Database Systems

Exercise 1:

Background: In this exercise, you will use sketches and sliding window algorithms to improve performance of stream analytics. Our pilot application is network monitoring. Internet service providers and large companies frequently deploy network monitoring algorithms for protecting their network from malicious attacks and unlawful actions, and for identifying possible misconfigurations and faults. These algorithms are sometimes installed at the edge routers of the corporate network, for keeping track of all incoming and outgoing networks. Today, you are asked to develop a set of sketches and algorithms that will help these companies to efficiently (in real time) summarise the network packets passing through the edge routers.

All code should be implemented and tested **in Oracle Java** (jdk 7 or jdk 8). You will not require – and you should not use – any third-party libraries. Your answers should be uploaded to Moodle by Sunday, March 19th @ midnight

Keep the following directory format for the answers:
/YourName/ExerciseNumber/TaskNumber/
e.g. for gaspar username "test" and exercise1 submit: /test/exercise1/task1/ etc.

Data sets: You are provided with an anonymized Internet trace collected by passive network monitors. The trace is stored in a TSV file (tab separated values) containing two columns (source IP, destination IP). We will refer to this data as stream1 (download from here).

Task 1 (30%). Efficient Jumping window structure

Assume that your input is the file comprising the Internet traces which arrive in a streaming fashion. Every line of the file represents a transmission of a packet from the source IP (the first IP address in the line) to the target IP (the second IP address). Every packet transmission is considered a different event, and the order of the events is the one appearing in the file.

Implement an **event-based** jumping window structure, as seen in the class (the algorithm that maintains a running sum), that enables frequency queries over the stream. Your structure will use lazy updating, and it will support the following queries:

a. Given a source IP x.y.z.w, the structure should return the number of packets that have been sent from any IP address x.*.*.* in the last W network packets, with a maximum absolute error ε*W. For example, given IP 142.212.132.4, the structure should estimate how many packets have been sent from the IPs 142.*.*.* in the last W events. Values of W and ε will be given as parameters at construction time.

b. Given a source IP x.y.z.w and a query window size $W_1 \leq W$, return the number of packets sent from the IP x.*.*.* in the last W_1 events.

Questions:

a. Explain how to compute the size of the sub-window W_{sw} that guarantees a maximum absolute error $\epsilon * W$ for a window size W.

b. Compute the maximum memory required for the structure as a function of ϵ, W, and the maximum number of distinct keys T (for this example, T=255, since we are only interested for IP addresses of the form x.*.*.*, and there are at most 255 of them). Assume that all integers and references/entries in an array require 4 bytes.

c. Implement the data structure. Use lazy updating, as shown at the exercise session. The structure class should implement the following interface:

```
class JumpingWindow {
    public JumpingWindow(int windowSizeW, double epsilon);
    void insertEvent(int srcIP);
     int getFreqEstimation(int srcIP, int queryWindowSizeW1);
     int getFreqEstimation(int srcIP);
}
```

d. Instantiate the structure with ε = 0.01 and query window size = 10000.

e. Load stream1 using the insertEvent function call.

f. Using the structure, estimate the frequency of the values stored in task1_queries.txt (download from here). The first column corresponds to the time of query arrival, the second to the queried IP and the 3rd to the query window size W1. If W1 is 0 then the query corresponds to the whole W. Save the output in out1.txt (separate the answers with comma). For example, assuming that the answers to the first 3 queries were 10, 5, and 20, then the file would start as follows: "10,5,20"

Deliverables:

- JumpingWindow.java,
- task1.java, which will contain the loading and querying code

- out1.txt
- questions1.pdf, which will contain the answers for questions a. and b.

Sanity check 1: Compute the average/min/max error. Verify that these are within the error bounds. Check if your experiments match the theory of maximum error.

Task 2 (30%). Frequency queries with memory constraints

You are working at an Internet Service Provider that handles a network of at most 400.000 IP addresses. You want to keep an approximate count of the number of packets sent by any IP address from your network. You are given a value Z, which denotes the maximum available memory that you are allowed to use, **in bytes**. Create the necessary data structure by combining any of the sketches seen in the class such that:

i) Any frequency query for an item that never appears in the stream will return 0, with a false positive probability at most pr_1, which is given by the user at construction time.

ii) Any frequency query for an item that appears in the stream will return an estimate for the frequency with a maximum absolute error ε*L, with a probability pr_2. L is the number of items seen so far. ε and pr_2 will be given by the user at construction time.

The structure should fit in the available space. Divide the space among the data structures you choose, such that the accuracy requirements are satisfied. If this is not possible, e.g., not enough space is allowed to satisfy the requirements, then the constructor should throw an exception.

a) Implement the following class:

```
class betterFrequencyEstimator {
        public betterFrequencyEstimator(int availableSpace, float pr1, float epsilon,
                float pr2) throws InsufficientMemoryException;
        void addArrival(int key);
        int getFreqEstimation(int key);
}
```

b) Instantiate the structure with ε = 0.01, pr_1=0.1, pr_2=0.9, and availableSpace=10^7 bytes.

c) Load stream1 using the insertEvent function call.

d) Estimate the frequency of values stored in task2_queries.txt (download from here). The first column corresponds to the time of query arrival, the second to the queried IP. Save the output in out2.txt (separate the answers with comma), similar to task1.

Deliverables

- newSketch.java
- task2.java, which will contain the loading code.
- out2.txt

Task 3 (40%). Range containment queries. You are working at an Internet Service Provider that handles a network of at most 400.000 IP addresses. You are asked to design a structure based on Bloom filters that can answer range containment queries, i.e., given a range [l,r] of

IP addresses from your network, the structure will return true if the packets seen so far contain at least one IP address x, with l <= x <=r, with a false positive probability at most pr. Constraint: You are allowed to execute at most 64 queries on your structure per user query, independently of the length of the query range.

a) Implement the following class:
```
class rangeBF {
        public rangeBF(double pr);
        void insertValue(int key);
        boolean existsInRange(int l, int r);
}
```
b) Instantiate a rangeBF with pr=0.1.
c) Load stream from file1.
d) Estimate the existence of values in the query ranges given in task3_queries.txt (download from here) (1st col: min, 2nd col: max)

Deliverables

- rangeBF.java
- task3.java
- out3.txt

Implementation specifications:

- Do not include test code in your submission; the submission will be timed and the checks will slow down your code.

Out of all the correct implementations, the 3 that will use the least memory will get a 10% bonus on the task.