Unscramble Computer Science Problems

July 11, 2019

1 Adrian Lievano Project 1 Submission: Unscrambling Computer Science Problems

```
In [96]: import csv
         with open('texts.csv', 'r') as f:
             reader = csv.reader(f)
             texts = list(reader)

         with open('calls.csv', 'r') as f:
             reader = csv.reader(f)
             calls = list(reader)
```

""" TASK 0: What is the first record of texts and what is the last record of calls? Print messages: "First record of texts, texts at time " "Last record of calls, calls at time , lasting seconds" """

1.1 Task 0

1.1.1 First Record

```
In [187]: texts[0]

          print('First record of texts, {} texts {} at time {}'.format(texts[0][0],texts[0][1]
```

```
First record of texts, 97424 22395 texts 90365 06212 at time 01-09-2016 06:03:22
```

1.1.2 Last Record

```
In [188]: calls[-1]
          print('Last record of calls, {} calls {} at time {}'.format(calls[-1][0],calls[-1][1]
```

```
Last record of calls, 98447 62998 calls (080)46304537 at time 30-09-2016 23:57:15
```

1.1.3 Worse-Case Big O Notation Analysis

O(1) --> indexing into a list is constant runtime; it does not depend on the input length.

2 Task 1

""" TASK 1: How many different telephone numbers are there in the records? Print a message: "There are different telephone numbers in the records." """

Method 1: 1. Find unique telephone numbers through text database (senders & receivers) 2. Find unique telephone numbers through calls database (senders & receivers) 3. Append 4. Calculate len 5. Print message

```
In [189]: unique_num = []
          i = 0
          #check text database

          #check sending nums
          for i in range(len(texts)):
              if texts[i][0] not in unique_num:
                  unique_num.append(texts[i][0])

              if texts[i][1] not in unique_num:
                  unique_num.append(texts[i][1])

          #check calls database
          for i in range(len(calls)):
              if calls[i][0] not in unique_num:
                  unique_num.append(calls[i][0])

              if calls[i][1] not in unique_num:
                  unique_num.append(calls[i][1])

          print("There are {} different telephone numbers in the records".format(len(unique_nu
```

```
There are 570 different telephone numbers in the records
```

2.0.1 Worse-Case Big O Notation Analysis

O(n) --> As the order of input scales by n, the worst case scenario means that the amount of computations increases by n. Because the for loops are not nested, we do not have O(n^2) runtime. Space complexity is O(n).

3 Task 2

""" TASK 2: Which telephone number spent the longest time on the phone during the period? Don't forget that time spent answering a call is also time spent on the phone. Print a message: " spent the longest time, seconds, on the phone during September 2016.". """

Method:

1. Find unique call numbers and increment the starting value by the duration value in the calls list, or calls[n][3], where n corresponds to that specific number

2. Build a hash with each unique number, iterate through calls list and if the key matches the list element, n, increment the value there by the value at that key

```
In [190]: #find unique values
          unique_call_nums = []
          i = 0

          for i in range(len(calls)):
              if calls[i][0] not in unique_call_nums:
                  unique_call_nums.append(calls[i][0])
              if calls[i][1] not in unique_call_nums:
                  unique_call_nums.append(calls[i][1])

          #build hash of unique CALL numbers and initialize to zero. O(n) runtime.
          num_hash = dict()
          for i in range(len(unique_call_nums)):
              num_hash[unique_call_nums[i]] = 0

          #iterate through calls list
          #If iterative element is in list of keys, increment that value to existing key's val
          j = 0

          for j in range(len(calls)):
              if calls[j][0] in num_hash.keys(): #O(1)
                  num_hash[calls[j][0]] += int(calls[j][3])

              if calls[j][1] in num_hash.keys(): #O(1)
                  num_hash[calls[j][1]] += int(calls[j][3])


          duration = max(num_hash.values()) #O(n)
          culprit = max(num_hash, key=num_hash.get) #O(n)

          print('{} spent the longest time, {} seconds, on the phone during September 2016.'.f
```

```
(080)33251027 spent the longest time, 90456 seconds, on the phone during September 2016.
```

3.0.1 Worse-Case Big O Notation Analysis

I implemented an algorithm that first (i) searches through n elements in the calls list (both sender & receiver) and builds another list that tracks if there are any unique elements. After, (ii) the code initializes a hash map with its keys each being a unique call number found in the calls list. From there, (iii) we again iterate through the calls list, length n, and search through the hash map to see if the phone number is equal to the key in the hash map, length m. Python's 'if in' high-level syntax iterates through the keys of length m. If we assume the worse case scenario, where each number in the calls list, n, is unique, m will be equal to n (m=n). Because we are looping twice, this algorithm in the worst case scenario is:

O(n^2).

We did, however, modify our code to include, instead of looping through keys, an if statement that checks the associated value at the designated key location, which is O(1). If we do this, our implementation can be reduced to:

O(n).

4 Task 3

""" TASK 3: (080) is the area code for fixed line telephones in Bangalore. Fixed line numbers include parentheses, so Bangalore numbers have the form (080)xxxxxxx.)

Part A: Find all of the area codes and mobile prefixes called by people in Bangalore. - Fixed lines start with an area code enclosed in brackets. The area codes vary in length but always begin with 0. - Mobile numbers have no parentheses, but have a space in the middle of the number to help readability. The prefix of a mobile number is its first four digits, and they always start with 7, 8 or 9. - Telemarketers' numbers have no parentheses or space, but they start with the area code 140.

Print the answer as part of a message: "The numbers called by people in Bangalore have codes:" The list of codes should be print out one per line in lexicographic order with no duplicates.

Part B: What percentage of calls from fixed lines in Bangalore are made to fixed lines also in Bangalore? In other words, of all the calls made from a number starting with "(080)", what percentage of these calls were made to a number also starting with "(080)"?

Print the answer as a part of a message:: " percent of calls from fixed lines in Bangalore are calls to other fixed lines in Bangalore." The percentage should have 2 decimal digits """

4.1 Part A

1. Find all the people with bangalore numbers in calls database that MADE phone calls.
2. Check who they called
3. Extract area codes & mobile prefixes
4. build unique list
5. identify fixed lines, mobile, and telemarketers
6. sort lexographically
7. Print iteratively each elemenet

```
In [191]: i = 0
          #check calls database
          bangalore_nums = []
          called_by_banga = []

          #Build total list of numbers called by bangalore numbers
          for i in range(len(calls)):
              if calls[i][0][0:5] == "(080)" :
                  bangalore_nums.append(calls[i][0])
                  called_by_banga.append(calls[i][1])

          #Build unique list of numbers called by bangalore phones
          unique_called_by_banga = []
          for i in range(len(called_by_banga)):
```

```
            if called_by_banga[i] not in unique_called_by_banga:
                unique_called_by_banga.append(called_by_banga[i])

        mobile_prefix = ['7','8','9']
        tele_pref = '140'
        pref = []

        for i in range(len(unique_called_by_banga)):
            #Fixed area code identifier
            if unique_called_by_banga[i][0] == '(':
                fixed_pref = '('
                p = 0
                while unique_called_by_banga[i][p] != ')':
                    fixed_pref += unique_called_by_banga[i][p + 1]
                    p += 1
                if fixed_pref not in pref:
                    pref.append(fixed_pref)

            #Mobile called identifier
            if unique_called_by_banga[i][0] in mobile_prefix and unique_called_by_banga[i][0
                pref.append(unique_called_by_banga[i][0: len(mobile_prefix) + 1])

            #Telemarketer identifier
            if unique_called_by_banga[i][0:3] == tele_pref and unique_called_by_banga[i][0:4]
                pref.append(unique_called_by_banga[i][0:4])


        #sort call list lexographically
        s_area_codes = sorted(pref)


        #Print area code prefixes
        print('The numbers called by people in Bangalore have codes:')
        print('')

        for i in range(len(s_area_codes)):
            print('{}'.format(s_area_codes[i]))
```

```
The numbers called by people in Bangalore have codes:

(022)
(040)
(04344)
(044)
(04546)
(0471)
(080)
(0821)
```

```
7406
7795
7813
7829
8151
8152
8301
8431
8714
9008
9019
9035
9036
9241
9242
9341
9342
9343
9400
9448
9449
9526
9656
9738
9740
9741
9742
9844
9845
9900
9961
```

4.1.1 Worse-Case Big O Notation Analysis

O(n^2): In particular, there are a few sections in my code that leverage python's 'if not in' structure within a for loop. In this scenario, the worse case is O(n^2). If the input list length, n, and the unique_called_by_banga, or the secondary list, denoted length m, being checked through each loop itearation, are unique, then m will equal n, and the worse run time is 0(n^2). "' unique_called_by_banga = [] for i in range(len(called_by_banga)): if called_by_banga[i] not in unique_called_by_banga: unique_called_by_banga.append(called_by_banga[i])
"'

4.2 Part B

```
In [185]: i = 0
          #check calls database
          bangalore_nums = []
```

```
called_by_banga = []

#Build total list of numbers called by bangalore numbers
for i in range(len(calls)):
    if calls[i][0][0:5] == "(080)" :
        bangalore_nums.append(calls[i][0])
        called_by_banga.append(calls[i][1])

count = 0
for i in range(len(bangalore_nums)):
    if (bangalore_nums[i][0:5] == "(080)" and called_by_banga[i][0:5] == "(080)"):
        count += 1
total_calls = len(called_by_banga)
percent_banga_to_banga = (count/total_calls)*100.0
print("%.2f percent of calls from fixed lines in Bangalore are calls to other fixed l
```

```
24.81 percent of calls from fixed lines in Bangalore are calls to other fixed lines in Bangalo
```

4.2.1 Worse-Case Big O Notation Analysis

Sequential implementation of linear for loops, based on the length of the calls database with a simple conditional check, is, in the worst case scenario:

O(n).

5 Task 4

""" TASK 4: The telephone company want to identify numbers that might be doing telephone marketing. Create a set of possible telemarketers: these are numbers that make outgoing calls but never send texts, receive texts or receive incoming calls.

Print a message: "These numbers could be telemarketers: " The list of numbers should be print out one per line in lexicographic order with no duplicates. """

1. Telemarkers make outgoing calls; or, in the calls database, they are never in the second index
2. Telemarkers never send texts, so we should compare the telemarkers we suspect from the prior condition and determine if any of them send a text
3. Telemarkers should not receive texts

In [206]:
```
#check calls database
#gets only callers
unique_num_callers = []
i = 0
j = 0

for i in range(len(calls)):
    if calls[i][0] not in unique_num_callers:
        unique_num_callers.append(calls[i][0])
```

```
        sender_texts = []

        for j in range(len(texts)):
                if texts[j][0] not in sender_texts:
                    sender_texts.append(texts[j][0])

        sorted(sender_texts)
        sorted(unique_num_callers)

        for ele in sender_texts:
            if ele in unique_num_callers:
                unique_num_callers.remove(ele)

        tele_nums = sorted(unique_num_callers)

        print("These numbers may be telemarkers: ")
        for i in tele_nums:
            print('{}'.format(i))
```

```
These numbers may be telemarkers:
(011)21017178
(011)68486606
(022)21884607
(022)22288051
(022)28765220
(022)28952819
(022)29303640
(022)30044349
(022)32517986
(022)34715405
(022)37572285
(022)38214945
(022)39006198
(022)40840621
(022)44927512
(022)45444747
(022)46574732
(022)47410783
(022)60273083
(022)62120910
(022)65548497
(022)66911540
(022)68535788
(022)69042431
(033)25441815
(040)26738737
(040)30429041
(040)34008657
```

(040)36649724
(040)66729318
(040)69695864
(04344)211113
(04344)228249
(04344)316423
(04344)322628
(04344)615310
(04344)617351
(04344)649705
(044)20550065
(044)22020822
(044)24037112
(044)25144377
(044)27523585
(044)27641880
(044)30360652
(044)30727085
(044)38044356
(044)41581342
(044)45416964
(044)45838604
(044)48154960
(044)49481100
(044)49868415
(044)69775060
(04546)218519
(04546)267875
(04546)388977
(0471)2171438
(0471)2225098
(0471)2953539
(0471)4255177
(0471)6537077
(0471)6579079
(080)20123809
(080)20227149
(080)20383942
(080)21129907
(080)21652896
(080)21697299
(080)22759842
(080)22816760
(080)23802940
(080)24444677
(080)25820765
(080)25863765
(080)26097534

(080)27498339
(080)29435303
(080)29483476
(080)30231886
(080)30270642
(080)31606520
(080)31863188
(080)31982490
(080)32255824
(080)32390650
(080)32638303
(080)32647101
(080)32679828
(080)32828889
(080)33118033
(080)33251027
(080)33277651
(080)34121098
(080)34932254
(080)35121497
(080)35538852
(080)35986130
(080)35987804
(080)37913009
(080)39755879
(080)39991213
(080)40362016
(080)40395498
(080)40929452
(080)41095396
(080)41203315
(080)41336994
(080)41712046
(080)43206415
(080)43215621
(080)43562014
(080)43685310
(080)43901222
(080)44046839
(080)44050207
(080)44076727
(080)44357306
(080)44389098
(080)45291968
(080)45547058
(080)45687418
(080)46221576
(080)46304537

(080)46566171
(080)46702492
(080)46772413
(080)47459867
(080)47999451
(080)48462898
(080)49328664
(080)49796269
(080)60062475
(080)60068611
(080)60463379
(080)60998034
(080)61123756
(080)61419142
(080)62164823
(080)62342282
(080)62963633
(080)63623429
(080)64015211
(080)64047472
(080)64431120
(080)64765396
(080)64819785
(080)65023950
(080)65275591
(080)66044294
(080)66857551
(080)66955387
(080)67362492
(080)67426410
(080)68104927
(080)68739140
(080)69104549
(080)69150162
(080)69245029
(080)69564399
(080)69609453
(080)69887826
(0821)2135265
(0821)3257740
(0821)3537229
(0821)3602212
(0821)3774599
(0821)4753474
(0821)4816394
(0821)6141380
(08214175)358
(0824)2022081

(0824)2145844
(0824)6366719
1400481538
1401747654
1402316533
1403072432
1403579926
1404073047
1404368883
1404787681
1407539117
1408371942
1408409918
1408672243
1409421631
1409668775
1409994233
70127 59322
74064 66270
74065 10917
74292 23928
74298 18325
74298 85702
77956 55474
78130 36804
78134 03625
78135 69048
78136 54214
78138 93826
78290 99865
78291 94593
78293 38561
78295 20931
78993 89387
81513 36123
84313 80377
84319 52539
87144 42283
87144 55014
87146 58071
87149 75762
89071 32787
89071 50880
90085 20915
90089 69682
90192 30758
90193 16567
90193 61937

90196 73585
90197 38885
90199 67471
90351 90193
90355 49499
90357 25284
90368 95100
92414 69419
92415 66985
92423 51078
92426 65661
92426 72402
93412 26084
93414 19669
93426 76415
93427 40118
93428 98469
93430 54160
93432 24657
94001 62063
94002 85593
94005 20878
94489 82688
94495 03761
95263 76972
95266 42732
96569 95359
97393 52893
97402 57057
97403 88244
97404 30456
97404 90013
97407 84573
97410 27512
97414 35000
97416 18084
97418 46131
97418 59299
97419 90520
97425 79921
97426 64618
97427 87999
97429 02055
97442 45192
98440 71648
98442 73671
98443 72004
98444 16192

```
98444 63396
98445 27876
98445 71741
98445 85687
98447 62998
98448 88411
98453 94494
98457 75681
98458 94162
98459 20681
99002 80226
99617 25274
```

5.0.1 Worse-Case Big O Notation Analysis

This is a brute force algorithm, where we build two lists, using linear algorithms, and then check if the element in the sender list is in the calling list. Because the worse case scenario could mean checking n elements from the sender list to m elements, this is a:

O(n^n) algorithm, or O(n!).