Surveillance Manager

Design Proposal

Henry Wang

CSC316: Data Structures & Algorithms

hwang228@ncsu.edu

North Carolina State University

Department of Computer Science

6/10/24

System Test Plan

Instructions. In this section, you must provide your system test plan with at least 5 test cases. **If you want to provide more than 5 test cases, add an appendix at the end of this document with the 6th, 7th, etc. test cases so that page numbers for all sections match the required template! Only the first 5 test cases will be graded.**

Make sure:

- You provide your sample test data
- Test IDs are uniquely identified and descriptive
- Test descriptions are fully specified with complete inputs, specific values, and preconditions
    - Be sure to provide SPECIFIC INPUTs and VALUEs so that your test cases are repeatable
- Expected results are fully specified with specific output values
- All tests cover scenarios based on the problem statement
- All tests cover unique scenarios for the system
- All strategies for system testing are demonstrated in the tests (testing equivalence classes, testing boundary values, testing exceptions/unexpected inputs)

Test Data:

We will use two test file for our System Test Plan: `people.txt` and `calls.txt`

`people.txt`

```
134-530-7421728,Roseanna,Herman

853-257-0109509,Sarai,Rodriguez

358-721-0140950,Tereasa,Kuphal

663-879-6377778,Rudolph,Buckridge

541-777-4740981,Brett,Mueller
```

```
289-378-3996038,Daniel,Walker

903-282-4112077,Tomas,Nguyen

881-633-0099232,Enoch,Quitzon

442-000-9865092,Albertina,Braun
```

calls.txt

```
EHEI9930,Wed Apr 22 14:56:45 EDT 2020,29,358-721-0140950|663-879-6377778

IIIL0182,Mon Sep 29 14:03:37 EDT 2014,82,358-721-0140950|541-777-4740981

RIFU0090,Sun May 14 10:52:47 EDT 2017,24,663-879-6377778|541-777-4740981

VFTD2516,Mon Sep 02 10:11:43 EDT 2019,39,541-777-4740981|134-530-7421728

JAID5291,Mon Sep 09 11:10:34 EDT 2019,22,289-378-3996038|134-530-7421728

APAF2182,Mon Sep 22 14:03:37 EDT 2014,2,289-378-3996038|881-633-0099232

NANS7122,Wed Apr 29 19:26:35 EDT 2020,5,442-000-9865092|663-879-6377778|903-282-4112077
```

To start the program, run surveillanceManagerUI.java

Test ID	Description	Expected Results	Actual Results

Test #1
 	
testID: testLoadPeople
 	
Strategy:
 	
Equivalence Class - loading people from file
	Preconditions:
	
	● **surveillanceManagerUI.java has been loaded successfully**
	● **people.txt exist**
	
	Steps:
	
	1. **Click “Load people from file”**
	2. **Browse to select people.txt**
	3. **Click submit**
	4. **Check results**
		The people list loads to display the nine people in sorted order by their phone numbers
		
		`134-530-7421728 (Roseanna Herman)`
		
		`289-378-3996038 (Daniel Walker)`
		
		`358-721-0140950 (Tereasa Kuphal)`
		
		`442-000-9865092 (Albertina Braun)`
		
		`541-777-4740981 (Brett Mueller)`
		
		`663-879-6377778 (Rudolph Buckridge)`
		
		`853-257-0109509 (Sarai Rodriguez)`
		
		`881-633-0099232 (Enoch Quitzon)`
		
		`903-282-4112077 (Tomas Nguyen)`	

Test #2
 	
testID:
 	
testLoadCalls
 	
Strategy:
 	
Equivalence Class - loading calls from file
	Preconditions:
	
	• **surveillanceManagerUI.java has been loaded successfully**
	• **call.txt exists**
	
	Steps:
	
	1. **Click “Load calls from file”**
	2. **Browse to select calls.txt**
	3. **Click submit**
	4. **Check results**
		The call list loads to display 7 calls
		
		`EHEI9930,Wed Apr 22 14:56:45 EDT 2020,29,358-721-0140950|663-879-6377778`
		
		`IIIL0182,Mon Sep 29 14:03:37 EDT 2014,82,358-721-0140950|541-777-4740981`
		
		`RIFU0090,Sun May 14 10:52:47 EDT 2017,24,663-879-6377778|541-777-4740981`
		
		`VFTD2516,Mon Sep 02 10:11:43 EDT 2019,39,541-777-4740981|134-530-7421728`
		
		`JAID5291,Mon Sep 09 11:10:34 EDT 2019,22,289-378-3996038|134-530-7421728`
		
		`APAF2182,Mon Sep 22 14:03:37 EDT 2014,2,289-378-3996038|881-633-0099232`
		
		`NANS7122,Wed Apr 29 19:26:35 EDT 2020,5,442-000-9865092|663-879-6377778|903-282-4112077`	

Test #3
 	
testID: testGenerateReport
 	
Strategy:
 	
(Functional testing - examining the results of generateReport.java)
	Preconditions: Test 1 and Test 2 has passed
	
	Steps:
	
	1. **Click “Generate Report”**
	2. **Check Results**
		The following report would be generated
		
		`Calls involving 134-530-7421728 (Roseanna Herman) [`
		
		`9/2/2019 at 10:11:43 AM involving 1 other number(s):`
		
		`541-777-4740981 (Brett Mueller)`
		
		`9/9/2019 at 11:10:34 AM involving 1 other number(s):`
		
		`289-378-3996038 (Daniel Walker)`
		
		`]`
		
		`Calls involving 289-378-3996038 (Daniel Walker) [`
		
		`9/22/2014 at 14:03:37 PM involving 1 other number(s):`
		
		`881-633-0099232 (Enoch Quitzon)`
		
		`9/9/2019 at 11:10:34 AM involving 1 other number(s):`
		
		`134-530-7421728 (Roseanna Herman)`
		
		`]`
		
		Continued in Appendix	

Test #4 **testID: testGenerateReportWithouCalls** **Strategy:** **Exception/unexpected input - trying to generate report without calls being loaded.**	**Preconditions: Test 1 has passed** **Steps:** 1. **Click “Generate Report”** 2. **Check Results**	The report would be generated explain how no calls exist `No calls exist in the call logs.`	

Test #5 **testID: testGetWarrant** **Strategy:** **Functional testing - Testing the getWarrant.java function**	**Preconditions: Test 1 and Test 2 passed** **Steps:** 1. **Click “Get Warrant”** 2. **Enter the phone number “358-721-0140950”** 3. **Enter the hops “2”** 4. **Click “Generate”** 5. **Check results**	The system will provide a warrant with 2 hops for 358-721-0140950 (Teresa Kuphal) **Phone numbers covered by a 2-hop warrant originating from 358-721-0140950 (Tereasa Kuphal) [** **1-hop: 663-879-6377778 (Rudolph Buckridge)** **1-hop: 541-777-4740981 (Brett Mueller)** **2-hop: 442-000-9865092 (Albertina Braun)** **2-hop: 134-530-7421728 (Roseanna Herman)** **2-hop: 903-282-4112077 (Tomas Nguyen)** **]**	

Algorithm Design

Instructions. In this section, you should provide pseudocode for the specified algorithms (see the project writeup).

Make sure:

- The algorithms use the abstract data type operations as much as possible
- The algorithms contain all necessary pseudocode components to fully describe the algorithm
- The algorithms include in-line comments to explain what the algorithm is doing

Algorithm:

```
Type your algorithm in this box.
Algorithm getPeopleByHop(C, p)
    Input C, a map of (key=phone number, value=Call record) based on the provided input
file
          p, a phone number that represents the starting point
    Output a map of: (key=phone number, value=number of hops away from p)

    //Assuming that call record is simply a list of call objects
    //The toString representation of call is on the example so
    //Call[EHEI9930,Wed Apr 22 14:56:45 EDT 2020,29,358-721-0140950|663-879-6377778]
    //Is an string representation of an call object and not just a string
    //All of this is to say we can use call behaviors in pseudocode

    //Output map of phone number and value
    mapHops <- new empty Map

    //We add the first element to map hops so to not double count it later on
    //We will later remove it at the end
    mapHops.put(p, 0)

    //Variable for current hop level
    hop <- 0

    //Array based list to contain values to search
    L <- new empty List

    L.addLast(p)

    while !L.isEmpty() do
        hop <- hop + 1


        //The recipients of phone calls not already in mapHops
        //Will be checked in next iteration
        peopleCalled <- new empty List
        //Goes through each person to be checked

        for each nextPerson in L do
            //Checks if the person is valid or in the records
            personCalls <- C.get(nextPerson)
            if personCalls NOT null
                //Access the call records for that person
```

```
            //Goes through each of the calls
            for each call in personCalls do
                //Gets the list of phone number of recipients of calls
                //from instance variable
                phoneCall <- call.getNumbers()
                //If not already in mapHops, then adds them
                for each number in phoneCall do
                    if mapHops.get(number) is null then
                        mapHops.put(number, hop)
                        //Adds the next recipients as future people to be
                        //checked as long as they are children
                        peopleCalled.addLast(number)
    //Changes to next people to be checked for next iteration
    L <- children
mapHops.remove(p)
return mapHops
```

Data Structures

Instructions. You must determine which data structure(s) would be the best choice for implementing an efficient solution to the problem. You must:

- Describe all of the abstract data type(s) you are using
- Describe each of the data structure(s) you will use for each ADT.
- Briefly justify why you chose the data structure(s) in terms of runtime efficiency.
- If you need to sort any data, explain which sorting algorithm you will use, why you chose the specific sorting algorithm, and how you will sort your data structure(s)

ADTs:

The following ADT are used:

Person ADT

Call ADT

Map ADT

List ADT

The Person ADT describes and gives information based on the instance fields of the person

The behaviors around the Person ADT mainly are accessor methods, here are the following operations:

- getFirstName() - Gets the first name of the person
- getLastName() - Gets the last name of the person
- getPhone() - Gets the phone number of the person

The Call ADT describes and gives information based on call records. This is important because we will be sorting based on these instances later. It also give information to generate warrants

The behaviors around the Call ADT mainly are accessor methods, here are the following operations:

- getCallID() - Gets the call ID
- getTimestamp() - Gets the timestamp of the call
- getCallLength() - Gets the call length
- getPhone() - Gets the phone numbers involved in the call
- getPeople() - Gets the people associated with the phone number involved in the call

We also are using MAP ADT to easily contain all Person ADT information with Call ADT to help with searching and connecting a person to corresponding calls. Also used for easily connecting people with their phone numbers and hops

- get(phone: int) - Finds a person from a given phone number
- get(Person) - Finds list of call records/hops by person
- get(hops: int) - Returns list of phone numbers below a certain number of hops
- put(Person, hops: int) - Adds person and hops to map
- remove(person) - Removes the Person from the map and returns their corresponding value
- size() - Gets size of map

We are using List ADT to mainly hold call records associated with a person. Also used to hold people in a call

- get(index) - Get call/person from index

- add(String, index) - Adds a string to an index to help with presenting all people called with respect to one person
- addLast(int) - Adds an integer phone number to the end of the array list
- size() - Number of phone calls

Data Structures:
Person ADT will be a POJO: Has instance fields to contain information about each person
They would have accessor methods mainly to get details for each person by calling instance variables. This will be helpful for sorting (same number of phones). The person mainly just holds information and doesn't have any need to modify any of the fields.

Call ADT will also be POJO: Has instance fields to contain information about the call.
They would also have mostly accessor methods to get details for each person. One difference is the method getPeople(), which utilizes a map to translate the List of phone numbers (an instance variable) into a list of Person objects.

Map ADT will be a search table. Most of our Map ADT behaviors are searching and getting methods. Our use of put(person, hops) is limited to where the map is at most the number of people. The only time we use the remove function is to eliminate the dummy variable for p in getPeopleByHop. Also, the number of put methods is comparatively not a lot compared to the amount of get methods we use to access phone records from people. The search table is strictly binary search and O(log(n)) for the get methods. Whereas a skip table is on average O(log(n)). It means that it could be worse. The comparative value of O(log(n)) for the numerous get methods we call outweigh the worse performance on put (O(n) for search table) than skip table (average O(log(n)). A search table also has better efficiency than all other methods (array-based, linked list, self-organizing) which are O(n) for both put and get. I've decided that a search table would be better simply due to how there are a lot more get methods used than put methods.

List ADT would be an array-based list. We are using an array based list for its efficiency in accessing values O(1). It is better than any linked list and the same as a positional list. We also utilize the addLast behavior, which arrayList can handle in constant O(1). While adding not at the end, which is inefficient compared to other methods, the number of gets vastly outweighs the number of adding. We are only adding to display the calls from one person, whereas we are using get to access many different people and calls from lists. We want the add method to have an index because we are adding based off sorting on timestamp (most recent). So adding isn't necessarily sequential. We also need to automatically resize (multiply current capacity by 2) to manage the number of call records or number of people involved in a call so as to not overflow capacity. But overall, a simple array-based list which is fast in its accessor is useful and fast.

Sorting:

The things we are sorting are the phone number, dates, call ID, hops, and last/first/phone number.

When sorting by phone number, we will be using radix sort because phone numbers are integers and unique (excluding the dashes). The time complexity of O(13*n) is better than merge sort of O(nlogn)

However, when sorting all other non-integer values, we cannot use radix sort. We will be using merge sort because it has better time complexity O(nlogn) than insertion, selection, and bubble sort $O(n^2)$

Algorithm Analysis

Algorithm Analysis:

Algorithm:	Analysis/Runtime Rationale

Algorithm getPeopleByHop(C, p)
    Input C, a map of (key=phone number, value=Call record) based on the provided input file
          p, a phone number that represents the starting point
    Output a map of: (key=phone number, value=number of hops away from p)
 	
    mapHops <- new empty Map
    mapHops.put(p, 0)
 	
    hop <- 0
    L <- new empty List
    L.addLast(p)
 	
    while !L.isEmpty() do
        hop <- hop + 1
        peopleCalled <- new empty List
        for each nextPerson in L do
            personCalls <- C.get(nextPerson)
            if personCalls NOT null
                for each call in personCalls do
                    phoneCalls <- call.getNumbers()
 	
                    for each number in phoneCall do
                        if mapHops.get(number) is null then
 	
                            mapHops.put(number, hop)
                            peopleCalled.addLast(number)
	n is number of people in C
	z is number of calls per person
	
	O(1)
	O(1) //First element so not to computationally heavy
	
	O(1)
	O(1)
	O(1) //ArrayList addLast has time complexity O(1) ignoring the automatic resizing
	O(1)_ //Start of iteration with O(1) for isEmpty
	O(1)
	O(1)
	__
	O(log n)
	O(1)
	___ //Access call records per person
	O(1) //Access the instance variable for list of phone numbers called in a call
	____
	O(log n) //Is log n because mapHop can have most n people, doesn't make sense to have people not already in C
	O(n)
	O(1)
	____ O(n(log n + n + 1)) //Max number of people involved in the call is n, can't be more people than recorded in a call
	___ O(z(n(log n + n + 1) + 1)) //Reason why can't

	simplify further is because two people (n=2) can have twenty calls between them (z=2)
	__ O(n(z(n(log n + n + 1) + 1) + log n + 1)) //Can at most be n, happens when everyone is one hop away so the first person calls everyone
	_ O(1 * n(z(n(log n + n + 1) + 1) + log n + 1) + 2n))
	//The while loop only checks if the list is empty and continues with the code if not. It works in tandem to the for loop as it accesses the values in the list. So the while loop doesn't multiply with for loop time complexity, rather they sum to n.
	
	//The sum of iterations in the for loop will sum to at most n (3 in first pass, 2 in second pass, etc). L has to catch every person in the loop, people called multiple times will not be put back into the list. So the for loop iteration will only happen at most n times.
	
	//This while loop only acts as a check. Nothing to multiply by (except for the constant O(2) from incrementing hop and creating new list, which is negligible to overall run time, worse case if every person only had one other person to talk to than their initial connection

L <- children return mapHops	O(1) O(1) //Caring only about the biggest, which neglects the constant runtime outside the while statement, our biggest is $O(n^3z)$

Software Design

Instructions. In this section, you must present your software design for implementing your proposed algorithm. You must provide a UML class diagram.

Make sure:

- All UML notation is correct
- All relationships required to implement the system are present
- All classes demonstrate high cohesion
- The full data structure(s) is/are included in the UML class diagram
- The UML class diagram clearly demonstrates or follows a design pattern described by the student

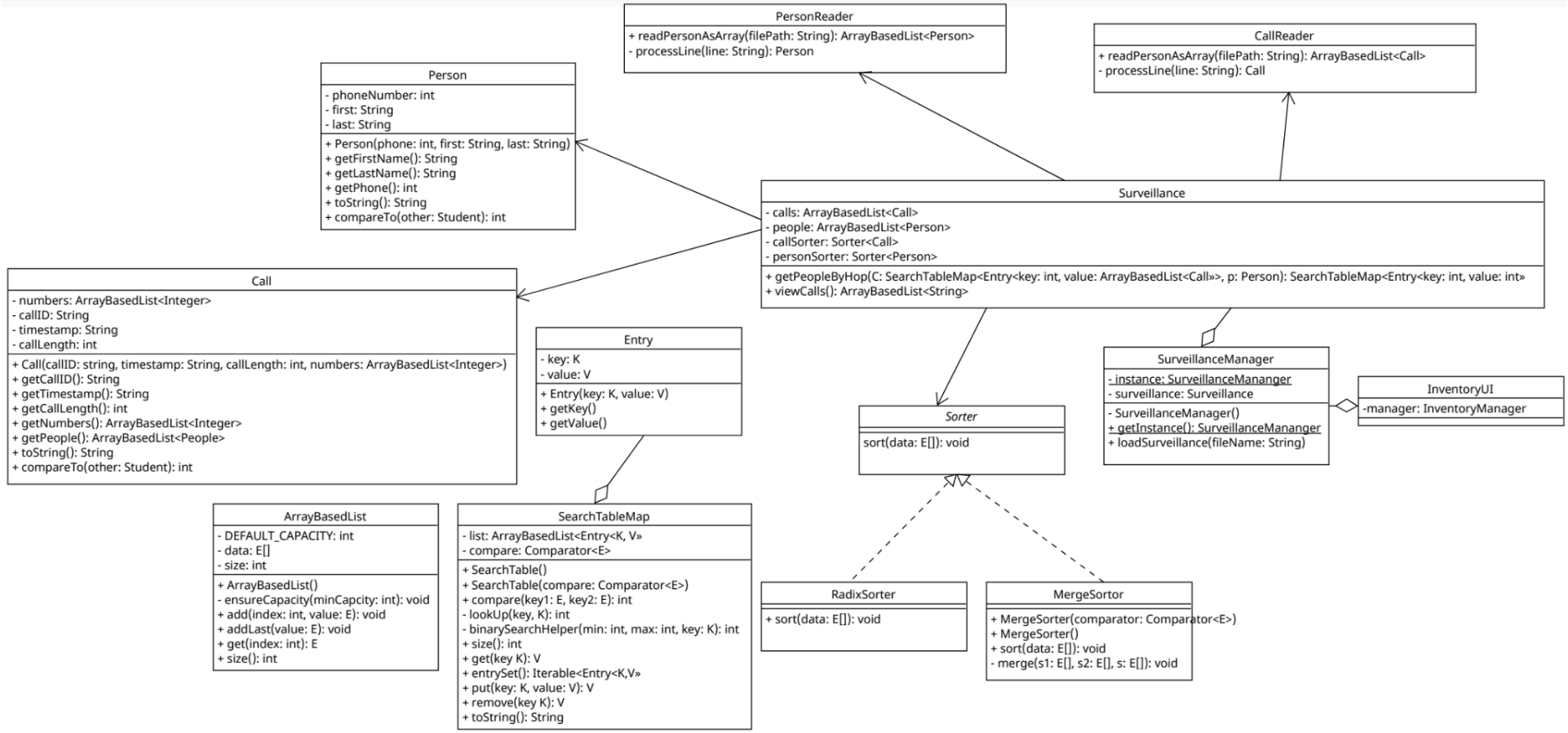
Brief Description:

We will be using a Singleton design pattern for Surveillance Manager to make only one SurveillanceManager exist. All of our ways to interact with surveillance (call records and people) are from the same records. We only have one set of people and call records so only need one SurveillanceManager.

SearchTable implements Iterable so we can use for-each loops. MergeSorter and SearchTable both implement java.util.comparator. Call and Person objects implement java.lang.comparable to help with comparison

It also makes things simple by making sure there are no inconsistencies with people and call data.
We will be using the Model-View-Controller design pattern for better user interaction and help with abstraction. We decided to use MVC to help encapsulate the data and logic from the user, who only needs to see the warrants and the list of calls, they do not need to change anything. We also used it because it helps make the coding logic distinct from coding the user interface, making it so you only have to focus on one.

UML Class Diagram:



Appendix

- If any of your responses from the previous pages overflow the area we have provided in this template for you (for example, if you need more than 2 pages to present your test data from pages 1-2), then continue your responses below in this appendix. This will help ensure your PDF submission still aligns with the template that Gradescope expects.
- If you use this appendix for overflow, be sure to reference the appendix from the relevant sections within your proposals document (for example, include ‘see the appendix for additional test data’ at the end of page 2)
- If all of your responses from the previous pages fit within the areas provided within this template, your appendix here will be empty.

Test 3 Continued Output

```
Calls involving 358-721-0140950 (Tereasa Kuphal) [

   9/29/2014 at 14:03:37 PM involving 1 other number(s):

      541-777-4740981 (Brett Mueller)

   4/22/2020 at 14:56:45 PM involving 1 other number(s):

      663-879-6377778 (Rudolph Buckridge)

]

Calls involving 442-000-9865092 (Albertina Braun) [

   4/29/2020 at 19:26:35 PM involving 2 other number(s):

      663-879-6377778 (Rudolph Buckridge)

      903-282-4112077 (Tomas Nguyen)

]

Calls involving 541-777-4740981 (Brett Mueller) [
```

```
   9/29/2014 at 14:03:37 PM involving 1 other number(s):
      358-721-0140950 (Tereasa Kuphal)
   5/14/2017 at 10:52:47 AM involving 1 other number(s):
      663-879-6377778 (Rudolph Buckridge)
   9/2/2019 at 10:11:43 AM involving 1 other number(s):
      134-530-7421728 (Roseanna Herman)
]
Calls involving 663-879-6377778 (Rudolph Buckridge) [
   5/14/2017 at 10:52:47 AM involving 1 other number(s):
      541-777-4740981 (Brett Mueller)
   4/22/2020 at 14:56:45 PM involving 1 other number(s):
      358-721-0140950 (Tereasa Kuphal)
   4/29/2020 at 19:26:35 PM involving 2 other number(s):
      442-000-9865092 (Albertina Braun)
      903-282-4112077 (Tomas Nguyen)
]
Calls involving 853-257-0109509 (Sarai Rodriguez) [
   (none)
```

```
]
Calls involving 881-633-0099232 (Enoch Quitzon) [
   9/22/2014 at 14:03:37 PM involving 1 other number(s):
      289-378-3996038 (Daniel Walker)
]
Calls involving 903-282-4112077 (Tomas Nguyen) [
   4/29/2020 at 19:26:35 PM involving 2 other number(s):
      442-000-9865092 (Albertina Braun)
      663-879-6377778 (Rudolph Buckridge)
]
```