Design Log Storage System

You are given several logs that each log contains a unique id and timestamp. Timestamp is a string that has the following format: `Year:Month:Day:Hour:Minute:Second`, for example, `2017:01:01:23:59:59`. All domains are zero-padded decimal numbers.

Design a log storage system to implement the following functions:

`void Put(int id, string timestamp)`: Given a log's unique id and timestamp, store the log in your storage system.

`int[] Retrieve(String start, String end, String granularity)`: Return the id of logs whose timestamps are within the range from start to end. Start and end all have the same format as timestamp. However, granularity means the time level for consideration. For example, start = "2017:01:01:23:59:59", end = "2017:01:02:23:59:59", granularity = "Day", it means that we need to find the logs within the range from Jan. 1st 2017 to Jan. 2nd 2017.

Example 1:

```
put(1, "2017:01:01:23:59:59");
put(2, "2017:01:01:22:59:59");
put(3, "2016:01:01:00:00:00");
retrieve("2016:01:01:01:01:01","2017:01:01:23:00:00","Year"); // return [1,2,3], beca
use you need to return all logs within 2016 and 2017.
retrieve("2016:01:01:01:01:01","2017:01:01:23:00:00","Hour"); // return [1,2], becaus
e you need to return all logs start from 2016:01:01:01 to 2017:01:01:23, where log 3
is left outside the range.
```

Note:

1. There will be at most 300 operations of Put or Retrieve.
2. Year ranges from [2000,2017]. Hour ranges from [00,23].
3. Output for Retrieve has no order required.

Solution 1

```python
class LogSystem(object):
    def __init__(self):
        self.logs = []

    def put(self, tid, timestamp):
        self.logs.append((tid, timestamp))

    def retrieve(self, s, e, gra):
        index = {'Year': 5, 'Month': 8, 'Day': 11,
                 'Hour': 14, 'Minute': 17, 'Second': 20}[gra]
        start = s[:index]
        end = e[:index]

        return sorted(tid for tid, timestamp in self.logs
                      if start <= timestamp[:index] <= end)
```

Because the number of operations is very small, we do not need a complicated structure to store the logs: a simple list will do.

Let's focus on the `retrieve` function. For each granularity, we should consider all timestamps to be truncated to that granularity. For example, if the granularity is `'Day'`, we should truncate the timestamp '2017:07:02:08:30:12' to be '2017:07:02'. Now for each log, if the truncated timetuple `cur` is between `start` and `end`, then we should add the id of that log into our answer.

written by awice original link here

Solution 2

```
public class LogSystem {

    List<String[]> timestamps = new LinkedList<>();
    List<String> units = Arrays.asList("Year", "Month", "Day", "Hour", "Minute", "S
econd");
    int[] indices = new int[]{4,7,10,13,16,19};

    public void put(int id, String timestamp) { timestamps.add(new String[]{Integer.
toString(id), timestamp}); }

    public List<Integer> retrieve(String s, String e, String gra) {
        List<Integer> res = new LinkedList<>();
        int idx = indices[units.indexOf(gra)];
        for (String[] timestamp : timestamps) {
            if (timestamp[1].substring(0, idx).compareTo(s.substring(0, idx)) >= 0
&&
                timestamp[1].substring(0, idx).compareTo(e.substring(0, idx)) <= 0)
res.add(Integer.parseInt(timestamp[0]));
        }
        return res;
    }
}
```

written by compton_scatter original link here

Solution 3

```
class LogSystem {
    Map<Integer,String> map = new HashMap<>();

    public  enum Index {
        Year(4), Month(7), Day(10), Hour(13), Minute(16), Second(19);
        int idx;   Index(int i) {this.idx = i;}
    }

    public void put(int id, String timestamp) { map.put(id,timestamp); }

    public List<Integer> retrieve(final String s, final String e, String gra) {
        Function<String, String> fn = str -> str.substring(0, Index.valueOf(gra).id
x);
        return map.entrySet().stream().filter(
                x -> fn.apply(x.getValue()).compareTo(fn.apply(s)) >= 0 && fn.apply
(x.getValue()).compareTo(fn.apply(e)) <= 0)
                .collect(mapping(v -> v.getKey(), toList()));
    }
}
```

written by johnyrufus16 original link here