CMPE 326 Concepts of Programming Languages

Spring 2020

Homework 1

Due date: 27/03/2020 23:59

In this homework you will develop a data analytics system in Python that processes structured data of logs. Assume that logs are related to user usage of a web or mobile application. The system can process queries so that a data analyst can explore the logs data.

The structure of a log entry is quite generic. Each log is composed of the following fields.

- **arrivaltime**: This field has the timestamp of the arrival of the log. It is the UNIX timestamp. See the standard Python package *time* and the following code snippets for its usage.
- **user**: This is a string that holds the id of the user who initiated the log.
- **session**: This is a integer number which represents the session number of the user.
- **event**: This is a string that holds the name of the event. For instance, `signin`, `signout`, `follow`, or `click` can be some events related to a Web usage session. The names of the event is of course domain-dependent and logs are generic in the sense that event name can be any string.
- **usrfield**: This is a domain dependent field holding a integer number. For example, for the `signin` event this field can hold a number showing how many unsuccessful login attempts occurred before the user signs in.

Here is the Python `Log` class.

```
class Log:
    def __init__(self, arrivaltime, user, session, event, usrfield):
        self.arrivaltime = arrivaltime
        self.user = user
        self.session = session
        self.event = event
        self.usrfield = usrfield

    def getArrivaltime(self):
        return self.arrivaltime

    def getUser(self):
        return self.user

    def getSession(self):
        return self.session

    def getEvent(self):
        return self.event

    def getUsrfield(self):
        return self.usrfield

    def __str__(self):
        return "(arrivaltime:%s, user:%s, session:%d, event:%s, usrfield:%d)" %
                        (time.ctime(self.arrivaltime),
                         self.user,
                         self.session,
                         self.event,
                         self.usrfield)
    def __repr__(self):
        return "(arrivaltime:%s, user:%s, session:%d, event:%s, usrfield:%d)" %
                        (time.ctime(self.arrivaltime),
                         self.user,
                         self.session,
                         self.event,
                         self.usrfield)
```

There are two types of queries.

Equal queries: With the help of equal queries, the user can retrieve logs that have a field with a specific value. For example, in case you want to get all the logs that have `john` value in the user field, an equal query can be handy.

Range queries: Range queries are used to retrieve logs with a field value within a range (inclusive). Unlike equal queries, range queries do not work on all fields, but only suitable for the *arrivaltime* and *session* fields. For instance, with the help of range queries one can retrieve all the logs that have arrived in between `Sat Jan 10 08:00:00 2020` and `Sat Jan 10 09:00:00 2020`.

In order to use equal or range queries, one needs to first register the query to the system. To this end, we will use Python closures where a nested function returns a function. You will be given the function `registerEqualQuery` (the definition is below) that registers an equal query. This function expects a list of log objects (check Python lists) as the first parameter and a field name ('getArrivaltime', 'getUser', etc.) as the second parameter. It returns a query function that expects a value parameter and returns a list of log objects that satisfy the equal query condition. If there are no such logs satisfying the condition, it simply returns an empty list.

```
def registerEqualQuery(logs, field):
    def query(val1):
        res = []
        for (index,log) in enumerate(logs):
            if getattr(log, field)() == val1:
                res.append(log)
        return res
    if field not in ['getArrivaltime','getUser','getSession','getEvent','getUsrfield']:
        raise Exception('field name not found')
    return query
```

Let's see an equal query in action. First we need some logs. Following Python interpreter session creates some logs.

```
>>> l1 = Log(int(time.mktime(time.strptime('Mon Mar 09 12:00:00 2020'))),
              'john', 23, 'signin', 0)
>>> l2 = Log(int(time.mktime(time.strptime('Mon Mar 09 14:10:00 2020'))),
              'mary', 42, 'click', 13)
>>> l3 = Log(int(time.mktime(time.strptime('Tue Mar 10 11:00:32 2020'))),
              'john', 23, 'signout', 0)
>>> l4 = Log(int(time.mktime(time.strptime('Wed Mar 11 22:10:00 2020'))),
              'viz', 350, 'follow', 1)
>>> l5 = Log(int(time.mktime(time.strptime('Fri Mar 13 03:33:02 2020'))),
              'admin12', 44, 'signin', 0)
>>> mylogs = [l1,l2,l3,l4,l5]
```

Now, let's register an equal query that finds all logs according to the user field.

```
>>> q1 = registerEqualQuery(mylogs,'getUser')
```

Next we want to search all logs of the user `mary`. We can easily use the registered query `q1` for this purpose. As expected `result` is a list of logs, in this case the only log that has `mary` value in the user field.

```
>>> result = q1('mary')
>>> result
[(arrivaltime:Mon Mar  9 14:10:00 2020, user:mary, session:42, event:click, usrfield:13)]
```

Below you can find further use cases.

```
>>> result = q1('john')
>>> result
[(arrivaltime:Mon Mar  9 12:00:00 2020, user:john, session:23, event:signin, usrfield:0),
(arrivaltime:Tue Mar 10 11:00:32 2020, user:john, session:23, event:signout, usrfield:0)]
>>> q2 = registerEqualQuery(mylogs, 'getArrivaltime')
>>> result = q2(time.mktime(time.strptime('Fri Mar 13 03:33:02 2020')))
>>> result
[(arrivaltime:Fri Mar 13 03:33:02 2020, user:admin12, session:44, event:signin, usrfield:0)]
>>> result = q1('mike')
>>> result
[]
>>> q3 = registerEqualQuery(mylogs, 'getSession')
>>> result = q3(350)
>>> result
[(arrivaltime:Wed Mar 11 22:10:00 2020, user:viz, session:350, event:follow, usrfield:1)]
```

Below are some examples of range queries. Note that for range queries the range is inclusive. For instance, when we use the query `q5` below in the call `q5(10,12)` it is expected to retrieve all the logs whose session fields have values greater or equal to 10 and smaller or equal to 12.

```
>>> q4 = registerRangeQuery(mylogs, 'getArrivaltime')
>>> startdate = time.mktime(time.strptime('Tue Mar 10 08:00:00 2020'))
>>> enddate = time.mktime(time.strptime('Fri Mar 13 00:00:00 2020'))
>>> result = q4(startdate,enddate)
>>> result
[(arrivaltime:Tue Mar 10 11:00:32 2020, user:john, session:23, event:signout, usrfield:0),
(arrivaltime:Wed Mar 11 22:10:00 2020, user:viz, session:350, event:follow, usrfield:1)]
>>> q5 = registerRangeQuery(mylogs, 'getSession')
>>> result = q5(10,12)
>>> result
[]
>>> result = q5(20,40)
>>> result
[(arrivaltime:Mon Mar  9 12:00:00 2020, user:john, session:23, event:signin, usrfield:0),
(arrivaltime:Tue Mar 10 11:00:32 2020, user:john, session:23, event:signout, usrfield:0)]
```

For the first part of the homework, you are expected to write the function `registerRangeQuery` with a similar signature to `registerEquaQuery` shown above. It must expect two parameters; the first one should be a list of logs and the second one should be a string corresponding the field (i.e., either session or arrivaltime fields). The function must return another function that expects two parameters p1 and p2 and should return a list of logs that has the value for the registered field in between p1 and p2 inclusively.

Optimized Queries

Consider that you have stored big number of logs in the system. For instance, the `biglogdata` below is a list of 1000000 logs (you will be provided with a utility function that generates random logs.)

```
>>> q6 = registerEqualQuery(biglogdata, 'getUser')
>>> s = time.time(); result = q6('abc'); e = time.time()
>>> e-s
0.28475475311279297
>>> 1/(e-s)
3.5117938825200024
```

The query `q6` took around 0.284 seconds to execute. This means the system can process roughly 3.5 queries per second. This is pretty inefficient considering that multiple users can access the system and run queries concurrently. You need to implement versions of query registering functions that creates queries that can be processed efficiently by the system. Consider the following `q7` query registered using `registerOptimizedEqualQuery` function. With the help of this optimized query, the system can process approximately 18800 queries per second.

```
>>> q7 = registerOptimizedEqualQuery(biglogdata, 'getUser')
>>> s = time.time(); result = q7('abc'); e = time.time()
>>> 1/(e-s)
18808.538116591928
```

Here is a similar comparison for a range query.

```
>>> q8 = registerRangeQuery(biglogdata, 'getArrivaltime')
>>> q9 = registerOptimizedRangeQuery(biglogdata, 'getArrivaltime')
>>> startdate = time.mktime(time.strptime('Mon Feb 10 00:00:00 2020'))
>>> enddate = time.mktime(time.strptime('Mon Feb 10 01:00:00 2020'))
>>> s = time.time(); result = q8(startdate,enddate); e = time.time()
>>> 1/(e-s)
2.028496520998544
>>> s = time.time(); result = q9(startdate,enddate); e = time.time()
>>> 1/(e-s)
824.8385447394297
```

In this part of the homework you must implement the functions `registerOptimizedEqualQuery` and `registerOptimizedRangeQuery` with the signatures as explained above. These functions implement efficient queries so that you should get ratios (ratio of number of queries that can be processed per seconds for normal and optimized queries) similar to the ones showcased in above tests. Of course, the ratio of the efficiency can vary from size of the logs data or size of the result of the query. But, you are expected to achieve similar ratios in similar situations (for example for data of 1000000 logs). Additionally, you will be provided with some test cases by your teaching assistant.

Implementation Language

You must use Python as the implementation language. You can appreciate some of the properties of Python while performing this task.

First of all since Python is dynamically typed language, you can prototype and develop programs fast.

Additionally, high-level data structures like *lists* and *dictionaries* are built-in for Python. Hence, you can benefit from them to form advanced data structures easily for storing and querying the data used in the homework. Actually, Python **dictionaries** are the biggest hint for successfully implementing this homework. With a dictionary, one can have an efficient way to access a value via a key.

Note that Python features the closure concept that nicely fits to this homework.

Submission

Each person must submit **his or her own work**.

You need to submit your Python file `hw1.py` using the course Moodle page.

You are **not allowed** to use special packages. You can only use modules from the standard Python libraries.

Your program will be evaluated using the Python 3.X (min version 3.6) interpreter.

You may be asked for a demo session.

The final submission must be one file named `hw1.py`. It should include definitions of functions `registerRangeQuery`, `registerOptimizedEqualQuery`, and `registerOptimizedRangeQuery`. Do not submit any compressed file or a file having a different name. You are **not allowed** to change the `Log` class or the `registerEqualQuery` function.

There will be 1 day **questions fence** for this homework. You are not allowed to ask questions to the instructor or the teaching assistant in 1 day period before the deadline (i.e., during 27/3/2020).

Your submission will be graded w.r.t. the maximum points calculated according to the following formula: $100 - (2^{NumOfLateDays} \times 5)$.