Daily Programming Challenges from Reddit

February 11, 2018

Contents

Chapter 1

Introduction

1.1 Introduction

I wrote this book originally inspired by the idea of a comparitive languages book, comparing the approaches of multiple programming paradigms: functional, object oriented, imperative, logic (e.g. Prolog), and the like. What I wound up with, instead, was a collection of programming challenges I put together over a couple of years for the Reddit /r/dailyprogrammer community.

I wrote each of these challenges in this book, although I certainly had inspiration from elsewhere in some cases. On the one hand I own the copyright for these challenges outright. On the other hand you wind up with a slightly narrow diversity in challenges.

As you'll see, I tend to favor things like graph theory, number theory (especially the generation of integer sequences), and puzzles. One of the other moderators introduced me to computational geometry and so I later threw a few of those in there. Other moderators got me thinking about computer science and algorithm fundamentals, so there are some challenges in that vein but they're not all that common. I threw in some challenges with ciphers and encryption algorithms to help illustrate the history of the field and also how approachable encryption can be.

But overall you'll find a bunch of challenges that remain variants on a theme: dynamic programming, constraint satisfaction, and such.

1.1.1 How to Use This Book

This book isn't meant to be read end to end, but instead picked from here and there. While challenges get harder through the book - progressing from easy to intermediate to hard - they don't follow a strict set of requirements for categorizing. Don't be discouraged, sometimes you can come back to a harder one after some thought and practice.

A good thing to keep in mind is that you should your time to design a solution first, then implement it. As you progress you'll learn how to exploit the features of your language of choice and improve your coding.

1.1.2 A Note on Solutions

While working on this book, and through the Reddit community, I used these challenges to work on a few langages. As such, I wrote some code here and there to solve various challenges, although not all. Remember, I had originally planned a book comparing different programming paradigms.

Solutions to some of these challenges appear peppered throughout the book. Here are some of the languages I used while coding and a brief bit about their style and such.

Python - a popular scripting language supporting object oriented and imperative paradigms. Python is a popular language for beginners and experts alike. I tend to write small prototypes in Python. Python has a lot of built-in features in many places.

FSharp - a mixed paradigm language from Microsoft that runs on the DotNet runtime. It's essentially OCaml for DotNet, mixing imperative, objects, and functional programming paradigms. I tend to favor the functional programming aspects of the language. FSharp has become my go-to functional programming language.

Go - an imperative language from Google, written in part by some of the original C and UNIX developers.

Scala - a mix-model programming language that runs on the JVM, mixing functional and object oriented aspects. While a lot of people tend to write more compact OOP in Scala, I tend to try and use it for functional programming aspects.

Challenge Input

Our much-loved enable1.txt.

Try some other dictionaries in other languages to see if you can find the highest-degree garland word in any human language.

Bonus Challenge

Find all garland words that have at least two different degrees.

Notes

This challenge was suggested by user /u/skeeto. 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.

2.16 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[8].

```
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
```

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

```
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: Adjacency 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

```
def degree(edges:String) =
    edges.
      split("\n").
      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)
```

```
14      }
15  }
16
17  func main() {
18      bdata, err := ioutil.ReadFile(os.Args[1])
19      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  }
```

2.17 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 through n - 1 (which may include negative numbers). 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

- A "decrypt" function (or method) that takes a key and the ciphertext and returns the plaintext.

Python Solution

```python
import sys

# def xor(b, s): return "".join(map(lambda x: chr(x^b), map(lambda x: ord(x), s)))
def xor(b, s): return list(map(lambda x: x^b, map(lambda x: ord(x), s)))

M = sys.maxsize
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

```scala
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
```

3.32 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[30] 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.

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

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)
```

4.30 Static HTTP Server

Description

I'm willing to bet most of you are familiar with HTTP, you're using it right now to read this content. If you've ever done any web programming you probably interacted with someone else's HTTP server stack - Flask, Apache, Nginx, Rack, etc.

For today's challenge, the task is to implement your own HTTP server. No borrowing your language's built in server (e.g. no, you can't just use Python's SimpleHTTPServer). The rules, requirements, and constraints:

- Your program will implement the bare basics of HTTP 1.0: GET requests required, any other methods (POST, HEAD, etc) are optional (see the bonus below).

- You have to write your own network listening code (e.g. `socket()`) and handle listening on a TCP port. Most languages support this, you have to start this low. Yep, learn some socket programming. `socket() ... bind() ... listen() ... accept() ...` and the like.

- Your server should handle static content only (e.g. static HTML pages or images), no need to support dynamic pages or even cgi-bin executables.

- Your server should support a document root which contains pages (and paths) served by the web server.

- Your server should correctly serve content it finds and can read, and yield the appropriate errors when it can't: 500 for a server error, 404 for a resource not found, and 403 for permission denied (e.g. exists but it can't read it).

- For it to display properly in a browser, you'll need to set the correct content type header in the response.

- You'll have to test this in a browser and verify it works as expected: content displays right (e.g. HTML as HTML, text as text, images as images), errors get handled properly, etc.

A basic, bare bones HTTP/1.0 request[39] looks like this;

```
GET /index.html HTTP/1.0
```

That's it, no Host header required etc., and all other headers like user-agent and such are optional. (HTTP/1.1 requires a host header, in contrast.)

A basic, bare bones HTTP/1.0 response[40] looks like this:

```
HTTP/1.0 200 OK
Content-type: text/html

<H1>Success!</H1>
```

The first line indicates the protocol (HTTP/1.0), the resulting status code (200 in this case means "you got it"), and the text of the status. The next line sets the content type for the browser to know how to display the content. Then a blank line, then the actual content. Date, server, etc headers are all optional.

Here's some basics on HTTP/1.0: http://tecfa.unige.ch/moo/book2/node93.html

Once you have this in your stash, you'll not only understand what more fully-featured servers like Apache or Nginx are doing, you'll have one you can customize. For example, I'm looking at extending my solution in C with an embedded Lua[41] interpreter.

---

[39] https://www.w3.org/Protocols/HTTP/1.0/spec.html#Request

[40] https://www.w3.org/Protocols/HTTP/1.0/spec.html#Response

[41] https://www.lua.org/

Bonus

Support threading for multiple connections at once.

Support HEAD[42] requests.

Support POST[43] requests.

C Solution

Years ago I wrote some C code to do this, which you can see in this gist[44]. It doesn't implement error handling well, at all.

4.31 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.

[42] https://www.w3.org/Protocols/HTTP/1.0/spec.html#HEAD

[43] https://www.w3.org/Protocols/HTTP/1.0/spec.html#POST

[44] https://gist.github.com/paralax/6f57e457b5edd7b11aae2988ae8564a0