karat

Age of Ads

Question Overview Guide

Age of Ads

Updated: June 9, 2021

karat^

Question 1 - Domain Counting

```
You are in charge of a display advertising program. Your ads are displayed
on websites all over the internet. You have some CSV input data that counts
how many times that users have clicked on an ad on each individual domain.
Every line consists of a click count and a domain name, like this:

counts = [ "900,google.com",
           "60,mail.yahoo.com",
           "10,mobile.sports.yahoo.com",
           "40,sports.yahoo.com",
           "300,yahoo.com",
           "10,stackoverflow.com",
           "20,overflow.com",
           "5,com.com",
           "2,en.wikipedia.org",
           "1,m.wikipedia.org",
           "1,mobile.sports",
           "1,google.co.uk"]

Write a function that takes this input as a parameter and returns a data
structure containing the number of clicks that were recorded on each domain
AND each subdomain under it. For example, a click on "mail.yahoo.com"
counts toward the totals for "mail.yahoo.com", "yahoo.com", and "com".
(Subdomains are added to the left of their parent domain. So "mail" and
"mail.yahoo" are not valid domains. Note that "mobile.sports" appears as a
separate domain near the bottom of the input.)

Sample output (in any order/format):

calculateClicksByDomain(counts) =>
      com:                      1345
      google.com:               900
      stackoverflow.com:        10
      overflow.com:             20
      yahoo.com:                410
      mail.yahoo.com:           60
      mobile.sports.yahoo.com:  10
      sports.yahoo.com:         50
      com.com:                  5
      org:                      3
```

```
    wikipedia.org:          3
    en.wikipedia.org:       2
    m.wikipedia.org:        1
    mobile.sports:          1
    sports:                 1
    uk:                     1
    co.uk:                  1
    google.co.uk:           1
```

Clarifications

- The input will consist of domain names only, never paths. All domains will contain at least two labels and one dot.
- The input will never deviate from the specified format; no counts or domain names which appear in any element of the input will be null or empty. (Domains might not be strictly alphabetical.)
- There will be no exact repetition of domains in the input (e.g., "yahoo.com" will not appear twice, but "mail.yahoo.com" and "yahoo.com" may both appear)
- Numbers are all small enough that there will be no integer overflow issues, either in the inputs or in the sum totals.
- Domain names have already been normalized to lowercase.
- In accordance with the “completeness over optimality” mantra, a candidate that proposed a trie-based approach should be given a warning to consider something more straightforward (but ultimately, they may proceed with that approach if they wish).
- Count numbers will be greater than zero.

Example Solution

Use a count map and split function

Question 2 - Browsing History Comparison

```
We have some clickstream data that we gathered on our client's website.
Using cookies, we collected snippets of users' anonymized URL histories
while they browsed the site. The histories are in chronological order, and
no URL was visited more than once per person.

Write a function that takes two users' browsing histories as input and
returns the longest contiguous sequence of URLs that appears in both.

Sample input:

user0 = ["/start", "/green", "/blue", "/pink", "/register", "/orange",
"/one/two"]
user1 = ["/start", "/pink", "/register", "/orange", "/red", "a"]
user2 = ["a", "/one", "/two"]
user3 = ["/pink", "/orange", "/yellow", "/plum", "/blue", "/tan", "/red",
"/amber", "/HotRodPink", "/CornflowerBlue", "/LightGoldenRodYellow",
"/BritishRacingGreen"]
user4 = ["/pink", "/orange", "/amber", "/BritishRacingGreen", "/plum",
"/blue", "/tan", "/red", "/lavender", "/HotRodPink", "/CornflowerBlue",
"/LightGoldenRodYellow"]
user5 = ["a"]
user6 = ["/pink","/orange","/six","/plum","/seven","/tan","/red", "/amber"]

Sample output:

findContiguousHistory(user0, user1) => ["/pink", "/register", "/orange"]
findContiguousHistory(user0, user2) => [] (empty)
findContiguousHistory(user0, user0) => ["/start", "/green", "/blue",
"/pink", "/register", "/orange", "/one/two"]
findContiguousHistory(user2, user1) => ["a"]
findContiguousHistory(user5, user2) => ["a"]
findContiguousHistory(user3, user4) => ["/plum", "/blue", "/tan", "/red"]
findContiguousHistory(user4, user3) => ["/plum", "/blue", "/tan", "/red"]
findContiguousHistory(user3, user6) => ["/tan", "/red", "/amber"]
```

Clarifications

- No user's browsing trail will be empty.
- If there are two or more sequences of the same length, return any one of them.
- The URLs are case sensitive, but you can assume they're normalised so /LightGoldenRod will always appear with that exact capitalization scheme. Basically, they can ignore case and we aren't going to test that their solution handles case-sensitivity at all.

Example Solution

Preprocess one user's browsing trail into a map, then iterate through the other user's browsing trail looking for matches. Use an accumulator variable to store the longest segment so far

Question 3 - Conversion Rates

```
The people who buy ads on our network don't have enough data about how ads
are working for their business. They've asked us to find out which ads
produce the most purchases on their website.

Our client provided us with a list of user IDs of customers who bought
something on a landing page after clicking one of their ads:

# Each user completed 1 purchase.
completed_purchase_user_ids = [
      "3123122444","234111110", "8321125440", "99911063"]

And our ops team provided us with some raw log data from our ad server
showing every time a user clicked on one of our ads:

ad_clicks = [
      #"IP_Address,Time,Ad_Text",
      "122.121.0.1,2016-11-03 11:41:19,Buy wool coats for your pets",
      "96.3.199.11,2016-10-15 20:18:31,2017 Pet Mittens",
      "122.121.0.250,2016-11-01 06:13:13,The Best Hollywood Coats",
      "82.1.106.8,2016-11-12 23:05:14,Buy wool coats for your pets",
      "92.130.6.144,2017-01-01 03:18:55,Buy wool coats for your pets",
      "122.121.0.155,2017-01-01 03:18:55,Buy wool coats for your pets",
      "92.130.6.145,2017-01-01 03:18:55,2017 Pet Mittens",
]

The client also sent over the IP addresses of all their users.

all_user_ips = [
      #"User_ID,IP_Address",
      "2339985511,122.121.0.155",
      "234111110,122.121.0.1",
      "3123122444,92.130.6.145",
      "39471289472,2001:0db8:ac10:fe01:0000:0000:0000:0000",
      "8321125440,82.1.106.8",
      "99911063,92.130.6.144"
]
```

Age of Ads

Updated: June 9, 2021

```
Write a function to parse this data, determine how many times each ad was
clicked, then return the ad text, that ad's number of clicks, and how many
of those ad clicks were from users who made a purchase.

Expected output:

      1 of 2       2017 Pet Mittens
      0 of 1       The Best Hollywood Coats
      3 of 4       Buy wool coats for your pets

purchases: number of purchase IDs
clicks: number of ad clicks
ips: number of IP addresses
```

Clarifications

- No user or IP address made more than one purchase or ad click.
- Not all IP addresses necessarily map to a registered user ID. A user might have clicked on an ad but left the site without registering.
- Only registered users made purchases.
- No two users share an IP. Each user logs in from only one IP.
- Output order is irrelevant.
- As the inputs are comma-separated, for simplicity, no fields themselves will contain actual commas.

Example Solution

Change completed_purchase_users to a hashset; change user_ips to a hashmap; build count maps by query string.