
Diameter: 5
```

2.28 Hitori Solver

Description

Hitori is a logic puzzle similar to Soduku in that you aim to get unique digits in any column and row, but different in that you're given an NxN matrix and you have to knock out the flawed numbers. The rules (from Wikipedia[13]): "Hitori is played with a grid of squares or cells, and each cell contains a number. The objective is to eliminate numbers by filling in the squares such that remaining cells do not contain numbers that appear more than once in either a given row or column. Filled-in cells cannot be horizontally or vertically adjacent, although they can be diagonally adjacent. The remaining un-filled cells must form a single component connected horizontally and vertically."

Challenge Input

You'll be given an integer showing the number of rows and columns, then the board as a series of numbers, your program must output a board with the correct values removed (e.g. replaced with an X)

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

[13] http://en.wikipedia.org/wiki/Hitori

2.29 IPv4 Subnet Calculator

Description

In IPv4 networking, classless inter-domain routing (CIDR) notation is used to specific network addresses that fall outside of the historic "class A", "class B" and "class C" desigation. Instead it's denoted in an IPv4 network address with a bit-lenegth mask. For example, the historic class A network of 10.0.0.0 is expressed as `10.0.0.0/8`, meaning only the first 8 bits of the network address are specified. CIDR notation allows you to specify networks outside of the classic octet boundaries.

For this challenge, you'll be given various IPv4 addresses and subnets and asked to remove ones already covered by a covering CIDR representation.

Input Description

You'll be given a single integer and then list of IPv4 host and addresses addresses, containing that many lines of input. Examples:

```
3
172.26.32.162/32
172.26.32.0/24
172.26.0.0/16
```

Output Description

Your program should emit the minimal covering set of the network addresses to remove ones already specified by the network addresses. From the above example only `172.26.0.0/16` would remain.

Challenge Input

```
12
192.168.0.0/16
172.24.96.17/32
172.50.137.225/32
202.139.219.192/32
172.24.68.0/24
192.183.125.71/32
201.45.111.138/32
192.168.59.211/32
192.168.26.13/32
172.24.0.0/17
172.24.5.1/32
172.24.68.37/32
```

Challenge Output

```
192.168.0.0/16
172.24.0.0/17
172.50.137.225/32
202.139.219.192/32
192.183.125.71/32
201.45.111.138/32
```

2.30 Isomorphic Words

Description

In graph theory, geometry and other mathematical subjects, an isomorph can be thought of as a relationship between two inputs that share a similar structure. We'll define the relationship between two words as being *isomorphic* if they reuse letters in the same pattern. As an example, the words `ESTATE` and `DUELED` both have the pattern `abcdca`:

```
ESTATE
DUELED
abcdca
```

Put another way, you can deduce a simple substitution cipher to convert from one word to another.

Input Description

Write code that takes two words and checks whether they are isomorphs. You'll be given two words per line and asked to determine if they're isomorphs of one another. Example:

```
ESTATE DUELED
FEED DEAD
```

Output Description

Your program should emit if the two words are isomorphic or not. Example:

```
ESTATE DUELED TRUE
FEED DEAD FALSE
```

Challenge Input

```
RAMBUNCTIOUSLY THERMODYNAMICS
DISCRIMINATIVE SIMPLIFICATION
BANANA SENSES
SNACK HEATER
```

Challenge Output

```
RAMBUNCTIOUSLY THERMODYNAMICS TRUE
DISCRIMINATIVE SIMPLIFICATION TRUE
BANANA SENSES FALSE
SNACK HEATER FALSE
```

Scala Solution

```
1  def isomorphic(w1:String, w2:String): Boolean = {
2      val m = w1.zip(w2).toMap
3      w2 == w1.map(x => m(x)).mkString
4  }
```

2.31 Laser in a Box

Description

Today's challenge is based on ASCII art, boxes, and lasers. Simply put, can you trace the path of a laser in a rectangular box, assuming it's coated with mirrors inside?

In this case, lasers will start in the upper left corner of the box and be made of only / or \. You should assume a perfect bounce off the side of the box.

Input Description

You'll be given two integers, *n* and *m*, representing the width and heigth of the box to build. Example:

5 3

Output Description

Your program should emit the box with the full path of the laser traced. Example:

```
#######
#\/\/\#
#/\/\/#
#\/\/\#
#######
```

Challenge Input

```
2 2
22 6
6 3
```

Challenge Output

```
####
#\ #
# \#
####

########################
#\  /\  /\  /\  /\  /\ #
# \/  \/  \/  \/  \/  \#
# /\  /\  /\  /\  /\  /#
#/  \/  \/  \/  \/  \/ #
#\  /\  /\  /\  /\  /\ #
# \/  \/  \/  \/  \/  \#
########################

########
#\    /#
# \  / #
#  \/  #
########
```

via http://codegolf.stackexchange.com/questions/48531/ascii-doodling-laser-in-a-box

2.32 Needles in Haystacks

Description

Most of us are familiar with the concept of subsequences - a short string contained in a larger string is just a sequence of characters. It has a role in things like Markov models, suffix trees, and biological sequence motif analysis.

For this challenge, your job is to implement a means to find the longest common substring between two strings. In this particular challenge data set I chose strings I built using a simple password generator based on English-language syllables.

Sample Input

You'll be given an integer on a line telling you how many pairs of words to read. Then you'll get that many lines of two words apiece. Example:

```
1
ARORHEREVESTNGEVEATALLULD HEANTOORISINOMETIOARNE
```

Sample Output

Your program should emit the longest common substring between the two strings. Example:

```
AR
```

Challenge Input

```
6
ARTHAOURENESERAHENNDTHER OMEWITHEVEROMENOTHERERETHIFOR
REENTHISHEERVERANDINGERATHE NTBUTTIOSHOSHONDMEULDTHIWIT
HESHOMEMEULDSEINVERNTVER WITSEEREVEEREWASWAESARERE
ARENEOMEINGERSTAROULHIITH ERTIOVERBUTOULVERVETHIWAER
ESYOUORTHAREFORHEOULLEVER HENNTMEVERHERBUTISHISHANT
VERNTMETERARHADHIFORERAENIONNGULDMEOMEHERTHTERERORNOTHAANLEONTHIHATSTANDERTHETHALEREHEOURTETIOAL

    ENTHADESANTEERWITOULHEVEEAHERTHARETIBUTNOTTHAAREANDARAREOMEHIENOULALNTINGTIOITHWITTIOVERTERH
```

Challenge Output

```
THER
THI
ES
OUL
EVER
EOMEH
```

Bonus

Can you find the longest common substring of any pair of words in this Limerick poem?

```
There was a young rustic named Mallory,
who drew but a very small salary.
  When he went to the show,
  his purse made him go
to a seat in the uppermost gallery.
```

Scala Solution

```
1  def LCS(a:String, b:String): String = {
2      def substrings(s:String): List[String] =
   ↪   s.inits.flatMap(_.tails.toList.init).toList
3
   ↪   substrings(a).toSet.intersect(substrings(b).toSet).toList.sortBy(_.length).last
4  }
```

2.33 Magic Squares

Description

A magic square is an arrangement of numbers (usually integers) in a square grid, where the numbers in each row, and in each column, and the numbers in the forward

and backward main diagonals, all add up to the same number. The requirement for a magic square is that the squares all have unique numbers, typically monotonically increasing (from 1 to *n^2*), arranged as described in the previous sentence. In math notation, a magic square is said to have *order n*, where *n* stands for the number of rows (and columns) it has. Magic squares were known to Chinese mathematicians as early as 650 BCE, and have since appeared in many cultures around the world.

Since then, algorithms have been found to construct magic squares and are now well known, with additional research done into strategies to solving larger ones.

Input Description

You will be given a single number *n* which is the order of the magic square to devise. Note that your magic square should start with 1 and go through *n^2* sequentially. Example:

```
3
```

Output Description

Your answer should emit the magic square as a simple grid of numbers separated by commas. Example:

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

Challenge Input

```
10
```

Challenge Input Solution

```
1,9,17,25,33,68,76,84,92,100
99,91,83,75,67,34,26,18,10,2
3,11,19,27,35,66,74,82,90,98
97,89,81,72,65,36,29,20,12,4
60,42,58,44,56,50,49,53,47,46
41,59,43,57,45,51,52,48,54,55
96,88,80,73,64,37,28,21,13,5
6,14,22,30,38,63,71,79,87,95
94,86,78,70,62,39,31,23,15,7
8,16,24,32,40,61,69,77,85,93
```

2.34 Generating Text with Markov Processes

Description

Text generation algorithms exist in a wide variety of formats, including "Mad Libs" and Markov processes. A Markov chain algorithm generates text by creating a statistical model of potential textual suffixes for a given prefix. That's a fancy way of saying "it basically determines the next most probable word given the training set." Markov chain programs typically do this by breaking the input text into a series of words, then by sliding along them in some fixed sized window, storing the first N-1 words as a prefix and then the Nth word as a member of a set to choose from randomly for the suffix. Then, given a prefix, pick randomly from the suffixes to make the next piece of the chain.

Take this example text:

```
Now is not the time for desert, now is the time for dinner
```

For a set of triples, yielding a bi-gram (2 word) prefix, we will generate the following prefixes and suffix:

```
Prefixes      Suffixes
--------      --------
Now, is       not
is, not       the
not, the      time
the, time     for
time, for     desert
for, desert   now
desert, now   is
now, is       not, the
is, the       time
the, time     for
time, for     desert, dinner
```

You'll see a couple of the prefixes have TWO suffixes, this is because they repeat but one with a different suffix and one with the same suffix. Repeating this over piles and piles of text will start to enable you to build statistically real but logically meaningless sentences. Take this example output from my program after running it over Star Trek plot summaries:

```
"attack." In fact, Yeoman Tamura's tricorder shows that Kirk has been
    killed after
beaming down to the bridge, Kirk reminisces about having time to beam
    down. Kirk wants
Spock to grab hold of him in a fist fight with Kirk and Spock try to
    escape, the collars
are activated, subjecting them to an entrance, which then opens. Scotty
    saves the day by
pretending to help Spock, and Mullhall voluntarily agree, and the others
    transported to
```

```
the one which is not at all obvious what to make diplomatic advances.
    Meanwhile Kirk is
able to get inside. McCoy and nerve pinches Chief at
```

Challenge

Your challenge today is to implement a Markov generator supporting a bi-gram prefix. It should be capable of ingesting a body of text for training and output a body of text generated from that.

Notes

Markov Chain Algorithm[14] from rose-hulman.edu

If you want to reproduce my Star Trek fun, I extracted the summaries from Eric Wasserman's site[15] and made them into a flat text file[16].

Python Solution

This is based on the example from Kernighan and Pike's The Practice of Programming, chapter 3.

```
from itertools import islice
import random
import sys

class Markov(object):
    def __init__(self, open_file):
        self.cache = {}
        self.open_file = open_file
        self.words = self.file_to_words()
        self.word_size = len(self.words)
        self.database()

    def file_to_words(self):
        self.open_file.seek(0)
        data = self.open_file.read()
        words = list(filter(None, data.split()))
        return words

    def window(self, seq, n=3):
        "Returns a sliding window (of width n) over data from the iterable"
        "   s -> (s0,s1,...s[n-1]), (s1,s2,...,sn), ...                   "
```

[14] http://www.rose-hulman.edu/Users/faculty/young/CS-Classes/csse220/200820/web/Programs/Markov/markov.html

[15] http://www.ericweisstein.com/fun/startrek/

[16] https://drive.google.com/file/d/0B3rX15hRO_71NEt0cl8tcWMxNnM/view?usp=sharing

```
        # from
↪  https://docs.python.org/release/2.3.5/lib/itertools-example.html
        it = iter(seq)
        result = tuple(islice(it, n))
        if len(result) == n:
            yield result
        for elem in it:
            result = result[1:] + (elem,)
            yield result

    def database(self):
        for w1, w2, w3 in self.window(self.words, n=3):
            key = (w1, w2)
            if key in self.cache:
                self.cache[key].append(w3)
            else:
                self.cache[key] = [w3]

    def generate_markov_text(self, size=25):
        seed = random.randint(0, self.word_size-3)
        w1, w2 = self.words[seed], self.words[seed+1]
        gen_words = []
        for i in range(size):
            gen_words.append(w1)
            try:
                w1, w2 = w2, random.choice(self.cache[(w1, w2)])
            except:
                pass
        gen_words.append(w2)
        return ' '.join(gen_words)

if __name__ == '__main__':
    for arg in sys.argv[1:]:
        m = Markov(open(sys.argv[1], 'r'))
        print(m.generate_markov_text(100))
        del(m)
```

2.35 Mathagrams

Description

A mathagram is a puzzle where you have to fill in the unknown digits to arrive at a given sum, with the values being added using every digit between 1 and 9 exactly once (yielding three 3-digit numbers). For this challenge, you'll write a program to solve such puzzles.

Input Description

You'll be given a simple addition equation and the sum to arrive at, with the letter *x* in place of the unknown digit for you to fill it. Example:

```
1xx + xxx = 468
```

Output Description

Emit the filled in equation with the *x* placeholders replaced by digits, making sure the addition adds up to the stated sum. Example:

```
193 + 275 = 468
```

Challenge Input

```
xx5 + xxx = 468
x9x + xxx = 468
xxx + x7x = 468
```

Challenge Output

```
175 + 293 = 468
195 + 273 = 468
295 + 173 = 468
```

2.36 Packing Stacks of Boxes

Description

You run a moving truck business, and you can pack the most in your truck when you have stacks of equal size - no slack space. So, you're an enterprising person, and you want to write some code to help you along.

Input Description

You'll be given two numbers per line. The first number is the number of stacks of boxes to yield. The second is a list of boxes, one integer per size, to pack.

Example:

```
2 343123321
```

That says "make two stacks of boxes with sizes 3, 4, 3, 1 etc".

Output Description

Your program should emit the stack of boxes as a series of integers, one stack per line. From the above example:

```
331
322
34
```

If you can't make equal sized stacks, your program should emit nothing.

Challenge Input

```
3 912743471352
3 42137586
9 2
4 064876318535318
```

Challenge Output

```
9124
7342
7135

426
138
75

(nothing)

0665
4733
8315
881
```

via http://codegolf.stackexchange.com/questions/48486/the-partition-problem-sorting-stacks-of-boxes

Scala Solution

```
1  import scala.annotation.tailrec
2
3  def elemsEqual(L:List[Int]): Boolean =
4      L.distinct.length == 1
5
6  def pack(n:Int, boxes:List[Int]): List[List[Int]] = {
7      val g = boxes.permutations
8      @tailrec def loop(g:Iterator[List[Int]]): List[List[Int]]= {
9          val s = g.next.grouped(n).toList
10         (elemsEqual(s.map(_.sum)) && (s.length == n)) match {
```

```
11              case true  => s
12              case false => loop(g)
13          }
14      }
15      try {loop(g)}
16      catch {
17          case _ => List(List[Int]())
18      }
19  }
20
21  val boxes = args(2).toCharArray.map(_.toString.toInt).toList
22  println(pack(args(1)toInt, boxes).map(_.mkString).mkString("\n"))
```

2.37 Picture Spot

Description

When we want to snap a picture at the zoo or a tourist attraction, we want to stand as close as we can. Sometimes there's a barrier in the way. Given a fence and the location of the thing you want to take a picture of, can you help calculate the best spot for a picture?

Example Input

On the first line, you'll be given a pair of coordinates that describe the fence as a straight line. Then on the second line you'll be given the coordinates for the thing you want to take a picture of (e.g. an animal at the zoo). Example:

```
(1,1),(1,5)
(3,3)
```

Example Output

Your program should emit the coordinates of where you should stand along the fence to be closest to the object you want to get a picture of. Example:

```
(1,3)
```

Challenge Input

```
(1,1),(10,10)
(7,3)

(3,9),(10,1)
(9,9)
```

Challenge Output

2.38 Finding Numbers with Manners

Description

In number theory, a polite number is a positive integer that can be written as the sum of two or more consecutive positive integers. 3 is a polite number because 1 + 2 = 3. 4 is not a polite number - it's an impolite number - because you cannot add any two consecutive numbers to yield a sum of 4. It turns out the impolite numbers are exactly the powers of two.

Furthermore, the *politeness* of a number is how many different arrangements of consecutive integers can sum to the given number. For instance, 9 has a politeness of 2 because:

```
9 = 2 + 3 + 4 = 4 + 5
```

For every x, the politeness of x equals the number of odd divisors of x that are greater than 1. An easy way of calculating the politeness of a positive number is that of decomposing the number into its prime factors, taking the powers of all prime factors greater than 2, adding 1 to all of them, multiplying the numbers thus obtained with each other and subtracting 1.

Input Description

You'll be given a row with a single integer *N* that tells you how many numbers to read. Then you'll be given *N* lines of integers to determine the politeness for. Example:

```
3
3
8
90
```

Output Description

Your program should emit the number and its politeness. Example:

```
3 -> 1
8 -> 0
90 -> 5
```

Challenge Input

```
10
42
87
3197
```

```
200
546
38
39
19
99
34
```

Challenge Output

```
42 -> 3
87 -> 3
3197 -> 3
200 -> 2
546 -> 7
38 -> 1
39 -> 3
19 -> 1
99 -> 5
34 -> 1
```

Bonus

Find the largest politeness of numbers below 10000.

Scala Solution

```
1  def politeness(n:Int): Int = {
2      def loop(n:Int, len:Int, windows:List[Int], sofar:List[Int]): List[Int]
   ↪   = {
3          if (len == n) { return sofar }
4          else {return loop(n, len+1, windows,
   ↪   windows.sliding(len).map(_.sum).filter(_==n).toList++sofar)}
5      }
6      if (n < 3) {0}
7      else {loop(n, 2, (1 to (n - 1)).toList, List()).length}
8  }
```

2.39 Polydivisble Numbers

Description

In mathematics a polydivisible number is a number with digits *abcde*... that has the following properties :-

- Its first digit *a* is not 0.
- The number formed by its first two digits *ab* is a multiple of 2.

- The number formed by its first three digits *abc* is a multiple of 3.
- The number formed by its first four digits *abcd* is a multiple of 4.
- etc.

Your challenge today is to write a program that can generate one or more polydivisible numbers of a given length.

Input Description

You'll be given a single integer *N* on a line telling you how many digits long to generate a polydivisible number. Example:

```
3
```

Output Description

Your program should emit one or more polydivisible numbers for that length. Example:

```
3 -> 246,249
```

Challenge Input

```
4
5
8
```

Challenge Output

These are just some of the many polydivisible numbers. Your program may generate other valid ones.

```
4 -> 1020,1024
5 -> 10205
8 -> 10205448
```

2.40 Primes in Several Bases

Description

Take a number that's prime in base 10 and convert it to its representation in other bases. Now let's forget that all these base numbers are in a bases b other than 10 and let's ask how many of them keep being primes base 10.

For example: 379081 is a prime in base 10, 637001 in base 9 (and is prime in base 10), 1344311 in base 8 (and is prime in base 10), etc.

Input Description

You'll be given a series of integers, one set per line. Example:

```
10 9 8
```

Output Description

Your program should emit an integer that is prime in base 10 when converted to representations in those bases. Example:

```
379081 637001 1344311
```

Challenge Input

```
10 9 8 7 6 5 3 2
10 9 8 7 5 4 3 2
10 9 8 7 6 5 3 2
10 9 8 7 6 5 4 2
```

Challenge Output

```
59771671
146752831
764479423
1479830551
```

via http://www.primepuzzles.net/puzzles/puzz_024.htm

2.41 Punch Card Creator

Description

Punch (or punched) cards are an archaic form of recording instruction. Many people here may think of them from the early digital computing era, but they actually go back to fairground organs and textile mills in the 19th century! The format most of us are familiar with was originally patented by Hollerith, using stiff card stock. Over the years this format changed slightly and varied on this them, including a diagonal cut corner. For this challenge we'll focus on the tail end of punch cards with IBM, GE and UNIVAC type cards.

To use them, a program would be transcribed to the punch cards. Each column represented a single character, 80 columns to the card, 12 rows to the column. The zone rows can be used to have *two* punches per column. You can visualize it like this:

```
              ------------
             /
       /  12 / 0
Zone rows 11|  0
```

```
        \/  0|   0
        /   1|    0
       /    2|     0
      /     3|      0
 Numerc     4|       0
 rows       5|        0
      \     6|         0
       \    7|          0
        \   8|           0
         \  9|            0
             |_____________
```

Each card vendor would have an alphabet, an array of characters that are numerically represented by the punches. Here's an example of the DEC9 simple alphabet showing you the punch codes and the order in which they appear.

```
DEC9 &-0123456789ABCDEFGHIJKLMNOPQR/STUVWXYZ:#@'="[.<(+^!$*);\],%_>?
```

```
    ------------------------------------------------------------------
   /&-0123456789ABCDEFGHIJKLMNOPQR/STUVWXYZ:#@'="[.<(+^!$*);\],%_>?
12 / 0          OOOOOOOOO                       OOOOOO
11|  0                  OOOOOOOOO                     OOOOOO
 0|   0                         OOOOOOOOO                   OOOOOO
 1|    0       0       0       0
 2|     0       0       0       0       0     0     0     0
 3|      0       0       0       0       0     0     0     0
 4|       0       0       0       0       0     0     0     0
 5|        0       0       0       0       0     0     0     0
 6|         0       0       0       0       0     0     0     0
 7|          0       0       0       0       0     0     0     0
 8|           0       0       0       0 OOOOOOOOOOOOOOOOOOOOOOO
 9|            0       0       0       0
  |_________________________________________________________________
```

You can see the first 12 characters are represented by a single punch, then the next 9 have two punches (with one in the upper zone), then the next 9 use the next zone as that second punch, the fourth 9 use the next zone as the second punch, then we start on the lower zone for the next sets of 6 with the upper zone punched increasingly.

For some more information, including from where some of this info was taken, please see http://homepage.cs.uiowa.edu/~jones/cards/codes.html or Wikipedia http://en.wikipedia.org/wiki/
.

So, given an alphabet array you should be able to encode a message in a punch card, right? Let's go back to the punch card! For this challenge, assume the same encoding methods as above given the character array at the top, they'll only differ in order of characters.

Input Description

On the first line you'll be given two words - the punched card identifier, and the alphabet in linear order. Then you'll be given *M*, a single integer on a line, telling

you how many cshort messages to represent on that type of punch card.

Output Description

Your program should emit an ASCII art punchcard in the format above, with the diagonal notch and everything, and the message across the top.

Challenge Input

```
DEC9 &-0123456789ABCDEFGHIJKLMNOPQR/STUVWXYZ:#@'="[.<(+^!$*);\],%_>?
3
Hello, world!
This is Reddit's r/dailyprogrammer challenge.
WRITE (6,7) FORMAT(13H HELLO, WORLD) STOP END
```

Challenge Output

2.42 Singles

Description

Singles is a number grid game where you must remove some squares in order to stabilize the grid. Black out some of the squares, in such a way that:

- no number appears twice in any row or column
- no two black squares are adjacent
- the white squares form a single connected group (connections along diagonals do not count).

Input Description

You'll be given a single digit integer, *N*, on the first line. Then you'll be given an *N* x *N* grid of integers to play. Example:

```
5
3 2 2 1 3
2 3 2 5 4
4 5 2 2 1
2 1 5 3 3
5 3 1 3 5
```

Challenge Output

Your program should emit the grid with the numbers you knock out (using the above rules) indicated, perhaps with an *x* marking where they were. Example:

```
3 2 x 1 x
x 3 2 5 4
4 5 x 2 1
2 1 5 x 3
5 x 1 3 x
```

Challenge Input

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

Challenge Output

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

via http://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/singles.html

2.43 Slitherlink

Description

Slitherlink is a logic puzzle that was first published by Nikoli in Japan. The puzzle consists of a grid of dots, with some clue cells containing numbers. You connect horizontally or vertically adjacent dots to form a meandering path that forms a single loop or "Slitherlink." The loop must not have any branches and must not cross itself. The clue numbers indicate how many lines surround the cell. Empty cells may be surrounded by any number of lines (from 0 to 3).

Remember, the three rules for Slitherlink are:

1. Connect dots with vertical / horizontal line and make one loop,
2. Numbers are the hints to know how many lines can be drawn around it. There may be any number of lines around cells without number,
3. Lines cannot be crossed or branch off.

A simple example would be like this:

```
+ + + +    +-+-+-+
 3 2 3  =>  |3 2 3|
+ + + +    +-+-+-+
```

Using dashes - and pipes | draw the lines to connect the dots.

Input Description

You'll be given a square of numbers and nodes. The numbers indicate the square and the dots are represented by the + sign. Some squares may be blank, that's a wildcard. Example:

```
+ + + + +
 1 0   1
+ + + + +
 3 3
+ + + + +
     2 2
+ + + + +
 3   1 0
+ + + + +
```

Output Description

You should link the dots with a dash - or a pipe | as needed to satisfy the puzzle. Example:

```
+ + + + +
 1 0   1
+-+ +-+-+
|3|3|   |
+ +-+ +-+
|    2|2
+ +-+-+ +
|3|  1 0
+-+ + + +
```

Challenge Input

```
+ + + + + + + + +
 0 2 3 3 3 2 2 3
+ + + + + + + + +
 2           3
+ + + + + + + + +
 2 2 3       3
+ + + + + + + + +
     1
```

```
+ + + + + + + + +
          3
+ + + + + + + + +
 0        0 2 2
+ + + + + + + + +
 2           1
+ + + + + + + + +
 3 1 1 3 2 2 3 1
+ + + + + + + + +
```

Challenge Output

```
+ + +-+ +-+ +-+-+
 0 2|3|3|3|2|2 3|
+ +-+ +-+ + + +-+
 2|       | | |3
+-+ +-+ + +-+ +-+
|2 2|3|       3|
+ +-+ + +-+-+-+-+
| |  1| |
+-+ + + + +-+ +-+
      | | |3| | |
+ + +-+ +-+ +-+ +
 0  |      0 2 2|
+ +-+ +-+-+ +-+-+
 2|   |   | |  1
+-+ + +-+ + +-+ +
|3 1 1 3|2|2 3|1
+-+-+-+-+ +-+-+ +
```

Credit

This puzzle cam from the New York Times magazine on May 24, 2015. You can find more slitherlink puzzles on this website: http://krazydad.com/slitherlink/
http://www.kakuro-online.com/slitherlink/

2.44 Spinning Gears

Description

A popular game for the babies of nerdy parents are wooden or plastic gears. Kids put them on a peg board (or use magnets to attach them to the refrigerator) and watch them spin. It's a simple way for them to learn about actions and interactions. As a parent, it was sometimes fun to see how big I could make the gear networks. There's a logic puzzle game I play on the iPad, too, to solve gear puzzles.

Today's challenge is to read a set of gear specifications - their size and layout - and describe what initial gear to turn to make the target gear move in the right fashion - speed and direction. It's a bit of algebra and graph theory all thrown into one. You should assume frictionless gears.

Sample Input

You'll be given a line with a single integer on it (N) telling you how many gear specifications to read, followed by N lines. Gear specifications are given as a unique letter to designate it and then the radius of the gear. Then you'll be given another link with a single integer on it (M) telling you the layout, showing you what gears are touching by their letters, followed by M lines. Finally, you'll be asked on the last line to spin a gear (designated by a letter) in a direction (clockwise or counter-clockwise, CW or CCS) and how fast in RPMs, ending with the name of the gear to turn. Example of 3 gears forming an A-B and B-C line, asking you to spin A to make C turn at 30 RPM counter clockwise:

```
3
A 6
B 12
C 3
2
A B
B C
C CCW 30 A
```

Sample Output

Your program should emit the gear name, direction (CW or CCW) and speed to make the target gear spin as needed. From our example:

```
A CCW 15
```

Challenge Input

```
9
A 6
B 12
C 3
D 4
E 5
F 10
G 15
H 13
I 9
J 18
15
A B
B B
B C
C A
C A
C B
```

```
C B
C B
C C
C I
D B
E B
F B
G B
H B
B CW 75 I
```

2.45 Spoonerism

Description

A spoonerism is an error in speech or deliberate play on words in which corresponding consonants, vowels, or morphemes are switched (see metathesis) between two words in a phrase. It is named after the Reverend William Archibald Spooner (1844–1930), Warden of New College, Oxford, who was notoriously prone to this mistake. The term "Spoonerism" was well established by 1921.

Example spoonerisms include:

- "A blushing crow." ("crushing blow")
- "Resident Pagan" (President Reagan)
- "belly jeans" (jelly beans)

Generating spoonerisms is kind of tricky, but follows a simple rule: swap the leading consonants of the two words. Programatically this is easy, adjusting spelling for the pronunciation is tougher (and out of scope for this challenge).

Input Description

You'll be given a two word phrase, one per line, for which to generate a spoonerism. Words with which you need to swap letters will be capitalized to differentiate from stop words. Example:

```
Bold Clue
```

Output Description

Your program should emit the spoonerism form of the two word phrase. Spelling isn't an issue. Example:

```
Bold Clue -> Cold Blue
```

Challenge Input

```
Swap Letters
Lord of the Flies
Daily Bread
```

Challenge Output

```
Swap Letters -> Lap Swetters (which sounds like "Lop Sweaters")
Lord of the Flies -> Flord of the Lies (which is awfully close to "Ford
    of the Lies")
Daily Bread -> Braily Dead (or "Baily Dread")
```

2.46 English Word Syllbalizer

Description

A syllable is a unit of organization for a sequence of speech sounds. For example, the word water is composed of two syllables: wa and ter. A syllable is typically made up of a syllable nucleus (most often a vowel) with optional initial and final margins (typically, consonants).

For this challenge your program should take English language words and count the number of syllables in them.

If you have ever wished to automatically create poetry like a haiku, this is one of the components you'll need.

Input Description

You'll be given a series of English language words.

Output Description

Your program should emit the word and the number of syllables it counted in them.

Challenge Input

```
water
weigh
neighbor
neighborhood
bread
breadboard
cummerbund
cucumber
approximately
```

Challenge Output

```
water 2
weigh 1
neighbor 2
neighborhood 3
bread 1
breadboard 2
cummerbund 3
cucumber 3
approximately 5
```

2.47 Tile Shuffling

Description

Imagine an ecosystem where bugs that like to be solitary live on colored tiles, and different species inhabit different tiles. The bugs don't like to be together because when they're next to the same species, they explode. So, you have to keep them apart. You have been shipped a habitat of colored tiles and your task is to arrange them in a safe manner to keep the bugs wont explode.

For this challenge, we'll work with the rule that we don't want contiguous color tiles. We need to rearrange them - so we have to keep the tile counts per color the same - to find a stable arrangement where none of the adjacent ones are the same color. We're only concerned about the 4 cardinal directions - up, down, left, right - so diagonals are OK. The same color can appear elsewhere in the row or column as long as they're not touching.

This problem actually has relevance to topics like contagion spread, where you want as heterogenous a connected population as possible.

Input Description

You'll be given a single integer, N, on a line, and then a matrix of colors in the next N rows, with N tiles per row - you get an N by N square. Example:

```
3
RRR
GGG
BBB
```

Output Description

Your program should emit a solution within the constraints - tiles rearranged so that no two adjacent ones are the same color. Tiles will be in the alphabet `ROYGBIV`. Here's one possible solution to the above challenge.

```
RGB
BRG
GBR
```

Challenge Input

```
75
GYGOYRVYRBBIOGIYIRBRYBYRRRIVBOVIVIROVIOOOOOGROYVYYOIBIIOOGIYORRIIGRBYIVYOI
YOVBGVGRYRVYRVBORIBBVGYRIRVYVIVRGYOGIGYGROBGVIIOGGBGBIOBYYOOOROVIVVBVIIRBY
IOGVGYIIVOYGGBGVGBBGIGGVIRGIYGRIRIBRIGRBOYIVVYRGOVOGYRBIRIOBGYVIYBORVBOVIB
IGGGIGOIBORVVORIIRROOIVOVYOGYRRVRRVYOGBYVVIYIYGRRGRORRBBRBOIYIBBRYOBGGRRBR
OGIBYROIYGRYBYGBVRBBIVVIYVBYRIIBROBVRIYVYIVOVYRVRRBRIYORBIBBYRYOYGVIYRYROY
VOGBGGRBIGYBBIIRBIVIVGYYORIYRIROOIOYRBGGBIYGVVGIGOGVGOYBRRBROYBOOVRVVROBYI
OGBOYVROOROBVYGBYRVRRRRVIGGYGOGRVVVOGIBBIOVOOGIOIOIGBBVBBIRVRVGBROOOGBBRIY
IVGYOVOGOIYBGRORGIOYBBIGBIOYYBBGIYIBYGIBIRIOYIRBBGIOVGVBIVRYRGBGVVVIOYVGI
YBVOOOVIVROVIOOYRROYYYIIVYBGRBOORYIBBRYGRVRRBYVBOROIYIBOVGVVBROGGOYBOYRVOR
GRRGYYYYYYVVRYIBGGGGVBIVGGGYYRYRGIOVIGGYGGVVBYYYIGVRYBIRYOOORYIORGBRYBGGGR
OIGBOIVYVBGYVOGIIRBYVYRVOOVOVIBBRYVIIOGRRIOIOOOOBVVIOIRGIYIOOBRRRBIBGGRYRY
BRVBORBGBOGOVBOIGRGIOVBIYVBYYOIVRIVBRGRBRBVGRIIBVVVOVVRYIVYRYYYOOOOIROGYRV
IORRGYOOIIOOVIRGVIYVIGBOBGIBGRYVVBGOVORBYIVRBOIOIYBGIBGVIVGGOVBVRBOVYIGBIO
GBOIVIBVGVGIIOVRVBYBVGGBGGGIGVBRRIVYGOYVVOBGRVVRVBYIVVRBYOBBOOBRYBRROOVRGO
BGYVOVYGIGRGRGBVRRBIYOOYOGYVORORIGORYVOVGGVVGRRRIBRGYYRIBIOOGROIGVVVOVROBI
RRIBRGIYYOYBIGBBBVRRBOYGOOOVGYOYBVVYGGVOVVRVBBROVYIIRVVBIBGYORGBRVYOVVBRIR
IORGGGVIGVVBIBYGOIYROIOIGYYYOORYVROOGBVBRGOVORIRYBIBOOVRYRRYBYIVYVOVYGGGIG
VGRYIGGGVVGOBIYRIGBGIVIIIYIVYIIOIGVVBYOOIIIOYIIIGIYBVGRYVOYRVROVVIOGIOVOVY
ORYGRIIBVROVYRYGIOGYGIBYIVYBOIBGOBBRRGROBGIOGOVRYIBVBORIVYRYVIIIRBVGBGGBOO
VRIGIVIGGBBIBYYOYRVGROGBIBGGIRYRIVVRRYYRGYBIGBBGRYRGYOIYRIGBYRYBIBGOIBIVOY
RRVIRYBOOBOBRIOOOVROOGVIOBYRGGYGYVOVOOBRRIYYOIOIYBRBVGBRROYOYGGGRBOGOGYYIY
GORVRYGVOBVGGROORIYGBVOVGOOIIVIGYIGYVBBVVRVVYVIOIGBIGGYYBGRBRRRGROOOGRRVRG
GOBVOYVIRRRVGYVGBBGYRGYVIVYBGGBYOGGOORVIIYOYOGVVRVYRIGYRGYOYVIVBROVVGVIOVR
RYRYYBOGOROYYRRGBYIRGIYYBOYRYOGVOVBVIRVBVIVYRBBVVYVRYIOVYBIOOVVIYVGGIVBOGB
```

YGIIVOVOVBYGGGYIIRIBYVYGVRGBYRRYIBRYGIIVGVYVRBVBGGOIVBYYOVIOVOIYBVOOIIYBGB

BGIIGBIVRYYVGVIIRIIYRVRVYYYYBGIOIRRVBGRBVGOBYRGIIGRVBVVIBYYIBVGVGVIIBBYBVR

GBIGIBGBGGBROIOGGIYVGIBIYIRBYYVRYIVOGVVGGRRYGYVYVOYRYOORBGOYYVVVVVGOVVGBIO

YIOBYYIIVOIYYBIVOBBVYGOVGIGYVVVVYVBOBGGVIVGYYVGOYRYBOYVGVYVROBVYGRBVBVOIRV

OYYIGRIIGVVBRGYOROVRGOVRBGBOVVYRORIVGOIOVRYIRBVRBYVBRVIIOORRBYOOVVBVBOBBOO

GVBBOIIOIIBGBRVVBRORYYRBGBOVRRGRBBYGIOYVROYGYRBBBIBGOYRVOIYVRIVBRVBIRGROII

OIGBROOVVYBROVBOVGGOYGGVGBRYBGGRIRIOVVOOYVVGGYVBVGVVVGYRIVOROVOBRIVGBVVRVG

RYYVOORVGIYRGGGOIBOBBVBRIBIRVYVVYVBYOIOGRVBYIBBGIRIBRYGGGOORGGYBBRVOVRBGBG

VYOYBORBRYVRGYGBOVIORVOGOVGBIGYOOYGGGBROGBGOGIBIBVIBORBGIVVOBYYBBBGIYYVGBI

BOOGYVROYVGBBBIBBOVBGGIGBOOGGIOIIRBGOBOGRBVVYYIYOYYIVIIYOORBBGBOGVYOGIRYYR

GBBVBGVYBIYYRRGORVGGBBYIVGGIRRYRRIBVIIBGYYRGIOBIOOGBGGIRIIGRGOVIYBVYYGOIOI

GVVGGOYOYIOIRGYYYIIVIYIIYBOOVRYYYIGGGBGVROBOGRGRBIORVVYRGRIOGVRYBIVRIVIRVR

IIRBYGYBRYVBVRIOYVVYIBRGYGIGVBGRVVORRYGOIYOYYORBGRVGRYOYROIGGRRGGVYROYVVVO

YIBVRIRGIOYBOOYBGIIIBYOYRGGVBIORGRIOYOBIOYBVIIYYBIBGRIIVBOBGRVYRYRIIGGVIGR

RYBOYIIIYOGOIGOIVRIYYBOOVOGGIVRGYBBRGVVIVBROBYGIBYBGBGYYBOGRIGIYVGVBVYGOVO

YYBBRVVBIYRYIVYVYOBBIOIBGOYVOBRGROGBGGVIGIBGBYYYYBGIBORBOVVBVIVIIBBOOOIOIR

BIBRIIIGGGYRGGRIOBYIBYYBROYYBBIVBGVBVORYBIVBRIVVVRVOOOBOBBYGGRVIRRVIRIBOGG

OYORYRGBVRIYGOVGIVIYOROOGGVVROBRRBGOBIYRVIVOBYRIOVVIIBGRIYRGGIYBBYYYOBVYVB

RRVIYIBIIVOGYBVYRBRVYBVOBGGGYVVOOBORGVYYROGIGOROGRIVYOVBIBBYORIYGRIOROOIYB

GRYOGBGROGGBVGBVVIGBGVYRBVRYVIOVVYRYYRRRRBOOGOGOYGRYGOIYROYOOIGGYIBYYIRGGV

GVROYIOGVBOIORGBRBGOIIVBOYRGIOYOOBVIIBOYOIOVVVBOOYOYRYIBIOGIVRIYBOVIORVBOY

RBRVIIORYOBIOBRVYBVYGYVYOVGRYVGORGGGBBBIVYOGBBBGYYGRGVVYYIIBOIYVGBIGVVRBRB

ORIOBYGBYIGRRBYBBBROBROVOYBBYIGRRRROOYOVOIGVIGIGORIVGYBVBGGIGRYVVRORYBOIVY

BIBGOOOYIIIIOOIRBBGGYBGOGGROOYROIOYGOYIBRVYVYVRIOIVYVGROOBGOOBBGYOOBBIIIRI

BBOVOGRORGGIOGBGYBOYGRRYYRBBVGIGRIGGBBGGROGBRRIBBGBYIRVIYRYYVRVOIGIIIOIOGO

RIRIGBGOGGIBRBRGBGOOIIVOOIGROYYGOGGYIIRGRBRVVVVRGGROYOOOGIIRIRVYYIVBROYIRB

GORVYVBOIBYBVOIRIGIVOOORYIIBOYBVYOOVYRYYYBRYGVOGBIGIGGBIGGBIGIOYIOYYOIYOIO

VVBVVOGOVOYBIOIYRGYYGVVOGBVGRIBBYORIRGOYGIVYYRGROYYGVIROYIOBYIGOGGRYIYOGIB

VVIIYVGRIOBOGIORRBRYVBIBIOIRBBBYOIOYBIYYRIIYOYGYVRYIIYVOVOVGBYGGIVOVBOGYGY

GIBORRRIOIBYROIYRYRGRVROGIIRRBGVYRBRRRGYVVRBBBVOGYRYYYBROIIGRIRVOVYOOYGBGV

GVYYBVRBOBVOOGIYYRBBRGIVYGRGIYGYGVIYGOBIIRRYRGOGOOOOGBORYVIYGGYYOGVGYIYOOI

RRIRIVVORYBRRRRVVYOVIRVIYRRGRVVIBOBGRGYOOGGOYVGGGIRIVVBIGYOIIIGYGOIBIOVYYO

RORGBOVIGOVVIYGORRGGYOVIRRRYGYRGBYIGOOVBVBVYOVVIBIRGBRIOVGYVOYBOYVYBYVBYBY

BIVVBOOVOOOYIYYRYYOIRVIVVRIYOIVOOYYIYIGYVIYGGGIBORYORIGGGBRGBBIGGGGRBYVYYG

VBORGGOIOBYVGOOGRBGBYIGGOOGYYRBIYRIYIRVRRORIGGRGRIBIIOBGIIBOBGBBOVYOOYOYGV

YOBYIVIIGVORVRORIRORBIIOOBRGBORBIGBGRRGVGYIVGBBYVBIBGIIRYRYOOOYRGBYOVOBYI

OIYBROIRYGIYYOYYVOYOIIRIVGVVGGRBOBOBGYGOBYOYIRVGBIYYOOYOGYYGIBRIYROBBROIOY

GGVGGVIOOBVGVIGVVBBIBIBBRRYYIIYOGBYGGBRYGOYRIVYBVOVVVVIGVRGGIYOVIGBYVGOGRR

GYIBRYIOGBGYYYORIIYOVGYVBGBIOGBIIYVVRBVOOVYVBBOYGRBVRIORORRGYYVIVVOVIBRVVY

RYIOROORIOGRRGOROIRVBIORVGIIVGBYOIIGVGGBBOGYYRRBGBGGIIOGBIYOIOBYYBBYBVRRBB

VOROORVOVRRGYBBVRIGVOOYOBVOVOBVBIGIYIGYVGIYIGRYRORVRBBYIGVOGGIIOYGIGGGVVBO

IRVIOBRBGOGIIGBVBIOORIIYYVVYORRRGYBRROVVIOIYGIGIVRIBVBRVVYORGRBIBGGYBOIRVG

BOOBYGIROGROIIBRVGOGBYIGBVORYGIRYIOOVRORVYROBYVGBBIGBBGYRIBYGOOGIBIOOOYYRI

VRROIRRVOOGVIYIOVBIOYIGIGBOGOGVRGGVGYVBYGBBYBGIVGBVBORROVBRRVBIGBBRYBIVIYV

OOVBGRVIVYIYIIIBBIBBIBYIIBYGYOBRIVROROIVGYIIIYIYIYIBOYBRRRGIYOBBVRIYGGBVOO

BVYYOGVGBIVVBVIIIGOVOGYBGIBIVVIIGYVRVRIIYBBRRRGGVRIIIYYVOIORYYRRVIYBGVRIYO

YGRROYRIRGYBIRVORRYYVOVYYBOOIROVVVOGIIGOYGBGYROIRYGBIIRVIIRORGOGYVRVOOROOV

OOYIYGRYYIIBIOORRGBVIRRGYIOBBIOIYVIBYOOGYRYIIVBIBYYIOIBVRORBIOOBOBOVRRGBVR

YYBYBBOYRRIIRBOVVBBVYYBBGBOIRYYVOYYVOIROOBOOIBGRVBBGIGYRRIOIIROIIIGVIIGIOO

OVOYIIBVRBIRVYOOORRIYVGBVVGIRVRBOVYVOYVVVBGYRYBIYYGOBOIGGVIBOYBGVYVOYRYYRG

2.48 Trees in the Park

Description

You manage the city parks of beautiful Tree Town. You have a team who loves to plant trees, but have so much pride that they insist their trees be visible by everyone and that no crowding exists. After many contentious meetings you arrive at a compromise: every tree planed must be in its own park, its own row and its own column, and no two trees can be diagonally from each other, either. In doing this, every arborist gets to maximize their tree's enjoyment.

It's now left to you to plan where these trees will go.

Input Description

You'll be given a single integer *N* which tells you four things: how many rows to read, how many columns to read, how many parks you're working with, and how many trees to plant. You'll then be given the park layout as an ASCII art map with letters indicating the different park designations (e.g. "a", "b", "c" ...). An example map:

```
5
a a b c d
a a b b d
e a b d d
e e e d d
e e e e e
```

Output Description

Your program should emit the map with the trees planted in the correct spot. Indicate trees with an uppercase "T". For the above example map the solution is:

```
a a b T d
T a b b d
e a T d d
e e e d T
e T e e e
```

Challenge Input

```
5
a b b b b
a a b b b
a c d b b
a d d d e
a a a e e
```

Challenge Output

```
T b b b b
a a b T b
a T d b b
a d d d T
a a T e e
```

2.49 Two for One

Description

This game is simple - swap one letter in the input word with a new pair of two letters (e.g. p -> nn) to generate a valid resulting word (English words, only, please).

Formal Input Description

You'll be given a list of English words as input.

Formal Output Description

Your program should emit the valid English words that result from the substitution. Use the enable wordlist[17] if you lack a list of English words (e.g. /usr/share/dict/-words).

Sample Input

```
chapel
agenda
```

Sample Output

```
channel
addenda
```

Challenge Input

```
barber
cogent
staple
behave
axle
```

Challenge Input Solution (not visible by default)

[17] https://code.google.com/p/dotnetperls-controls/downloads/detail?name=enable1.txt

```
barbell
comment
steeple
behoove
apple
```

Note (optional)

This was from the NYTimes magazine on March 16. Puzzle credit goes to the always estimable Will Shortz.

Have a cool idea for a challenge? Submit it to /r/DailyProgrammer_Ideas!

Scala Solution

```
1  def candidates(word:String): List[String] = {
2      val len = word.length
3      scala.io.Source.
4                    fromFile("/usr/share/dict/words").
5                    getLines.
6                    filter(_.length == len+1).
7                    toList
8  }
9
10 def hammingDistance(a:String, b:String): Int =
11     a.toCharArray.zip(b.toCharArray).filter(x => (x._2 != x._1)).length
12
13 def twoforone(a:String, b:String): Boolean = {
14     for (i <- (0 to a.length-1)) {
15         if (a(i) != b(i)) {
16             println((a.slice(0,i) + b(i) + b(i) + a.slice(i+1, a.length)))
17             if (a.slice(0,i) + b(i) + b(i) + a.slice(i+1, a.length) == b) {
18                 return true
19             }
20         }
21     }
22     return false
23 }
24
25 def check(word:String): List[String] =
26     candidates(word).filter(x => hammingDistance(word, x) == 3).filter(x =>
   ↪  twoforone(word, x))
```

2.50 Calculting Ulam Numbers

Description

Ulam numbers are a sequence of integers that are composed of previous, distinct members of the sequence that have exactly one possible representation of two pre-

vious Ulam numbers added together. Starting with U_1 = 1 and U_2 = 2, the next Ulam number is 3 (1+2), then 4 (1+3). 5 is not a Ulam number because it can be represented in two ways: 1+4 and 2+3. Through 100 the first few terms are

1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, 62, 69, 72, 77, 82, 87, 97, 99

The sequence is named after the Polish physicist Stanislaw Ulam, who introduced it in 1964. There are infinitely many Ulam numbers. For, after the first n numbers in the sequence have already been determined, it is always possible to extend the sequence by one more element.

Different sequences can be generated if U_1 and U_2 differ but the same rules apply. For example, for (1, 3) the sequence starts 1, 3, 4, 5, 6, 8, 10, 12, 17, 21, . . .

Input Description

You'll be given 2 integers representing U_1 and U_2. Example:

```
1 2
```

Output Description

Your program should emit the first 10 or so terms of the sequence for the given U starting values Example:

```
1, 2, 3, 4, 6, 8, 11, 13, 16, 18
```

Challenge Input

```
2 5
1 5
3 7
```

Challenge Output

2.51 Unique County Names

Description

In the United States, states are divided first into counties (or in the case of Louisiana *parishes*, in Alaska *minicipalities*, *census areas* or *boroughs*, and in the US territory Puerto Rico *municipios*) before being further divided into smaller chunks of land and then finally cities, towns and villages. A great number of these counties have common names (e.g. for US presidents), but even more have local and historic names such as native tribes.

In this challenge, you're looking for those unique names. A couple of things to note:

- You can ignore the *type* of division it is, such as a county, city or parish. For comparisons, focus only on the **name** (e.g. Polk County would be the same as Polk Parish).
- Some of them have two word names (e.g. Prince William County), treat that as the name (Prince William).
- Yes, this is part data munging because of the complexity of the data source - human names.
- Yes, there will be many, many unique names. That's part of the challenge.
- Focus only on the 50 US states, so ignore Puerto Rico, DC, etc. The data set includes extra territories, so you'll have to analyze it first to discover which ones to omit.

The data comes from the US Census Bureau and can be downloaded from this URL: http://www2.census.gov/geo/docs/reference/codes/files/national_county.txt

Input Description

You'll find that the website formatted the data in CSV. An example line is:

```
AL,01,001,Autauga County,H1
```

We only care about the first and fourth columns, so in this case "AL" and "Autauga County". If you want to know more about the data, it's the 2010 FIPS Codes for Counties and County Equivalent Entities[18], see that site for a full explanation.

Output Description

Your program should emit the unique county (or county equivalent) name as pairs of abbreviated state name (e.g. "AL") and county name (e.g. "Polk County") for the 50 US states.

2.52 Unwrap Some Text

Description

Most of us are familiar with word wrap and justifying blocks of text. Our text editors do this for us - "wrap text to a width of 80 characters" and such. We've done challenges where we have made columns of text[19] and we've also played with decolumnizing text[20]. But this one's a bit different.

[18] https://www.census.gov/geo/reference/codes/cou.html

[19] https://www.reddit.com/r/dailyprogrammer/comments/2hssx6/29092014_challenge_182_easy_the_column_conundrum/

[20] https://www.reddit.com/r/dailyprogrammer/comments/3esrkm/20150727_challenge_225_easyintermediate/

Given a block of text, can your program correctly identify the start of the next paragraph? You're free to use any heuristic you want. This one differs from previous challenges in that there is no whitespace between paragraphs like you had before. You may want to think about the statistics of lines the close a paragraph.

Challenge Input

```
The ability to securely access (replicate and distribute) directory
information throughout the network is necessary for successful
deployment. LDAP's acceptance as an access protocol for directory
information is driving the need to provide an access control model
definition for LDAP directory content among servers within an
enterprise and the Internet. Currently LDAP does not define an
access control model, but is needed to ensure consistent secure
access across heterogeneous LDAP implementations. The requirements
for access control are critical to the successful deployment and
acceptance of LDAP in the market place.
This section is divided into several areas of requirements: general,
semantics/policy, usability, and nested groups (an unresolved issue).
The requirements are not in any priority order. Examples and
explanatory text is provided where deemed necessary. Usability is
perhaps the one set of requirements that is generally overlooked, but
must be addressed to provide a secure system. Usability is a security
issue, not just a nice design goal and requirement. If it is
impossible to set and manage a policy for a secure situation that a
human can understand, then what was set up will probably be non-
secure. We all need to think of usability as a functional security
requirement.
Copyright (C) The Internet Society (2000). All Rights Reserved.
This document and translations of it may be copied and furnished to
others, and derivative works that comment on or otherwise explain it
or assist in its implementation may be prepared, copied, published
and distributed, in whole or in part, without restriction of any
kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing
the copyright notice or references to the Internet Society or other
Internet organizations, except as needed for the purpose of
developing Internet standards in which case the procedures for
copyrights defined in the Internet Standards process must be
followed, or as required to translate it into languages other than
English.
The limited permissions granted above are perpetual and will not be
revoked by the Internet Society or its successors or assigns.
This document and the information contained herein is provided on an
"AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
```

```
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
```

Challenge Output

Your program should emit something like this:

The ability to securely access (replicate and distribute) directory information throughout the network is necessary for successful deployment. LDAP's acceptance as an access protocol for directory information is driving the need to provide an access control model definition for LDAP directory content among servers within an enterprise and the Internet. Currently LDAP does not define an access control model, but is needed to ensure consistent secure access across heterogeneous LDAP implementations. The requirements for access control are critical to the successful deployment and acceptance of LDAP in the market place.

This section is divided into several areas of requirements: general, semantics/policy, usability, and nested groups (an unresolved issue). The requirements are not in any priority order. Examples and explanatory text is provided where deemed necessary. Usability is perhaps the one set of requirements that is generally overlooked, but must be addressed to provide a secure system. Usability is a security issue, not just a nice design goal and requirement. If it is impossible to set and manage a policy for a secure situation that a human can understand, then what was set up will probably be non- secure. We all need to think of usability as a functional security requirement.

2.53 Vaki Puzzle Solver

Description

Vaki puzzles are solved by completing the grid so that there are two symbols in each cell (most commonly a letter and a number), each letter and each number appears once in each column and once in each row, and each pair appears only once in the grid.

Vaki puzzles are solved by completing the grid so that there is a letter and a number and in each cell, each letter and each number appears once in each column and once in each row, and each pair appears only once in the grid.

For a little bit more, see http://www.vakipuzzles.com/.

Input Description

On the first line you'll be given a single integer, *N*, which is the size of the grid to make (both in the horizontal and vertical axes, it's a square grid). Assume the numbers go from *1* to *N* and the letters from *A* to the *Nth* letter (e.g. C = 3, D = 4, etc). Subsequent lines will specify 3 values - two integers (row and column respectively) and one additional value, either a letter (e.g. *C_*), a number (e.g. *_2*), or a combination (e.g.* D1*). The "_" underscore means that there's a missing value.

Output Description

You should emit a simple grid showing all positions filled in for a valid puzzle with those constraints.

Sample Input

```
4
1 2 D3
4 4 C_
2 3 _2
```

Sample Output

```
C2      D3      A1      B4
D1      C4      B2      A3
A4      B1      C3      D2
B3      A2      D4      C1
```

Notes

The basic ideas of a Soduku solver apply here, although at a different depth (3 fold compared to Soduku).

2.54 Longest Word in a Box

Description

Can you find the shortest path from one corner of a box of letters to another? Here's the rub: the path you take through the box has to spell an English word.

Input Description

You'll be given an integer *N* which tells you how many rows and columns it has (it's a square), and then the box of letters. You can move up, down, left, or right but not diagonally.

Output Description

Starting from the top left corner and ending at the lower right, your program should emit the shortest English language word starting in one corner and ending in the opposite it can find by tracing a path through the box. Optionally show the path it took.

Challenge Input

```
11
b  u  h  l  d  r  t  w  f  v  b
f  c  j  k  c  k  r  g  k  m  r
x  k  m  i  n  s  t  f  q  q  w
d  x  f  n  s  q  e  r  g  h  j
v  t  w  w  q  t  z  f  u  t  g
b  r  w  d  l  r  s  a  l  e  e
n  w  t  q  q  e  x  e  l  h  w
h  r  q  w  g  w  v  t  e  r  q
r  r  w  r  t  w  g  y  v  e  n
w  f  t  h  h  d  h  u  s  j  e
q  d  y  y  j  n  p  j  w  v  s
```

Challenge Output

```
buckminsterfullerenes
```

Title

Word Wheel

Description

A word wheel is a puzzle where there's a series of letters on the outside of a circle and one in the middle. You then have to find as many words as possible that contain the letter in the middle and two or more of the letters on the outside.

For this challenge you'll be asked to solve word wheels.

Sample Input

You'll be given a word wheel as a small piece of ASCII art. Because wheels in ASCII are very tough, instead you'll be given three lines. The middle line will be the center of the wheel and the first and third lines will be the outside of the wheel. For your dictionary use /usr/share/dict/words or the classic enable1.txt file: http://code.google.com/p/dotnetperls-controls/downloads/detail?name=enable1.txt

```
e f t
  i
p   d
```

Sample Output

Your program should emit all words of the maximum length. For the above wheel we get:

```
tepid
fetid
```

Challenge Input

```
t a f e
   m
 b i s
```

Challenge Output

```
ambits
bemist
misate
miseat
samite
```

Credit

Many thanks to Ben Everard for the idea!

2.55 XOR Decoding

Description

One of the simplest ways to obfuscate text is to XOR a single byte value over the input data. This is not uncommon in malware, for example. If you know you have text strings in there, even better a *particular* text string, you can use that to test for the correct XOR value.

For example, take the following string in hex encoding (using ASCII values for characters):

```
0x6a 0x67 0x6e 0x6e 0x6d 0x22 0x75 0x6d 0x70 0x6e 0x66
```

Or in ASCII:

```
jgnnm"umpnf
```

This is XOR encoded with 0x02, so we can decode it the same way:

```
j XOR 0x2 = h
g XOR 0x2 = e
...
```

yielding "hello world".

Decoding Notes

You may want to think about how to decide if it's decoded. Perhaps letter frequencies, dictionary lookups, etc.

Input Description

You'll be given an input string that has been XOR encoded (with a single byte).

Output Description

Your program should emit the XOR byte value and the decoded string.

Challenge Input

```
Sont'wuh‘ufj'dfiihs'eb'uri'ni'CHT'Jhcb)
jvvr8--uuu,pgffkv,amo-p-fckn{rpmepcoogp
```

Challenge Output

```
0x07 This program cannot be run in DOS Mode.
0x02 http://www.reddit.com/r/dailyprogrammer
```

2.56 Zeckendorf Representations of Positive Integers

Description

Zeckendorf's theorem, named after Belgian mathematician Edouard Zeckendorf, is a theorem about the representation of integers as sums of Fibonacci numbers.

Zeckendorf's theorem states that every positive integer can be represented uniquely as the sum of one or more distinct Fibonacci numbers in such a way that the sum does not include any two consecutive Fibonacci numbers.

For example, the Zeckendorf representation of 100 is

```
100 = 89 + 8 + 3
```

There are other ways of representing 100 as the sum of Fibonacci numbers – for example

```
100 = 89 + 8 + 2 + 1
100 = 55 + 34 + 8 + 3
```

but these are not Zeckendorf representations because 1 and 2 are consecutive Fibonacci numbers, as are 34 and 55.

Your challenge today is to write a program that can decompose a positive integer into its Zeckendorf representation.

Sample Input

You'll be given a number *N* on the first line, telling you how many lines to read. You'll be given a list of *N* positive integers, one per line. Example:

```
3
4
100
30
```

Sample Output

Your program should emit the Zeckendorf representation for each of the numbers. Example:

```
4 = 3 + 1
100 = 89 + 8 + 3
30 = 21 + 8 + 1
```

Challenge Input

```
5
120
34
88
90
320
```

Challenge Output

2.57 Zombies Invaded my Village

Description

(Read this in the tone of "Manic Pixie Dream Girls", the spirit in which it's intended.)

Help! Zombies invaded my village! I need your help to isolate them!

Let me describe my village. We have houses connected with paths between them, and that's how we get from one house to another. Sometimes I have to visit one house before I can get to another because I don't have a path from my house to that one, but that's OK. For example, I love walking past grandma's house, she always makes me rice pudding (yum!).

But now zombies threaten our village. Our mayor, Francine, started to describe how to isolate the zombies by tearing up the paths and breaking the village into two parts. That means that some parts of the village will be turned into zombies, but thats better than the whole village being infected. (I hope grandma will be ok!) Before we could get to work, Francine was zombified, and now we're all left trying to figure out how to achieve her plan. She went to a really good university, and I am not that smart so I need your help!

What we want to do is to tell you about my village and have you tell me which paths to tear up and split our village into two parts so that we can defend it. We have to work fast so please tell me the minimum number of paths to tear up.

Help us, DailyProgrammer, you're my only hope!

Input Description

You'll be given a two integers on the first line, telling you how many lines to read (N) and how many distinct node IDs (M). Then you'll be given N lines of integers showing you the edge between the two nodes referenced by ID (1 -> M). Assume an edge weight of 1 and an undirected graph. Example:

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

Output Description

Your program should emit the smallest number of edges to remove from the graph to split it into two. Example:

```
1 2
1 3
```

Challenge Input

```
89 18 14
1 1
1 2
1 3
1 4
1 5
1 6
1 7
```

1 8
2 1
2 2
2 3
2 5
2 6
2 9
2 7
3 2
3 3
3 4
3 5
3 6
3 9
3 7
3 8
4 1
4 3
4 4
4 5
4 6
4 9
4 7
5 3
5 4
5 5
5 9
6 3
6 5
6 6
6 7
7 5
7 6
7 9
7 7
8 6
8 7
8 8
9 5
9 9
9 7
9 8
10 9
10 7
10 8
10 10
11 7
11 8
11 11
11 10

```
12 7
12 8
12 11
12 10
12 12
12 13
13 9
13 7
13 8
13 11
13 10
13 12
13 13
14 6
14 9
14 8
14 11
14 14
14 10
14 12
14 13
15 9
15 7
15 11
15 14
15 10
16 7
16 8
17 8
17 14
18 8
18 14
```

Chapter 3

Hard

Hard challenges are meant to really get you thinking like a computer scientist. Often I choose NP hard problems because they force you to consider time-space tradeoffs in a strive for efficiency. These not only require a confidence in a language but often the skills needed to be a good programmer - laying out the problem clearly and seeing a path to a solution, often with a pencil and paper, before you begin coding.

3.1 8 Puzzle

Description

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. You are permitted to slide blocks horizontally or vertically into the blank square. Here's an example showing some of the intermediate steps:

```
   1  3       1     3       1  2  3       1  2  3       1  2  3
 4  2  5   =>  4  2  5   =>  4     5   =>  4  5     =>  4  5  6
 7  8  6       7  8  6       7  8  6       7  8  6       7  8

 initial                                                 goal
```

Write a program that reads the initial board and yields a sequence of board positions that solves the puzzle in the fewest number of moves. Also print out the total number of moves and the total number of states ever enqueued.

Input Description

You'll be given the puzzle as a series of numbers and a blank. You can assume that it is a *3x3* grid and that you have (3*3)-1 (8) tiles. Example:

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

Output Description

Output the steps you took to get to the solution.

Challenge Input

```
8 7 6
5 4 3
2 1

5  3
2 8 4
6 7 1
```

Title Coiled sentence
Difficulty Hard
Description

You'll be given a matrix of letters that contain a coiled sentence. Your program should walk the grid to adjacent squares using only left, right, up, down (no diagonal) and every letter exactly once. You should wind up with a six word sentence made up of regular English words.

Your input will be a list of integers *N*, which tells you how many lines to read, then the row and column (indexed from 1) to start with, and then the letter matrix beginning on the next line.

Input

```
6 1 1
T H T L E D
P E N U R G
I G S D I S
Y G A W S I
W H L Y N T
I T A R G I
```

(Start at the T in the upper left corner.)

Expected Output

```
THE PIGGY WITH LARYNGITIS WAS DISGRUNTLED
```

Challenge Input

```
5 1 1
I E E H E
T K P T L
O Y S F I
U E C F N
R N K O E
```

(Start with the I in the upper left corner)
Challenge Output

```
IT KEEPS YOUR NECK OFF THE LINE
```

http://en.wikipedia.org/wiki/Vehicle_routing_problem http://www.mafy.lut.fi/study/DiscreteOpt/CH6.pdf http://wps.prenhall.com/wps/media/objects/9434/9660836/online_tutorials/heizer10e_tut5.pdf

3.2 Customer Unit Delivery Scheduling

Description

You run a business where you sell doohickies, and business is booming. You're customers are all local, but you're just getting off the ground and you don't have a large fleet of trucks, just one driver. Your truck has a finite capacity, and you have to keep costs down as you make deliveries - minimize milage, maximize deliveries, etc. That's where today's challenge program comes in.

As you make delivery runs, your truck will run out of enough doohickies and so you have to return to the depot and refill it. Assume that you refill the truck to its full capacity on visiting the depot. Finally, assume the truck has an infinite energy source, so don't worry about refueling.

Input Description

You'll be given a line with an integer *N*, which tells you how many doohickies your truck can hold, and a two-tuple of coordinates (x & y) where the doohickie depot is. Then you'll be given a line with another single integer *M*, which tells you how many customers to read. Each customer line (of which there are *M*) will be how many units they want and then a two-tuple telling you the x,y coordinated where the customer is located.

Output Description

Your program should emit the sequence of stops you need to make, including depot stops, that *minimizes* the miles driven. You must deliver enough units for every customer when you stop! No customer will ask for more than *N* doohickies (your truck's capacity), and you *should* expect to travel from one customer to the next without stopping at the depot if you can deliver enough units at once.

Challenge Input

```
40 (20,20)
12
10 (20,8)
15 (31,20)
18 (13,21)
17 (30,20)
3 (20,10)
```

```
5 (11,29)
9 (28,12)
4 (14,14)
6 (32,8)
12 (1,1)
18 (3,32)
23 (5,5)
```

3.3 DNA Shotgun Sequencing

Description

DNA sequences are made up of a 4 character alphabet - A, C, T or G, that describe the nucleotide bases in a gene sequence. To ascertain the sequence of DNA, scientists use chemical methods to identify the component nucleotides in a method called DNA sequencing. DNA *shotgun* sequencing is a method whereby DNA subsequences of the same larger sequence are produced at massive parallel scale by DNA sequencing methods, and the overlap between segments is used to reconstruct the input gene. This is a fast and accurate method, and is dropping in price. Shotgun sequencing was used to perform the first entire sequence of a human's DNA, for example. For additional background information, see Wikipedia on shotgun sequencing[1].

You're working in a DNA laboratory and you have to reconstruct a gene's sequence from a series of fragments!

Formal Input Description

You'll be given a series of DNA sequence fragments, which include overlaps with neighbor sequences, but not in any specific order - it's random. Your job is to read in the fragments and reconstruct the input DNA sequence that lead to the fragments.

Formal Output Description

Your program should emit the DNA sequence it calculated.

Sample Input

```
tgca
taggcta
gtcatgcttaggcta
agcatgctgcag
tcatgct
```

Sample Output

[1] http://en.wikipedia.org/wiki/Shotgun_sequencing

```
agcatgctgcagtcatgcttaggcta
```

Challenge Input

```
gatccatctggatcctatagttcatggaaagccgctgc
tatttcaacattaattgttggttttgatacagatggtacacca
aaaagaaattcaaaaagaacaagaaaaatctgaaaaacaacaaaa
ggaatgtcaatcctatagattaactgttgaagattcaccatcagttg
tggaaataaaaatattgaaattgcagtcattagaaataaacaac
tcaagtagaatatgccatggaagcagtaagaaaaggtactgttg
tgcaagatcaattagaaaaatcgttaaattagatgaccacatt
tgtcgttgaagctgaaaaagaaattcaaaaagaacaagaaaaatct
gaaaaacaacaaaaataaattacatcaaattccttttttt
caatcgttttattagatgaacaagaaattgataaattagttgc
aatctttatcaaactgatccatctggatcctatagttcatg
gaaattgcagtcattagaaataaacaaccaatcgttttattagatg
atcgttaaattagatgaccacatttgtttaacctttgctggt
aattatacagacgttagtgaagaggaatcaattaaattagcagtta
tatactcaaagtggtggtgttagaccatttggtatttcaacattaat
ttttaggtgttgaaaagaaagcaaccgctaaacttcaaga
aagaaagcaaccgctaaacttcaagatgcaagatcaattagaaaa
ccccacctttttttttaattatcttcaagttttttttaaaaaaaaaaaaaaaaa
gaatttttagaaaagaattatacagacgttagtgaagaggaatc
agtgcaagatacgatagagcaattacagttttctcaccagatg
aattaaattagcagttagagctttattagagattgttgaaag
cagttggtgtacgtggtaaagatgttattgttttaggtgttgaa
ttcaacaacgttatactcaaagtggtggtgttagaccatttgg
ataaattacatcaaattccttttttttccccacctttttttt
aattggtcgtagttcaaagagtgttggtgaatttttagaaaag
aatatatttctaaatttattgctggtattcaacaacgt
aacaagaaattgataaattagttgctgtcgttgaagctga
gagctttattagagattgttgaaagtggaaataaaaatatt
ttaactgccgattcacgtgtattaattagtaaagcattaat
acgatagagcaattacagttttctcaccagatggtcatctttt
aaggtactgttgcagttggtgtacgtggtaaagatgttattg
tgtttaacctttgctggtttaactgccgattcacgtgtattaatt
aataatataatatatatataaatacataataatgtcaagtgcaagat
agtaaagcattaatggaatgtcaatcctatagattaactgt
tgaagattcaccatcagttgaatatatttctaaatttattgctggta
gaaagccgctgcaattggtcgtagttcaaagagtgttggt
gtcatctttttcaagtagaatatgccatggaagcagtaagaa
tgttggttttgatacagatggtacaccaaatctttatcaaact
```

Challenge Input Solution

```
1
  ↪ aataatataatatatatataaatacataataatgtcaagtgcaagatacgatagagcaattacagttttctcaccagatggtcatctttttcaagtagaatatgccatggaa
```

Credit

This same idea - shortest common superstring - was also suggested by /u/202halffound, many thanks! If you have your own idea for a challenge, submit it to /r/DailyProgrammer_Ideas, and there's a good chance we'll post it.

3.4 Elevator Scheduling

Description

Most of us have seen and ridden elevators - you crazy folks in the UK and commonwealth countries often call them "lifts" - but I'm sure I'm not the only one who has puzzled about the scheduling algorithms. Which riders do you pick up and when? Do you service requests in the order of arrival or do you work on maximal overlap?

For this challenge, you'll have to anwer those questions. You're designing an elevator scheduling algorithm for a building and you have plenty of riders to keep happy. You can have any algorithm you want as long as you stick to the constraints - the cars have a fixed capacity and speed.

Make sure you see the bonus questions after the challenge input.

Input Description

You'll be given a single integer *N* on a line. The first *N* lines will identify elevator cars and with these fields: Car identifier, capacity, vertical speed in floors per second, and starting floor. Assume instantaneous getting on or off the elevator for the riders once you arrive on the floor. Assume that the elevator *is able to* leave with the rider as soon as it is able, but it *may* linger waiting for more people to arrive - the choice is yours.

Example:

```
C1 12 .1 1
```

This translates to Car 1, capacity of 12 people, moves at .1 floors per second (ten seconds to traverse a floor up or down), and starting at floor 1.

Then you'll get another integer on a line, *M*. The next *M* lines will show riders, with fields: Rider identification, elevator request time in seconds, source floor and destination floor. Rider identification numbers will be *stable*, meaning the rider will have the same identifier the entire exercise. Examples:

```
R1 0 1 4
```

This translates to Rider 1 who at time point 0 wants to go from floor 1 to floor 4. Riders will not transit floors without an elevator.

Output Description

The main thing to show in the output is the time point at which all requests have been satisfied. (Yes, this is trying to get you guys to compete for the most efficient

algorithm). Optionally show all intermediate steps and journeys, and wait times for riders.

Challenge Input

This was randomly generated, and so it has a few "oddities" in it, like riders who get on and off on the same floor, and riders who change their destination in the next second (e.g. in the middle of a ride). You still have to satisfy *every* request.

```
2
C1 12 .1 1
C2 12 .2 1
359
R3 0 1 9
R4 1 1 11
R0 11 1 7
R2 11 1 9
R15 13 1 9
R5 26 1 4
R16 27 1 2
R1 28 1 2
R13 28 1 9
R10 32 1 3
R14 35 1 4
R8 36 1 10
R17 38 1 12
R3 49 9 9
R18 50 1 10
R7 51 1 3
R10 53 3 10
R12 54 1 6
R0 60 7 1
R1 62 2 1
R9 66 1 8
R19 66 1 6
R15 71 9 2
R11 72 1 8
R16 78 2 4
R6 82 1 12
R8 85 10 11
R10 89 10 12
R3 90 9 6
R5 94 4 7
R2 94 9 10
R6 95 12 1
R3 111 6 9
R14 114 4 5
R13 115 9 5
R19 117 6 2
R12 122 6 12
```

```
R4 123 11 7
R9 123 8 12
R6 124 1 5
R0 124 1 6
R7 127 3 3
R11 139 8 9
R7 141 3 4
R17 143 12 2
R14 143 5 5
R16 151 4 9
R5 155 7 12
R1 155 1 11
R18 159 10 10
R15 160 2 4
R19 162 2 3
R2 164 10 3
R11 164 9 9
R3 165 9 4
R12 167 12 1
R10 169 12 1
R0 174 6 9
R11 181 9 2
R18 182 10 12
R9 184 12 4
R5 185 12 11
R4 197 7 5
R2 198 3 3
R3 198 4 8
R6 199 5 5
R8 199 11 6
R13 201 5 5
R14 203 5 4
R1 205 11 12
R16 211 9 1
R6 212 5 11
R7 214 4 8
R15 216 4 6
R19 226 3 11
R1 230 12 12
R7 232 8 5
R0 234 9 12
R3 237 8 2
R17 238 2 6
R2 240 3 11
R12 240 1 3
R15 246 6 6
R13 247 5 10
R5 248 11 5
R10 249 1 6
R18 252 12 4
```

```
R9 253 4 8
R1 256 12 12
R4 257 5 12
R16 258 1 2
R13 258 10 5
R6 262 11 2
R11 263 2 7
R9 269 8 5
R3 271 2 6
R14 274 4 9
R5 282 5 12
R11 285 7 6
R16 287 2 8
R14 290 9 5
R2 297 11 4
R18 299 4 6
R13 300 5 5
R8 301 6 5
R0 303 12 3
R19 305 11 1
R7 310 5 8
R2 311 4 4
R1 315 12 8
R16 318 8 11
R8 320 5 8
R1 324 8 2
R10 325 6 9
R17 325 6 2
R2 330 4 11
R19 330 1 9
R9 332 5 5
R5 335 12 11
R18 338 6 9
R11 340 6 8
R12 342 3 9
R9 344 5 11
R12 346 9 12
R13 346 5 12
R6 351 2 2
R0 354 3 10
R10 358 9 9
R4 369 12 12
R15 370 6 8
R3 372 6 5
R17 374 2 9
R14 383 5 4
R7 389 8 1
R18 396 9 6
R12 396 12 7
R8 411 8 1
```

```
R16 419 11 3
R2 420 11 1
R10 420 9 11
R6 423 2 12
R1 423 2 8
R7 425 1 1
R11 426 8 4
R13 429 12 11
R19 430 9 7
R5 432 11 9
R15 435 8 3
R0 438 10 6
R6 444 12 9
R17 449 9 9
R14 452 4 4
R9 456 11 2
R18 460 6 7
R5 463 9 2
R12 464 7 2
R4 468 12 5
R13 468 11 6
R2 475 1 4
R19 478 7 4
R12 491 2 10
R10 496 11 7
R0 501 6 3
R2 501 4 2
R7 502 1 3
R3 502 5 7
R14 505 4 11
R6 507 9 2
R1 508 8 12
R15 510 3 1
R16 512 3 12
R11 515 4 10
R18 515 7 2
R19 517 4 3
R15 519 1 5
R9 521 2 2
R2 524 2 5
R14 525 11 2
R18 526 2 11
R4 530 5 5
R6 531 2 5
R8 536 1 5
R12 536 10 3
R16 536 12 7
R15 538 5 7
R17 538 9 5
R13 544 6 7
```

```
R10 546 7 11
R11 547 10 5
R7 548 3 1
R4 554 5 1
R3 558 7 11
R10 568 11 7
R6 570 5 5
R12 572 3 7
R7 573 1 4
R19 574 3 6
R16 576 7 3
R0 577 3 8
R4 586 1 9
R11 587 5 9
R14 587 2 4
R2 590 5 5
R5 599 2 2
R10 599 7 7
R9 601 2 4
R1 603 12 6
R3 606 11 1
R18 606 11 9
R13 610 7 11
R10 614 7 4
R17 615 5 4
R16 616 3 3
R12 617 7 10
R7 621 4 2
R6 622 5 4
R19 626 6 12
R2 628 5 11
R15 629 7 7
R14 630 4 4
R11 632 9 6
R8 632 5 3
R0 639 8 6
R6 649 4 10
R10 651 4 11
R9 653 4 6
R14 653 4 12
R4 655 9 10
R0 656 6 4
R2 660 11 5
R13 660 11 6
R3 663 1 6
R18 664 9 5
R1 667 6 7
R5 668 2 11
R12 668 10 9
R16 672 3 9
```

```
R15 675 7 4
R17 680 4 3
R7 681 2 10
R9 681 6 9
R10 686 11 10
R14 689 12 9
R4 690 10 3
R1 698 7 9
R18 698 5 8
R0 699 4 12
R19 705 12 7
R2 708 5 1
R8 712 3 8
R13 718 6 2
R0 721 12 7
R14 721 9 5
R18 722 8 7
R15 723 4 8
R14 730 5 11
R4 733 3 12
R13 738 2 4
R6 741 10 1
R10 741 10 1
R15 741 8 9
R19 743 7 2
R13 751 4 7
R3 752 6 1
R14 755 11 9
R4 758 12 2
R11 759 6 9
R5 762 11 9
R15 765 9 2
R19 770 2 6
R9 775 9 9
R12 777 9 12
R17 778 3 7
R0 780 7 3
R0 781 3 11
R18 785 7 1
R8 787 8 11
R6 788 1 11
R7 790 10 4
R19 791 6 7
R13 791 7 6
R2 792 1 1
R9 794 9 5
R10 800 1 10
R15 804 2 5
R12 807 12 1
R11 808 9 4
```

```
R5 809 9 5
R14 813 9 2
R1 819 9 11
R19 819 7 5
R16 822 9 4
R0 823 11 8
R17 828 7 2
R11 834 4 4
R8 834 11 11
R3 837 1 6
R5 839 5 4
R4 842 2 4
R2 844 1 11
R18 851 1 1
R15 854 5 8
R0 855 8 5
R6 857 11 11
R12 857 1 3
R9 858 5 11
R8 859 11 3
R10 863 10 5
R7 867 4 6
R5 869 4 6
R0 878 5 8
R6 879 11 12
R7 882 6 12
R17 883 2 10
R13 883 6 5
R8 885 3 11
R13 887 5 7
R15 888 8 6
R3 891 6 6
R6 898 12 10
R17 898 10 3
R3 899 6 5
R5 900 6 11
R18 901 1 9
R15 906 6 10
R19 907 5 12
R13 908 7 9
R11 914 4 5
R16 917 4 5
R8 924 11 11
R14 924 2 2
R0 926 8 9
R9 926 11 2
R2 935 11 7
R1 937 11 5
R10 940 5 8
R18 946 9 11
```

```
R19 946 12 4
R3 947 5 8
R8 947 11 4
R13 947 9 4
R12 948 3 4
R4 950 4 2
R9 951 2 9
R0 963 9 11
R17 973 3 3
R16 975 5 12
R18 977 11 12
R9 980 9 6
R13 980 4 9
R5 983 11 1
R3 983 8 11
R7 985 12 7
R14 985 2 8
R10 991 8 12
R19 991 4 6
R17 992 3 5
R0 993 11 6
R1 997 5 3
```

Bonus

Which improves delivery efficiency most?

- Longer linger times?
- More cars?
- Faster cars?

Code to generate rider input

```
import random

THRESH = 0.01
FLOORS = 12

class Rider(object):
    """docstring for Rider"""
    def __init__(self, name, floor):
        self.name = name
        self.rnd = random.Random()
        self.floor = floor
        self.thresh = 0.005

    def move(self):
```

```
        ret = self.rnd.random() <= self.thresh
        if ret:
            self.thresh = 0.001
        else:
            self.thresh += .0005
        return ret

    def newfloor(self):
        old, self.floor = self.floor, self.rnd.randint(1, FLOORS)
        return '%s %s' % (old, self.floor)

    def __repr__(self):
        return '%s %%d %s' % (self.name, self.newfloor())

    def __str__(self):
        return self.__repr__()

def main():
    riders = {}
    for r in range(20):
        riders['R%d' % r] = Rider('R%d' % r, 1)

    for t in range(1000):
        for r in riders.values():
            if r.move():
                print str(r).replace('%d', `t`)

main()
```

3.5 Golomb Rulers

Description

A typical ruler has many evenly spaced markings. For instance a standard 12" ruler has 13 marks along its edge, each spaced 1" apart. This is great, and allows the measurement all (integer) values of length between 1" and 12".

However, a standard ruler is grossly inefficient. For example, the distance of length 1" can be measured multiple ways on this ruler: 0 to 1, 1 to 2, 2 to 3, etc.

A mathematician named Solomon W. Golomb had an idea about making rulers more efficient, and rulers of this type are named after him. A Golomb ruler comprises a series of marks such that no two pairs of marks are the same distance apart. Below is an example. This ruler has markings that allow all integer distances between 1-6 units to be measured. Not only that, but each distance can be measured in only way way.

```
0 1    4    6
+-+-----+----+
```

You can see how you can measure every integer distance between 1 and 6:

```
  0 1    4    6
  +-+-----+----+

1 +-+
2         +----+
3   +-----+
4 +-------+
5   +----------+
6 +------------+
```

Golomb rulers are described by their **order**, which is the number of marks on their edge. The example above is an order 4 ruler. The length of a Golomb ruler is the distance between the outer two marks and, obviously, represents the longest distance it can measure. The above example has a length of 6.

There is no requirement that a Golomb ruler measures all distances up to their length – the only requirement is that each distance is only measured in one way. However, if a ruler does measure all distances, it is classified as a *perfect* Golomb ruler. The above example is a perfect Golumb ruler. Finally, a Golomb ruler is described as *optimal* if no shorter ruler of the same order exists.

Today's challenge is to determine where to place the marks on an optimal *but not necessarily perfect) Golomb ruler when given its order.

Input Description

You'll be given a single integer on a line representing the optimal Golomb ruler order. Examples:

```
3
5
```

Output Description

Your program should emit the length of the optimal Golomb ruler and the placement of the marks. Note that some have multiple solutions, so any or all of the solutions can be yielded. Examples:

```
3   3   0 1 3
5   11  0 1 4 9 11
        0 2 7 8 11
```

Here you can see that we have two solutions for a Golomb ruler of order five and length 11.

Challenge Input

```
8
7
10
```

```
20
26
```

Challenge Output

Beware the word wrap!

```
8   34  0 1 4 9 15 22 32 34
7   25  0 1 4 10 18 23 25
        0 1 7 11 20 23 25
        0 1 11 16 19 23 25
        0 2 3 10 16 21 25
        0 2 7 13 21 22 25
10  55  0 1 6 10 23 26 34 41 53 55
20  283 0 1 8 11 68 77 94 116 121 156 158 179 194 208 212 228 240 253 259
      283
26  492 0 1 33 83 104 110 124 163 185 200 203 249 251 258 314 318 343 356
      386 430 440 456 464 475 487 492
```

http://datagenetics.com/blog/february22013/index.html

3.6 Kakuro Solver

Description

Kakuro is a popular Japanese logic puzzle sometimes called a mathematical crossword. The objective of the puzzle is to insert a digit from 1 to 9 inclusive into each white cell such that the sum of the numbers in each entry matches the clue associated with it and that no digit is duplicated in any entry. It is that lack of duplication that makes creating Kakuro puzzles with unique solutions possible. Numbers in cells elsewhere in the grid may be reused.

More background on Kakuro can be found on Wikipedia[2]. There's an online version[3] you can play as well.

Input Description

You'll be given a pair of integers showing you the number of columns and rows (respectively) for the game puzzle. Then you'll be given *col* + *row* lines with the sum and the cell identifiers as *col id* and *row number*. Example:

```
1 2
3 A1 A2
```

This example means that the sum of two values in A1 and A2 should equal 3.

[2] https://en.wikipedia.org/wiki/Kakuro
[3] http://www.kakuroconquest.com/

Challenge Output

Your program should emit the puzzle as a 2D grid of numbers, with columns as letters (e.g. A, B, C) and rows as numbers (1, 2, 3). Example:

```
  A
1 1
2 2
```

Challenge Input

This puzzle is a 2x3 matrix. Note that it has non-unique solutions.

```
2 3
13 A1 A2 A3
8 B1 B2 B3
6 A1 B1
6 A2 B2
9 A3 B3
```

Challenge Output

One possible solution for the above puzzle is

```
  A  B
1 5  1
2 2  4
3 6  3
```

3.7 Kanoodle Solver

Description

The game of Kanoodle provides 12 distinctly shaped pieces (triminoes, tetraminoes and pentaminoes) and asks the player to assemble them into a 5 by 11 rectangular grid. Furthermore they're shown in one column to aide your solution in reading them in.

The pieces are (and they're given here made up with different letters to help you see them in place). Pieces may be rotated, flipped, etc to make them fit but you may not overlap them or leave any blank squares in the 5x11 grid.

```
 A
 A
AA

 B
BB
BB
```

```
 C
 C
 C
CC

 D
 D
DD
 D

 E
 E
EE
E

F
FF

  G
  G
GGG

  H
 HH
HH

I I
III

J
J
J
J

KK
KK

 L
LLL
 L
```

A solution is found when:

- Every piece is used exactly once.
- Every square in the grid is covered exactly once (no overlaps).

Note

This is an instance of the exact cover problem. There's "Algorithm X" by Knuth for finding solutions to the exact cover problem. It's not particularly sophisticated; indeed Knuth refers to it as "a statement of the obvious trial-and-error approach."

Challenge Output

The puzzle is to arrange all of the above tiles into a four sided figure with no gaps or overlaps.

Your program should be able to emit a solution to this challenge. Bonus points if you can discover all of them. It's helpful if you keep the pieces identified by their letters to indicate their uniqueness.

One solution is:

```
EEGGGJJJJII
AEEEGCDDDDI
AAALGCHHDII
BBLLLCFHHKK
BBBLCCFFHKK
```

3.8 KenKen Solver

Description

KenKen are trademarked names for a style of arithmetic and logic puzzle invented in 2004 by Japanese math teacher Tetsuya Miyamoto, who intended the puzzles to be an instruction-free method of training the brain. KenKen now appears in more than 200 newspapers in the United States and worldwide.

As in sudoku, the goal of each puzzle is to fill a grid with digits —- 1 through 4 for a 4x4 grid, 1 through 5 for a 5x5, etc. —- so that no digit appears more than once in any row or any column (a Latin square). Grids range in size from 3x3 to 9x9. Additionally, KenKen grids are divided into heavily outlined groups of cells —- often called "cages" —- and the numbers in the cells of each cage must produce a certain "target" number when combined using a specified mathematical operation (either addition, subtraction, multiplication or division). For example, a linear three-cell cage specifying addition and a target number of 6 in a 4x4 puzzle must be satisfied with the digits 1, 2, and 3. Digits may be repeated within a cage, as long as they are not in the same row or column. No operation is relevant for a single-cell cage: placing the "target" in the cell is the only possibility (thus being a "free space"). The target number and operation appear in the upper left-hand corner of the cage.

Because we can't use the same layout that a printed KenKen board does, we will have to express the board in a lengthier fashion. The board layout will be given as row and column, with rows as numbers and columns as letters. A 6x6 board, therefore, looks like this:

```
 A B C D E F G
1. . . . . . .
2. . . . . . .
3. . . . . . .
4. . . . . . .
5. . . . . . .
6. . . . . . .
```

Cages will be described as the target value, the operator to use, and then the cells to include. For example, if the upper left corner's cage covered A1 and A2 and should combine using the addition operator to a sum of 11, we would write:

```
11 + A1 A2
```

We will use standard ASCII notation for mathematical operators: +, −, /, ∗, and =. The equals sign basically says "this square is this value" - a gimme.

Sample Input

Describing the representative KenKen problem from the Wikipedia KenKen page[4] we have this as our input, describing a 6x6 grid:

```
6
11 + A1 A2
2 / B1 C1
20 * D1 D2
6 * E1 F1 F2 F3
3 - B2 C2
3 / E2 E3
240 * A3 A4 B3 B4
6 * C3 D3
6 * C3 C5
7 + D4 D5 E5
30 * E3 F4
6 * A5 B5 9 + F5 F6
8 + A6 B6 C6
2 / D6 E6
```

Sample Output

Your program should emit the grid of numbers that satisfies the rules - yield the target value for each cage using the operator specified, and ensure that no number is repeated per column and row. From the above example you should get this solution:

```
5 6 3 4 1 2
6 1 4 5 2 3
4 5 2 3 6 1
3 4 1 2 5 6
```

[4] https://en.wikipedia.org/wiki/KenKen

```
2 3 6 1 4 5
1 2 5 6 3 4
```

Challenge Input

```
6
13 + A1 B2 B1 B2
180 * C1 D1 D2 E1
9 + F1 F2 F3
2 = C2
20 * E2 E3
15 + A3 A4 A5
6 * B3 C3
11 + C4 D3 D4
3 = B4
9 + D5 E4 E5 F4
2 / B5 C5
18 + D6 E6 F5 F6
8 + A6 B6 C6
```

Challenge Output

You can see the result here: http://imgur.com/JHHt6Hg

```
1 4 3 5 2 6
3 5 2 6 4 1
4 6 1 3 5 2
5 3 6 2 1 4
6 2 4 1 3 5
2 1 5 4 6 3
```

3.9 Museum Cameras

Description

You run a museum, and you have a small budget - but you have to protect the museum with cameras. Given some descriptions of rooms, can you organize the smallest number of cameras to view the whole room?

Some assumptions and other factors for you to work with:

- Cameras can't see around corners.
- You can only place cameras in corners.
- Assume every camera has a field of view of 180 degrees, yielding a semicircular field of view.

- Assume every camera's field of view will be equal to the left and right of the line in the corner where the camera is placed; this line bisects the angle of the corner. The camera points away from the corner.
- Assume every camera has an otherwise infinite view.

Input Description

You'll be given a row with a single number *N* that tells you how many points to read. Then on the next line you'll be given *N* points in a Cartesian coordinate space to draw the bounding box of the museum room. For example:

```
3
(0,0) (3,6) (6,0)
```

This translates to (pardon my ugly ASCII art) this triangle:

Output Description

Your program should emit the position of the cameras needed to cover the area. From our example:

```
(0,0)
```

That's one possible solution (for this one any of the corners would have worked).

If the shape has no solution, emit something like "The architect has no concept of security" because maybe they're collaborating with art theives.

Challenge Input

```
4
(0,0) (5,0) (5,6) (0,6)

5
(0,0) (7,0) (7,3) (5,6) (0,6)

13
(0,5) (2,8) (5,7) (9,6) (10,9) (13,10) (13,6) (17,6) (16,3) (13,1) (7,1)
    (5,3) (2,3)
```

Notes

This is a classic computational geometry problem called the Art Gallery Problem[5]. For some ideas on calculating 2d visibility from a top down map, click here[6]

3.10 Bioinformatics 3: Predicting Protein Secondary Structures

Description

The Chou-Fasman method is an empirical technique for the prediction of secondary structures in proteins, originally developed in the 1970s by Peter Y. Chou and Gerald D. Fasman. The method is based on analyses of the relative frequencies of each amino acid in alpha helices, beta sheets, and turns based on known protein structures. From these frequencies a set of probability parameters were derived for the appearance of each amino acid in each secondary structure type, and these parameters are used to predict the probability that a given sequence of amino acids would form a helix, a beta strand, or a turn in a protein. The method is at most about 50–60% accurate in identifying correct secondary structures, and is mostly of historical significance at this point (it's been updated by better methods).

The Chou-Fasman method predicts helices and strands in a similar fashion, first searching linearly through the sequence for a "nucleation" region of high helix or strand probability and then extending the region until a subsequent four-residue window carries a probability of less than 1. As originally described, four out of any six contiguous amino acids were sufficient to nucleate helix, and three out of any contiguous five were sufficient for a sheet. The probability thresholds for helix and strand nucleations are constant but not necessarily equal; originally 1.03 was set as the helix cutoff and 1.00 for the strand cutoff.

You can find a table showing propensities for an amino acid to help form an alpha-helix or a beta-sheet at this link http://employees.csbsju.edu/hjakubowski/classes/ch331/protstructure/ or this one http://prowl.rockefeller.edu/aainfo/chou.htm along with an algorithm description.

You can learn more about the Chou-Fasman method via Wikipedia - http://en.wikipedia.org/wiki/Chou%E . Also, slide 17 of this deck describes the approach quite cleanly - http://www.slideshare.net/RoshanKarunarat fasman-algorithm-for-protein-structure .

In this challenge you'll be given a protein sequence and asked to suggest its secondary structure.

Input

```
MET LYS ILE ASP ALA ILE VAL GLY ARG ASN SER ALA LYS ASP ILE ARG THR GLU
    GLU ARG ALA ARG
```

[5] https://en.wikipedia.org/wiki/Art_gallery_problem
[6] http://www.redblobgames.com/articles/visibility/

```
VAL GLN LEU GLY ASN VAL VAL THR ALA ALA ALA LEU HIS GLY GLY ILE ARG ILE
    SER ASP GLN THR
THR ASN SER VAL GLU THR VAL VAL GLY LYS GLY GLU SER ARG VAL LEU ILE GLY
    ASN GLU TYR
GLY GLY LYS GLY PHE TRP ASP ASN HIS HIS HIS HIS HIS HIS
```

Output

Based on http://pdbj.org/mine/structural_details/2rnm

```
MET LYS ILE ASP ALA ILE VAL GLY ARG ASN SER ALA LYS ASP ILE ARG THR GLU
    GLU ARG ALA ARG
                                        B   B   B   B   B   B
VAL GLN LEU GLY ASN VAL VAL THR ALA ALA ALA LEU HIS GLY GLY ILE ARG ILE
    SER ASP GLN THR
B   B   B           B   B
THR ASN SER VAL GLU THR VAL VAL GLY LYS GLY GLU SER ARG VAL LEU ILE GLY
    ASN GLU TYR
B   B   B   B   B   B   B   B   B                   B   B   B   B          B   B
GLY GLY LYS GLY PHE TRP ASP ASN HIS HIS HIS HIS HIS HIS
```

Notes

Other interesting proteins you could analyze include 1VPX or 1BKF, they'll give you some mixed structures. Use the European Protein Databank site (for example for 1VPX http://www.ebi.ac.uk/pdbe-srv/view/entry/1vpx/secondary.html) to confirm your results.

If you have your own idea for a challenge, submit it to /r/DailyProgrammer_Ideas, and there's a good chance we'll post it.

3.11 DNA and Protein Sequence Alignment

Description

If you are studying a particular pair of genes or proteins, an important question is to what extent the two sequences are similar. To quantify similarity, it is necessary to align the two sequences, and then you can calculate a similarity score based on the alignment.

There are two types of alignment in general. A global alignment is an alignment of the full length of two sequences, for example, of two protein sequences or of two DNA sequences. A local alignment is an alignment of part of one sequence to part of another sequence.

Alignment treats the two inputs as a linear sequence to be lined up as much as possible, with optional gaps and conversions allowed. The goal is to minimize these differences.

The first step in computing a sequence alignment is to decide on a scoring system. For this exercise, we'll simplify this and give a score of $+2$ to a match and a penalty of -1 to a mismatch, and a penalty of -2 to a gap.

Here's a small example. Our two DNA sequences to align:

```
CTCTAGCATTAG
GTGCACCCA
```

One sligmment might look like this:

```
CTCTAGCATTAG
GT---GCACCCA
```

But that one adds three gaps. We can do a bit better with only one gap added (and a small shift in starting position):

```
CTCTAGCATTAG
  GT-GCACCCA
```

While not an exact match, it now minimizes the conversion penalties between the two and aligns them as best we can.

For more information and how to do this using an R package, see the chapter "Pairwise Sequence Alignment"[7], or this set of lecture notes from George Washington University[8]. The key algorithm is Needleman-Wunsch[9].

For this challenge your task is to write a program that accepts two sequences and globally aligns them. If you want to make this harder and integrate the BLOSUM matrices, you may.

Input Description

You'll be given two sequences on two lines, one line per sequence. They'll be the same type of input, DNA or protein.

Output Description

Your program should emit the aligned sequences with gaps introduced represented by dashed ("-").

Challenge Input

DNA example

```
ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTAC
ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTAC
```

Protein example

```
MTNRTLSREEIRKLDRDLRILVATNGTLTRVLNVVANEEIVVDIINQQLLDVAPKIPELENLKIGRILQRDILLKGQKSGILFVAAESLIVIDLLP

MLAVLPEKREMTECHLSDEEIRKLNRDLRILIATNGTLTRILNVLANDEIVVEIVKQQIQDAAPEMDGCDHSSIGRVLRRDIVLKGRRSGIPFVAA
```

[7] http://a-little-book-of-r-for-bioinformatics.readthedocs.org/en/latest/src/chapter4.html

[8] http://www.seas.gwu.edu/~simhaweb/cs151/lectures/module12/align.html

[9] http://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm

Challenge Output

DNA example

```
ACGTACGTAC GTACGTACGT ACGTACGTAC GTACGTACGT ACGTACGTAC
ACGTACGTAC GTACGTACGT ----ACGTAC GTACGTACGT ACGTACGTAC
```

Protein example

```
MT-----NR--T---
    LSREEIRKLDRDLRILVATNGTLTRVLNVVANEEIVVDIINQQLLDVAPKIPELENLKIGRILQRDILLKGQKSGILFVAAESLIVIDLLPTAITTYLTKTHH

MLAVLPEKREMTECHLSDEEIRKLNRDLRILIATNGTLTRILNVLANDEIVVEIVKQQIQDAAPEMDGCDHSSIGRVLRRDIVLKGRRSGIPFVAAESFIAIDLLPP
    --VGT-SA-R---SGRSICT
```

Notes

Have a cool challenge idea? Post it to /r/DailyProgrammer_Ideas!

3.12 Redistricting Voting Blocks

Description

In the US, voting districts are drawn by state legislatures once every decade after the census is taken. In recent decades, these maps have become increasingly convoluted and have become hotly debated. One method proposed to address this is to insist that the maps be drawn using the "Shortest Splitline Algorithm" (see http://rangevoting.org/FastShortestSplitline.html for a description). The algorithm is basically a recursive count and divide process:

1. Let N=A+B where A and B are as nearly equal whole numbers as possible, and N is the total population of the area to be divided.
2. Among all possible dividing lines that split the state into two parts with population ratio A:B, choose the *shortest*.
3. We now have two hemi-states, each to contain a specified number (namely A and B) of districts. Handle them recursively via the same splitting procedure.

This has some relationship to Voronoi diagrams, for what it's worth.

In this challenge, we'll ask you to do just that: implement the SS algorithm with an ASCII art map. You'll be given a map and then asked to calculate the best splitlines that maximize equal populations per district.

For instance, if we have the following populations:

```
2 1
2 1
```

And you were told you could make only 2 lines, a successfully dividied map would look like this:

```
2|1
-|
2|1
```

This splits it into 3 distinct districts with 2 members each.

Note that lines needn't go all the way across the map, they can intersect with another line (e.g. you're not cutting up a pizza). Also, all of your districts needn't be *exactly* the same, but it should be the minimum number of differences globally for the map you have.

Input Description

You'll be given a line with 3 numbers. The first tells you how many lines to draw, the second tells you how many rows and columns to read. The next *N* lines are of the map, showing people per area.

Output Description

You should emit a map with the lines drawn, and a report containing how many people are in each district.

Challenge Input

```
8 20 20
8 0 6 1 0 4 0 0 8 8 8 2 4 8 5 3 4 8 7 4
5 7 0 3 6 1 0 7 1 1 1 1 2 5 6 4 5 1 5 0
3 0 5 8 8 7 6 5 1 4 3 1 2 6 0 4 7 5 1 5
1 7 2 0 4 6 1 6 2 2 0 3 3 5 6 8 7 4 4 0
6 7 6 7 0 6 1 3 6 8 0 2 0 4 0 3 6 1 0 7
8 6 7 6 5 8 5 5 5 2 0 3 6 1 4 2 8 2 7 0
0 6 0 6 5 8 1 2 7 6 3 1 0 3 0 4 0 1 0 5
5 5 7 4 3 0 0 5 0 0 8 1 1 8 7 2 8 0 0 8
2 4 0 5 6 7 0 5 6 3 8 1 2 5 3 3 1 8 3 7
0 7 6 6 2 8 3 4 6 8 4 6 2 5 7 0 3 1 2 1
0 3 6 4 0 4 0 6 0 3 4 8 2 3 3 8 0 6 1 0
7 2 6 5 4 5 8 6 4 4 1 1 2 3 1 0 0 8 0 0
6 7 3 6 2 6 5 0 2 7 7 2 7 0 4 0 0 6 3 6
8 0 0 5 0 0 1 4 2 6 7 1 7 8 1 6 2 7 0 0
8 4 7 1 7 5 6 2 5 2 8 5 7 7 8 2 3 1 5 7
7 2 8 1 1 0 1 0 1 3 8 7 7 5 2 6 3 0 5 5
1 2 0 1 6 6 0 4 6 7 0 5 0 0 5 5 7 0 7 7
7 7 3 6 0 1 5 8 5 8 7 0 0 0 4 0 2 1 3 4
4 3 0 6 5 1 0 6 2 0 6 5 5 7 8 2 0 4 3 4
4 1 0 4 6 0 6 4 3 2 2 6 2 2 7 3 6 3 0 4
```

Credit

This challenge was suggested by user /u/Gigabyte. If you have any ideas for challenges, head on over to /r/dailyprogrammer_ideas and suggest them! If they're

good, we might use them and award you a gold medal!

3.13 Eight Husbands for Eight Sisters

Description

For a set of men $\{A,B,\ldots,Z\}$ and a set of women $\{a,b,\ldots,z\}$ they have a preference table - A would prefer to marry b, but will be satisfied to marry c; c would prefer to marry B, will be OK to marry C, etc. Matches are considered *unstable* if there exists a pair who likes each other more than their spouses. The challenge is then to construct a stable set of marriages given the preferences.

The Gale-Shapely Theorem tells us that a stable marriage is always possible, and found in $O(\ n\hat{}2\ )$ time.

Formal Input Description

You'll be given the individual (uppercase for men, lowercase for women) identifier first, then the identifiers for their preferences for each member of the set of men (uppercase letters) and women (given by lowercase letters).

Formal Output Description

You'll emit the list of pairs that satisfy the constraints.

Sample Input

```
A, b, c, a
B, b, a, c
C, c, a, b
a, C, B, A
b, A, C, B
c, A, C, B
```

Sample Output

```
(A; b)
(B; c)
(C; a)
```

Challenge Input

```
A, b, d, g, h, c, j, a, f, i, e
B, f, b, i, g, a, j, h, e, c, d
C, b, i, j, g, h, d, e, f, c, a
D, f, a, e, i, c, j, b, g, d, h
E, f, d, a, e, i, b, c, g, j, h
F, d, f, a, c, j, e, i, b, g, h
```

```
G, e, g, c, b, f, d, a, i, j, h
H, f, i, b, c, e, a, h, g, d, j
I, i, a, j, f, c, e, b, g, h, d
J, h, f, c, e, b, a, j, g, d, i
a, J, C, E, I, B, F, D, G, A, H
b, I, H, J, C, D, A, E, B, G, F
c, C, B, I, F, H, A, D, J, G, E
d, F, G, J, D, C, E, I, H, B, A
e, D, G, J, C, A, H, I, E, B, F
f, E, H, C, J, B, F, D, A, G, I
g, J, F, G, E, I, A, H, B, D, C
h, E, C, B, H, I, A, G, D, F, J
i, J, A, F, G, E, D, H, B, I, C
j, E, A, B, C, J, I, G, D, H, F
```

Challenge Output

```
(A; h)
(B; j
(C; b)
(D; e)
(F; d)
(G; g)
(H; i)
(I; a)
```

3.14 New York Street Sweeper Paths

Description

In graph theory, various walks and cycles occupy a number of theorems, lemmas, and papers. They have practical value, it turns out, in a wide variety of applications: computer networking and street sweepers among them.

For this challenge you're the commissioner of NYC street sweeping. You have to keep costs down and keep the streets clean, so you'll minimize the number of streets swept twice while respecting traffic directionality. The goal of this challenge is to visit all edges at least one while minimizing the number of streets swept to the bare minimum.

Can you find a route to give your drivers?

Input Description

Your program will be given two integers h and w on one line which tell you hot tall and wide the street map is. On the next line will be a single uppercase letter n telling you where to begin. Then the ASCII map will begin using the dimensions you were given (hxw). Your tour should end the day at the starting point (n).

You'll be given an ASCII art graph. Intersections will be named as uppercase letters `A-Z`. Streets will connect them. The streets may be bi-directional (- or |)

or one-way (one of ^ for up only, v for down only, < for left only, and > for right only) and you may not violate the rules of the road as the commissioner by driving down a one way street the wrong way. Bi-directional streets (- or |) need only be visited in one direction, not both. You don't need to return to the starting point.

Output Description

Your program should emit the intersections visited in order and the number of street segments you swept.

Challenge Input

```
3 3
F
A - B - C
|   |   v
D > E - F
^   v   v
G - H < I
```

Challenge Output

The shortest walk of all streets at least once I've been able to come up with is F-I-H-G-D-E-H-G-D-A-B-C-F-E-B, but there may be shorter ones.

Notes

http://en.wikipedia.org/wiki/Route_inspection_problem
http://www.oocities.org/harry_robinson_testing/graph_theory.htm

3.15 Word Squares Part 2

Description

Back to word squares, a type of acrostic, a word puzzle. A word square is formed using a grid with letters arranged that spell valid English language words when you read from left to right or from top to bottom. The challenge is that in arranging the words that you spell valid words.

Today's challenge is to input a set of dimensions (n*m) and work with the enable1.txt[10] dictionary file and produce a valid word square.

Input Description

You'll be given pair of integers telling you how many rows and columns to solve for. Example:

[10] https://github.com/dolph/dictionary/blob/master/enable1.txt

```
4 4
```

Output Description

Your program should emit a valid word square with the letters placed to form valid English language words. Example:

```
rose
oven
send
ends
```

Challenge Input

```
6 6
5 7
```

Challenge Output

Multiple valid ones may be produced, but here's a few:

```
glasses
relapse
imitate
smeared
tannery

garter
averse
recite
tribal
estate
reeled
```

3.16 Mission Impossible: Fooling the Anomaly Detector and Exfiltrating Data

Description

In data mining, anomaly detection (or outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset. In particular in the context of abuse and network intrusion detection, the interesting objects are often not rare objects, but unexpected bursts in activity. Usually these models are built on averages and look for outliers 3 standard deviations away; models are continually updated to account for the natural evolution of behaviors.

For example, you may have an e-commerce business that has day-of-week and hour-of-day models of transactions. If there's a significant drop or burst of activity, you know something's afoot. Similarly, if you walk outside and see a tremendous burst of traffic on the roads - perhaps a traffic jam at that point - when it's normally flowing smoothly with intermittent cars, then you know something is odd.

In this challenge, we're trying to throw off such an anomaly detector and cover our activity. We have a simplistic anomaly detector we're trying to bypass as we move sensitive corporate documents - the latest summer blockbuster about to be released - into the wild. Some points about the network we're attacking:

- The x axis is your timeline
- Values above the y axis are bytes leaving the network, below the y axis are bytes entering the network
- The anomaly detection system is very simple and based only on the history seen so far
- It's just starting out, and has no robust history
- Alarm thresholds are measured in units of standard deviations
- Calculate the volume of traffic sent by taking the area under the curve defined by the time series

Your mission, should you choose to accept it, is to maximize the rate of data exfiltration without triggering the anomaly detector. You can do this by sending and receiving data over the network to confuse the detector into thinking the traffic rates it sees are normal.

Input Description

You'll be given the anomaly detector informatio on the first line. The numers *S*, *T0* and *T1* telling you how many standard devitations the alarm is set for, and the first two data points as (+y, -y) coordinates. Example:

```
3 (100,-100) (200,-200)
```

The next line will tell you how many bytes you have to target to exfiltrate. Example:

```
700000000
```

In this case that's about 700MB.

Output Description

Your program should emit the time series of data you need to send and receive to keep your exfiltration mission a quiet secret (e.g. you didn't trip the anomaly detector).

Challenge Input

```
3 (100,-50) (125,-65)
4000000000
```

3.17 Chess Solitaire

Description

Chess solitaire is a variant of chess that sets up puzzles using the pieces and a smaller board. You're given a set of several pieces all of one color, and using the legal moves of those pieces, you're supposed to take them one by one until only one piece is left. The rules state that every move has to be a legal move for that piece, and every move must be a capture of one piece by another. The challenge then is to figure out the right sequence of moves - what piece and what capture - that leads to only one piece left at the end.

Some other differences with regular chess. Again: every move *must* be a capture. For the king, check doesn't matter. There is no "direction" or "forward", anyone can move (within their legal moves) in any direction. Pawns cannot promote, and pawns (as usual) take on the diagonal.

The board is 4x4, with rows are 1-4 and columns are a-d; the lower left is square a1, so the board looks like this:

```
d . . . .
c . . . .
b . . . .
a . . . .
  1 2 3 4
```

The pieces are given by their single character:

- R - rook
- N - knight
- B - bishop
- Q - queen
- K - king
- P - pawn

Input Description

You'll be given a 4x4 board in ASCII showing the opening board and the pieces positioned on the board. Blank squares are given by a ".". An example:

```
R . . R
. . P .
. N . .
. . . .
```

Output Description

You should give the sequence of moves needed to successfully clear the board to one piece using the rules by showing the start square (e.g. "a1") and the captured square (e.g. "b2") joined by an "x". For the above board, the solution would be:

```
d4xa4
b2xa4
a4xc3
```

Any piece can make the next move *as long as the move results in a legal capture*. It doen't have to be one piece snaking its way through the board.

Challenge Input

```
. N . .
. . B .
R . . .
. . . .

. . B .
R . . P
. R . .
. P B .

N . . .
B P . B
. . . N
R R P .
```

Challenge Output

In order of the above starting boards:

```
b4xa2
a2xc3

d3xc4
b2xc4
c1xa3
c4xa3
a3xb1

d2xb3
b1xb3
b3xd3
d3xa3
a3xa4
a4xa1
a1xc1
```

3.18 Calculate Graph Chromatic Number

Description

The chromatic number[11] of a graph G is the *smallest* number of colors needed to color the vertices of G so that no two adjacent vertices share the same color.

To color a given graph, their first step is to scour the graph for a structure called a "prism," which consists of a pair of three-holes connected to each other via three paths.

Chudnovsky and colleagues have been doing some neat work in this space, and recently celebrated some breakthroughs[12] in coloring real-world graphs.

An interesting application via this page[13]: *Akamai runs a network of thousands of servers and the servers are used to distribute content on Internet. They install a new software or update existing softwares pretty much every week. The update cannot be deployed on every server at the same time, because the server may have to be taken down for the install. Also, the update should not be done one at a time, because it will take a lot of time. There are sets of servers that cannot be taken down together, because they have certain critical functions. This is a typical scheduling application of graph coloring problem. It turned out that 8 colors were good enough to color the graph of 75000 nodes. So they could install updates in 8 passes.*

Your challenge today is to implement an algorithm that calculates the chromatic number of an undirected graph. A useful link: Graph Coloring[14] on Wikipedia.

Input Description

You'll be given a row with a two integers *n* and *m* on it telling you how many nodes (n) and how many edges (m) to parse. Then you'll be given the edges as two integers per line telling you which nodes are connected. These graphs are *undirected*, meaning the two edges have no directionality. Example:

```
10 15
0 1
0 4
0 5
1 2
1 6
2 3
2 7
3 8
3 4
4 9
5 8
```

[11] http://mathworld.wolfram.com/ChromaticNumber.html

[12] https://www.quantamagazine.org/20151020-perfect-graph-coloring/#st_refDomain=t.co&st_refQuery=/7ghlXNoW1b

[13] http://www.geeksforgeeks.org/graph-coloring-applications/

[14] https://en.wikipedia.org/wiki/Graph_coloring

```
5 7
6 8
6 9
7 9
```

Output Description

Your program should emit the minimal chromatic number of the graph. The example graph above is the well known Petersen graph (GP(5,2))[15]. The example output would be:

```
3
```

Challenge Input

```
18 153
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 10
0 11
0 12
0 13
0 14
0 15
0 16
0 17
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
1 10
1 11
1 12
1 13
1 14
```

[15] http://mathworld.wolfram.com/PetersenGraph.html

```
1 15
1 16
1 17
2 3
2 4
2 5
2 6
2 7
2 8
2 9
2 10
2 11
2 12
2 13
2 14
2 15
2 16
2 17
3 4
3 5
3 6
3 7
3 8
3 9
3 10
3 11
3 12
3 13
3 14
3 15
3 16
3 17
4 5
4 6
4 7
4 8
4 9
4 10
4 11
4 12
4 13
4 14
4 15
4 16
4 17
5 6
5 7
5 8
5 9
5 10
```

```
5 11
5 12
5 13
5 14
5 15
5 16
5 17
6 7
6 8
6 9
6 10
6 11
6 12
6 13
6 14
6 15
6 16
6 17
7 8
7 9
7 10
7 11
7 12
7 13
7 14
7 15
7 16
7 17
8 9
8 10
8 11
8 12
8 13
8 14
8 15
8 16
8 17
9 10
9 11
9 12
9 13
9 14
9 15
9 16
9 17
10 11
10 12
10 13
10 14
10 15
```

```
10 16
10 17
11 12
11 13
11 14
11 15
11 16
11 17
12 13
12 14
12 15
12 16
12 17
13 14
13 15
13 16
13 17
14 15
14 16
14 17
15 16
15 17
16 17
```

Challenge Output

The smallest I can get is 16 colors (e.g. every node has an independent color).

3.19 Congruent Numbers

Description

In mathematics, a congruent number is a positive integer that is the area of a right triangle with three rational number sides. 6 is a congruent number because it is the area of a 3,4,5 triangle. The question of determining whether a given rational number is a congruent number is called the congruent number problem.

This problem has not (as of 2012) been brought to a successful resolution. Dr Dobb's Journal recently (2009) posted a "Lousy Algorithm"[16] for finding most congruent numbers under 100. While not perfect, it illustrates a strategy for determining if a number is congruent. There's a great discussion on the bit-player blog[17] discussing various strategies to solve this challenge.

Your challenge today is to determine if a given number is congruent or not.

[16] http://www.drdobbs.com/architecture-and-design/congruent-numbers-and-the-lousy-algorith/228701878

[17] http://bit-player.org/2009/congruent-numbers

Input Description

You'll be given an integer, one per line. Example:

```
23
12
```

Output Description

Your program should emit if the number is congruent or not. Example:

```
23 CONGRUENT
12 NOT CONGRUENT
```

Challenge Input

```
6
14
44
46
29
```

Challenge Output

```
6 CONGRUENT
14 CONGRUENT
44 NOT CONGRUENT
46 CONGRUENT
29 CONGRUENT
```

Scala Solution

```
1  def pythagorean(a:Int, b:Int, c:Int) = (a*a + b*b) == (c*c)
2  def congruent(a:Int, b:Int, c:Int) = if (pythagorean(a,b,c)) { (a*b)/2 }
   ↪   else {-1}
```

3.20 Finding Friends in the Social Graph

Description

In all of our communities, we have a strong core of friends and people on the periphery of that core, e.g. people that we know that not everyone in that strong core knows. We're all familiar with these sorts of groups with the proliferation of Facebook and the like.

These networks can be used for all sorts of things, such as recommender systems or detecting collusion.

Given a social network graph identifying friendships, can you identify the largest strong group of friends who all know eachother and are connected?

Input Description

On the first line you'll be given a single integer *N* telling you how many distinct nodes are in the graph. Then you'll be given a list of edges between nodes (it's an undirected graph, so assume if you see *a b* that *a* knows *b* and *b* knows *a*). Example:

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

Output Description

Your program should emit a list of all of the members of the largest group of friends. Example:

```
4 5 6 7
```

If the graph has multiple, distinct friend groups of the same size, you can print all or any of them.

Challenge Input

About this data set, it's kind of interesting. I downloaded it from here http://networkrepository.com/soc.php .

```
% The graph dolphins contains an undirected social network of frequent
% associations between 62 dolphins in a community living off Doubtful
    Sound,
% New Zealand, as compiled by Lusseau et al. (2003). Please cite
%
%   D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and
%   S. M. Dawson, The bottlenose dolphin community of Doubtful Sound
    features
%   a large proportion of long-lasting associations, Behavioral Ecology
    and
%   Sociobiology 54, 396-405 (2003).
```

```
%
% Additional information on the network can be found in
%
%  D. Lusseau, The emergent properties of a dolphin social network,
%  Proc. R. Soc. London B (suppl.) 270, S186-S188 (2003).
%
%  D. Lusseau, Evidence for social role in a dolphin social network,
%  Preprint q-bio/0607048 (http://arxiv.org/abs/q-bio.PE/0607048)
```

And here's the data set.

```
62
11 1
15 1
16 1
41 1
43 1
48 1
18 2
20 2
27 2
28 2
29 2
37 2
42 2
55 2
11 3
43 3
45 3
62 3
9 4
15 4
60 4
52 5
10 6
14 6
57 6
58 6
10 7
14 7
18 7
55 7
57 7
58 7
20 8
28 8
31 8
41 8
55 8
21 9
29 9
```

```
38 9
46 9
60 9
14 10
18 10
33 10
42 10
58 10
30 11
43 11
48 11
52 12
34 13
18 14
33 14
42 14
55 14
58 14
17 15
25 15
34 15
35 15
38 15
39 15
41 15
44 15
51 15
53 15
19 16
25 16
41 16
46 16
56 16
60 16
21 17
34 17
38 17
39 17
51 17
23 18
26 18
28 18
32 18
58 18
21 19
22 19
25 19
30 19
46 19
52 19
```

```
31 20
55 20
29 21
37 21
39 21
45 21
48 21
51 21
30 22
34 22
38 22
46 22
52 22
37 24
46 24
52 24
30 25
46 25
52 25
27 26
28 26
28 27
31 29
48 29
36 30
44 30
46 30
52 30
53 30
43 31
48 31
61 33
35 34
38 34
39 34
41 34
44 34
51 34
38 35
45 35
50 35
38 37
40 37
41 37
60 37
41 38
44 38
46 38
62 38
44 39
```

```
45 39
53 39
59 39
58 40
53 41
55 42
58 42
48 43
51 43
47 44
54 44
51 46
52 46
60 46
50 47
58 49
52 51
56 52
62 54
58 55
```

Challenge Output

This challenge has 3 distinct sets of 5 friends. Any or all of the below will count.

```
18 10 14 58 7
30 19 46 52 22
30 19 46 52 25
```

3.21 Drainage Ditches

Description

Every time it rains on Farmer John's fields, a pond forms over Bessie's favorite clover patch. This means that the clover is covered by water for awhile and takes quite a long time to regrow. Thus, Farmer John has built a set of drainage ditches so that Bessie's clover patch is never covered in water. Instead, the water is drained to a nearby stream. Being an ace engineer, Farmer John has also installed regulators at the beginning of each ditch, so he can control at what rate water flows into that ditch.

Farmer John knows not only how many gallons of water each ditch can transport per minute but also the exact layout of the ditches, which feed out of the pond and into each other and stream in a potentially complex network.

Given all this information, determine the maximum rate at which water can be transported out of the pond and into the stream. For any given ditch, water flows in only one direction, but there might be a way that water can flow in a circle.

Input Description

You'll be given two integers on the first line, N and M. N tells you how many intersections for M number of ditches. The next N lines will inform you of the flow points (from x to y) and the capacity.

Output Description

Your program should emit the maxiumum flow rate as the pond drains.

Challenge Input

```
5 4
1 2 40
1 4 20
2 4 20
2 3 30
3 4 10
```

Challenge Output

```
50
```

via http://poj.org/problem?id=1273

3.22 Buying Groceries

Description

You walk into a grocery store with some money in your wallet, a target amount of food to buy, and one bag with a finite capcity. Identify what you can buy under those conditions. You have to buy at least a quarter pound of any one thing.

Input Description

You'll be given a row with three numbers on it representing the bag capacity in pounds, your budget in dollars, and the target calorie count for the week. Then you'll be given a series of items showing the item, its calories per pound and its price per pound. Example:

```
10 100.00 14000
Ham 600 9.00
```

This translates to a bag that can hold ten pounds, a $100 bill in your wallet, and a target alorie count of 14000 for the week (seven * 2000). Second, you can buy some ham for $9 a pound and get 600 calories per pound.

All foods *should* be one word, hopefully cut and paste didn't foul that up.

Output Description

Your program should emit one or more solutions for the shopping trip.

Challenge Input

```
10 100.00 14000
Ham 650 8.50
Lettuce 70 0.75
Cheese 1670 6.00
Tuna 830 20.00
Bread 1300 1.20
Eggs 200 4.25
Milk 130 4.00
Yogurt 475 3.75
Sugar 936 2.00
Oil 3200 3.50
Cereal 700 5.25
Chicken 500 3.00
Raisins 1776 5.00
Bananas 160 1.10
Grapes 90 4.00
Mango 85 1.25
Cherries 80 6.00
Pears 68 4.00
Apples 55 3.00
Pineapple 51 4.00
Oranges 48 6.00
Plums 43 6.00
OrangeJuice 41 4.50
Grapefruit 39 3.00
Berries 37 2.50
Papaya 30 2.75
Peaches 30 2.25
Honeydew 26 3.00
Cantaloupe 23 3.00
Strawberries 20 2.75
Watermelon 18 1.75
```

stolen from http://www.jasq.org/just-another-scala-quant/new-agey-interviews-at-the-grocery-startup

3.23 Integer Sequence Search Part 2

Description

This is part 2 of the Integer sequence search functionality. An integer sequence is a sequence (i.e., an ordered list) of integers.

Not all sequences are computable (e.g. not all have a formula that can express them), but many do.

For this challenge you'll be searching sequences expressed as a recurrence relation. We did recurrence relations as a previous exercise #206E[18].

Input Description

You'll be given two integers, *N* and *M*, which tell you how many sequences to read to form your database and then how many search queries to process, respectively. Then you'll be given the database as *N* pairs of *name* and *recurrence relation* pair. Starting conditions will be given following the "with" keyword. Then you'll be given *M* queries of a series of integers. Some notes:

- All sequences to search will be contiguous (no gaps).
- Not all sequences you will be searching for will start at the beginning.
- The overlap of the query and the sequence database will be unambiguous but is not guaranteed to overlap completely.
- Sequence names will use the OEIS naming convention.

Recurrence relations will be given as a relationship of numbers in the sequence and the starting conditions.

Example:

```
1 1
A000045 F(n) = F(n-1)+F(n-2) with F(0)=0, F(1)=1
0, 1, 1, 2, 3, 5, 8, 13
```

Output Description

Your program should emit the sequence and its OEIS identifier. Use the identifier "NOTFOUND" for those not in the database. Example:

```
0, 1, 1, 2, 3, 5, 8, 13 A000045
```

Challenge Input

```
5 6
A000045 F(n) = F(n-1)+F(n-2) with F(0)=0, F(1)=1
A000931 F(n) = F(n-2)+F(n-3) with F(0)=1, F(1)=0, F(2)=0
A001608 F(n) = F(n-2)+F(n-3) with F(0)=3, F(1)=0, F(2)=2
A000032 F(n) = F(n-1)+F(n-2)
A001608 F(n) = F(n-2)+F(n-3) with F(0)=3, F(1)=0, F(2)=2
0, 1, 1, 2, 3, 5, 8, 13
1, 0, 1, 1, 1, 2, 2, 3, 4, 5
3, 0, 2, 3, 2, 5, 5, 7, 10, 12
```

[18] http://www.reddit.com/r/dailyprogrammer/comments/2z68di/20150316_challenge_206_easy_recurrence_relations/

```
2, 1, 3, 4, 7, 11, 18, 29, 47, 76
3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17
1, 1, 5, 5, 45, 95, 465, 1165
```

Challenge Output

```
0, 1, 1, 2, 3, 5, 8, 13 A000045
1, 0, 1, 1, 1, 2, 2, 3, 4, 5 A000931
3, 0, 2, 3, 2, 5, 5, 7, 10, 12 A001608
2, 1, 3, 4, 7, 11, 18, 29, 47. 76 A000032
3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17 A001608
1, 1, 5, 5, 45, 95, 465, 1165 NOTFOUND
```

3.24 Loop Puzzle Solver

Description

The New York Times magazine recently (as of late 2015) introduced a new puzzle at the back of the magazine, dubbed "Loop". In the puzzle, you're given a grid with one or more squares blocked off, and one or more squares with a circle in it. To solve the puzze, you have to draw a continuous line - the loop - through the squares but there are some constraints:

- The loop has to visit every non-blocked cell exactly once
- The loop has to proceed across, up or down, never diagonally. You can enter a grid square on one side and leave on any of the other three.
- The loop must go through cells marked with a circle straight across, with no turn

Your challenge today is to implement a solver for a loop puzzle.

Input Description

You'll be given an ASCII grid showing you the empty squares as underscores, the blocked off squares marked with a hash mark, and the circle grid squares with a lowercase o. Example:

```
_ _ # _ _
_ _ o _ _
_ _ _ _ _
# _ _ _ _
```

Output Description

Your program should emit the ASCII art grid with the loop drawn in. For up and down, use a pipe |, for ascross use a dash −, and for a turn use a plus +; use the spaces in the input square to fill in the final parts of the loop (e.g. the dashes −).

For the circle, it's show the line in the normal gap on either side and leave the circle in place. Example:

```
+-+ # +-+
| +-o-+ |
+-+ +-+ |
# +-+ +-+
```

By filling in the gap columns we can more easily see the loop we drew.

Challenge Input

```
_ _ # _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
o _ _ _ _ _ _ o _ _
_ # _ _ _ _ _ _ o o
_ _ _ _ _ _ _ _ _ _
_ o # _ _ _ _ o _ _
_ _ o _ _ _ o _ _ _
_ _ _ _ # _ _ _ _ _
```

Challenge Solution

3.25 Severing the Power Grid

Description

In energy production, the power grid is a a large directed graph of energy consumers and producers. At times you need to cut at certain nodes and trim demand because you cannot supply enough of a load.

In DailyProgrammeropolis, all buildings are connected to the grid and all consume power to varying degrees. Some generate power because they have installed on-site generation and sell the excess to the grid, some do not.

The scenario you're facing is this: due to a fault with the bulk power generation facility not local to DailyProgrammerololis, you must trim the power grid. You have connectivity data, and power consumption and production data. Your goal with this challenge is to **maximize the number of powered nodes with the generated energy you have**. Note that when you cut off a node, you run the risk the downstream ones will loose power, too, if they are no longer connected. This is how you'll shed demand, by selectively cutting the graph. You can make as many cuts as you want (there is no restriction on this).

Input Description

You'll be given an extensive set of data for this challenge. The first set of data looks like this: you'll be given a single integer on one line telling you how many

nodes to read. Then you'll be given those nodes, one per line, with the node ID, the amount of power it consumes in kWH, then how much the node generates in kWH. Not all nodes produce electricity, but some do (e.g. a wind farm, solar cells, etc), and there is obviously one that generates the most - that's your main power plant.

The next set of data is the edge data. The first line is how many edges to read, then the next *N* lines have data showing how the nodes are connected (e.g. power flows from node a to b).

Example:

```
3
0 40.926 0.0
1 36.812 1.552
2 1.007 0.0
2
0 1
0 2
```

Challenge Input

```
101
0 1.926 0.0
1 36.812 0.0
2 1.007 0.0
3 6.812 0.0
4 1.589 0.0
5 1.002 0.0
6 1.531 0.0
7 2.810 0.0
8 1.246 0.0
9 5.816 0.0
10 1.167 0.0
11 1.357 0.0
12 1.585 0.0
13 1.117 0.0
14 3.110 1.553
15 2.743 0.0
16 1.282 0.0
17 1.154 0.0
18 1.160 0.0
19 1.253 0.0
20 1.086 0.0
21 1.148 0.0
22 1.357 0.0
23 2.161 0.0
24 1.260 0.0
25 2.241 0.0
26 2.970 0.0
```

```
27 6.972 0.0
28 2.443 0.0
29 1.255 0.0
30 1.844 0.0
31 2.503 0.0
32 1.054 0.0
33 1.368 0.0
34 1.011 1.601
35 1.432 0.0
36 1.061 1.452
37 1.432 0.0
38 2.011 0.0
39 1.232 0.0
40 1.767 0.0
41 1.590 0.0
42 2.453 0.0
43 1.972 0.0
44 1.445 0.0
45 1.197 0.0
46 2.497 0.0
47 3.510 0.0
48 12.510 0.0
49 3.237 0.0
50 1.287 0.0
51 1.613 0.0
52 1.776 0.0
53 2.013 0.0
54 1.079 0.0
55 1.345 1.230
56 1.613 0.0
57 2.243 0.0
58 1.209 0.0
59 1.429 0.0
60 7.709 0.0
61 1.282 8.371
62 1.036 0.0
63 1.086 0.0
64 1.087 0.0
65 1.000 0.0
66 1.140 0.0
67 1.210 0.0
68 1.080 0.0
69 1.087 0.0
70 1.399 0.0
71 2.681 0.0
72 1.693 0.0
73 1.266 0.0
74 1.234 0.0
75 2.755 0.0
76 2.173 0.0
```

```
77 1.093 0.0
78 1.005 0.0
79 1.420 0.0
80 1.135 0.0
81 1.101 0.0
82 1.187 1.668
83 2.334 0.0
84 2.054 3.447
85 1.711 0.0
86 2.083 0.0
87 2.724 0.0
88 1.654 0.0
89 1.608 0.0
90 1.033 17.707
91 1.017 0.0
92 1.528 0.0
93 1.278 0.0
94 1.128 0.0
95 1.508 1.149
96 5.123 0.0
97 2.000 0.0
98 1.426 0.0
99 1.802 0.0
100 2.995 98.606
```

Edge data is too much to put up here. You can download it here: https://github.com/paralax/ColossalOpe

Node data generation (Python)

```
import random

ALPHA=3
CHANCE=0.05

r2 = ALPHA
for i in xrange(100):
    r1 = random.paretovariate(r2)
    r2 = random.paretovariate(r1)
    if random.random() < CHANCE:
        print "%s %.3f %.3f" % (i, r1, r2)
    else:
        print "%s %.3f 0.0" % (i, r1)
```

Edge data generation (Python)

via http://stackoverflow.com/questions/13543069/how-to-create-random-single-source-random-acyclic-directed-graphs-with-negative

```
import networkx as nx
import random

G=nx.gnp_random_graph(101,0.12,directed=True)
```

```
DAG = nx.DiGraph([(u,v,{'weight':random.randint(-10,10)}) for (u,v) in G.
    edges() if u>v])
with open('/tmp/edges', 'w') as f:
    for k,v in DAG.edges():
        f.write("%s %s\n" % (k,v))
```

3.26 Nonogram Solver

Description

Nonograms are picture logic puzzles in which cells in a grid must be colored or left blank according to numbers at the side of the grid to reveal a hidden picture. In this puzzle type, the numbers are a form of discrete tomography that measures how many unbroken lines of filled-in squares there are in any given row or column. For example, a clue of "4 8 3" would mean there are sets of four, eight, and three filled squares, in that order, with at least one blank square between successive groups.

The puzzles were invented in 1987 when Non Ishida, a Japanese graphics editor, won a competition in Tokyo by designing grid pictures using skyscraper lights that were turned on or off. Coincidentally, a professional Japanese puzzler named Tetsuya Nishio invented the same puzzles.

To solve a puzzle, one needs to determine which cells will be boxes and which will be empty. Determining which cells are to be left empty (called spaces) is as important as determining which to fill (called boxes). Later in the solving process, the spaces help determine where a clue (continuing block of boxes and a number in the legend) may spread. Simpler puzzles can usually be solved by a reasoning on a single row only (or a single column) at each given time, to determine as many boxes and spaces on that row as possible. Then trying another row (or column), until there are no rows that contain undetermined cells.

For more on Nonograms, see Wikipedia[19]. This challenge was inspired by a blog post on the December 2015 GCHQ puzzle challenge[20].

Example Input

You'll be given a line with two integers telling you how many rows and columns (n x m) the puzzle contains. Then you'll be given the columns as numbers and the rows as numbers, along with the grid as dots `.` arranged in an n x m grid. Example showing a 4x4 grid:

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

[19] https://en.wikipedia.org/wiki/Nonogram

[20] http://neilmitchell.blogspot.com/2015/12/solving-gchq-puzzle-by-hand.html

```
 1 . . . .
```

Example Output

Your program should emit a solution with the shaded cells marked with a pound sign #. Example:

```
# # # #
# . . #
# # . .
. . # .
```

Challenge Input

```
10 10
   4 6 8 8 8 8 8 8 6 4
1 1 . . . . . . . . . .
3 3 . . . . . . . . . .
 10 . . . . . . . . . .
 10 . . . . . . . . . .
 10 . . . . . . . . . .
 10 . . . . . . . . . .
  8 . . . . . . . . . .
  6 . . . . . . . . . .
  4 . . . . . . . . . .
  2 . . . . . . . . . .
```

Challenge Output

```
..#....#..
.###..###.
##########
##########
##########
##########
.########.
..######..
...####...
....##....
```

(Yeah, this one was for fun.)

3.27 Number Grid Puzzles

Description

Imagine a grid of numbers, not unlike soduku. Your task is to place digits in them following some rules, but then it gets different than soduku. First, each row and

column of the grid forms a multi-digit number. Second, each of those numbers must meet some specific properties.

Today's challenge is to play those games.

Input Description

You'll be given the description of the puzzle on the first two lines. The first line of input has the size of the grid as *x* * *y*. The second line tells you what digits are available. Then the next x*y numbers tell you the row and column constraints. The language will be regular. Descriptions will be "col" or "row" for column or row, respectively, and then the number. Then the constrains of the row or column will be given. Choices will include "prime", "cube", or "multiple of N" where N is an integer. Example:

```
2 2
1 2 3 4
row 1 prime
row 2 multiple of 8
col 1 prime
col 2 multiple of 4
```

Output Description

Your program should emit a valid solution to the puzzle. From the above example:

```
4 1
3 2
```

This one works because both 41 and 43 are primes, 12 is a multiple of 4 and 32 is a multiple of 8.

Challenge Input

```
3 3
1 2 3 4 5 6 7 8 9
row 1 even
row 2 prime
row 3 cube
col 1 prime
col 2 cube
col 3 prime

2 2
1 3 5 7
row 1 prime
row 2 prime
col 1 prime
col 2 prime
```

Challenge Output

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

NO SOLUTION
```

From http://chalkdustmagazine.com/regulars/puzzles/puzzles-on-square-grids/#more-1396

3.28 Nurikabe Puzzle Solver

Description

A "nurikabe" is a kind of Japanese spirit that manifests as a wall that impedes or misdirects walking travelers at night. To solve a Nurikabe puzzle, paint some cells black, representing "ocean," leaving the remaining cells white, representing "islands." Island cells connect left/right and up/down, but not diagonally. The same is true of ocean cells. Each island must contain exactly one numbered cell, which describes its area in number of cells. When the puzzle is done, all the ocean cells must be connected. No 2 x 2 cells can be completely ocean (although they can be completely island).

For today's challenge you'll be asked to write a program that can solve a Nurikabe puzzle.

The Wikipedia page on Nurikabe[21] is pretty good and includes a solution strategy you may want to implement.

Input Description

You'll be given given a row with a number *N*, telling you how many grid columns and rows there are - the grid is a square. Then you'll be given a the grid with empty cells indicated with an underscore _ and the number squares indicated, as well. A squared marked with a 1 is a gimme. Example:

```
6
_ 1 _ _ _ _
_ _ _ 4 _ _
_ 3 _ _ _ _
5 _ _ _ _ _
_ _ _ _ _ _
_ _ _ _ _ 2
```

Output Description

Your program should emit the puzzle as ASCII art with the open squares marked with an underscore _ and the filled squares with a hash mark #. Example:

[21] https://en.wikipedia.org/wiki/Nurikabe_(puzzle)

```
# 1 # # # #
# # # 4 _ #
# 3 _ # _ #
5 # _ # _ #
_ # # # # #
_ _ _ # _ 2
```

Challenge Input

```
8
_ _ 2 _ 1 _ _ 6
_ _ _ _ _ 1 _ _
_ 2 _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ 3 _
_ _ 1 _ _ _ _ _
9 _ _ 1 _ 2 _ _
```

Output Description

```
# _ 2 # 1 # # 6
# # # # # # 1 #
_ 2 # _ # # # _
# # # _ # _ _ _
_ _ _ _ # # # #
_ # # # _ _ 3 #
_ # 1 # # # # #
9 # # 1 # 2 _ #
```

via http://wordplay.blogs.nytimes.com/2016/01/18/finkel-the-switch/#more-104056

3.29 Pairs of Musical Artists

Description

Like many of you, I've more or less ditched my MP3 collection in favor of streaming services. I have tons of playlist sfor every setting (work, play, etc). A lot of these services use the co-occurrence of entries to make suggestions.

For this challenge, you'll look at playlist artists and try and identify co-occurrence of artists, which may be useful for a recommender system.

Input Description

You'll be given a set of 1000 rows, with each row a unique playlist and the artists from their playlists separated by commas. For example:

```
Radiohead,Pulp,Morrissey,Delays,Stereophonics,Blur,Suede,Sleeper,The La's
    ,Super Furry Animals
```

```
Band of Horses,Iggy Pop,The Velvet Underground,Radiohead,The Decemberists
    ,Morrissey,Television etc.
```

Output Description

Your program should emit the list of musical artists who appear together in at least fifty (50) different playlists. From the above example:

```
Radiohead + Morrissey
```

Challenge Input

The data is available via Github here: https://gist.github.com/paralax/f2bba6dbe1aa51693ba1

Notes

Shamelessly stolen from https://github.com/sming/Shapeways via http://www.reddit.com/r/compsci/comme

3.30 Text Summarizer

Description

Automatic summarization is the process of reducing a text document with a computer program in order to create a summary that retains the most important points of the original document. A number of algorithms have been developed, with the simplest being one that parses the text, finds the most unique (or important) words, and then finds a sentence or two that contains the most number of the most important words discovered. This is sometimes called "extraction-based summarization" because you are extracting a sentence that conveys the summary of the text.

For your challenge, you should write an implementation of a text summarizer that can take a block of text (e.g. a paragraph) and emit a one or two sentence summarization of it. You can use a stop word list (words that appear in English that don't add any value) from here: http://snowball.tartarus.org/algorithms/english/stop.txt

Example Input

Here's a paragraph that we want to summarize:

```
The purpose of this paper is to extend existing research on
    entrepreneurial team formation under
a competence-based perspective by empirically testing the influence of
    the sectoral context on
that dynamics. We use inductive, theory-building design to understand how
     different sectoral
characteristics moderate the influence of entrepreneurial opportunity
    recognition on subsequent
```

```
entrepreneurial team formation. A sample of 195 founders who teamed up in
     the nascent phase of
Interned-based and Cleantech sectors is analysed. The results suggest a
    twofold moderating effect
of the sectoral context. First, a technologically more challenging sector
     (i.e. Cleantech) demands
technically more skilled entrepreneurs, but at the same time, it requires
     still fairly
commercially experienced and economically competent individuals.
    Furthermore, the business context
also appears to exert an important influence on team formation dynamics:
    data reveals that
individuals are more prone to team up with co-founders possessing
    complementary know-how when they
are starting a new business venture in Cleantech rather than in the
    Internet-based sector.
Overall, these results stress how the business context cannot be ignored
    when analysing
entrepreneurial team formation dynamics by offering interesting insights
    on the matter to
prospective entrepreneurs and interested policymakers.
```

Example Output

Here's a simple extraction-based summary of that paragraph, one of a few possible outputs:

```
Furthermore, the business context also appears to exert an important
    influence on team
formation dynamics: data reveals that individuals are more prone to team
    up with co-founders
possessing complementary know-how when they are starting a new business
    venture in Cleantech
rather than in the Internet-based sector.
```

Challenge Input

```
This case describes the establishment of a new Cisco Systems R&D facility
     in Shanghai, China,
and the great concern that arises when a collaborating R&D site in the
    United States is closed
down. What will that closure do to relationships between the Shanghai and
     San Jose business
units? Will they be blamed and accused of replacing the U.S. engineers?
    How will it affect
other projects? The case also covers aspects of the site's establishment,
     such as securing an
```

```
appropriate building, assembling a workforce, seeking appropriate
    projects, developing
managers, building teams, evaluating performance, protecting intellectual
     property, and
managing growth. Suitable for use in organizational behavior, human
    resource management, and
strategy classes at the MBA and executive education levels, the material
    dramatizes the
challenges of changing a U.S.-based company into a global competitor.
```

Index