635. Design Log Storage

July 1, 2017 | 15.7K views

Average Rating: 3.67 (15 votes)

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], becau
retrieve("2016:01:01:01:01:01","2017:01:01:23:00:00","Hour"); // return [1,2], because
```

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

Approach #1 Converting timestamp into a number [Accepted]

This solution is based on converting the given timestap into a number. This can help because retrieval of Logs lying within a current range can be very easily done if the range to be used can be represented in the form of a single number. Let's look at the individual implementations directly.

1. `put`: Firstly, we split the given timestamp based on `:` and store the individual components obtained into an st array. Now, in order to put this log's entry into the storage, firstly, we convert this timestamp, now available as individual components in the st array into a single number. To obtain a number which is unique for each timestamp, the number chosen is such that it represents the timestamp in terms of seconds. But, doing so for the Year values can lead to very large numbers, which could lead to a potential overflow. Since, we know that the Year's value can start from 2000 only, we subtract 1999 from the Year's value before doing the conversion of the given timestamp into seconds. We store this timestamp(in the form of a number now), along with the Log's id, in a $list$, which stores data in the form $[converted\_timestamp, id]$.

2. `retrieve`: In order to retrieve the logs' ids lying within the timestamp s and e, with a granularity gra, firstly, we need to convert the given timestamps into seconds. But, since, we need to take care of granularity, before doing the conversion, we need to consider the effect of granularity. Granularity, in a way, depicts the precision of the results. Thus, for a granularity corresponding to a Day, we need to consider the portion of the timestamp considering the precision upto Day only. The rest of the fields can be assumed to be all 0's. After doing this for both the start and end timestamp, we do the conversion of the timestamp with the required precision into seconds. Once this has been done, we iterate over all the logs in the $list$ to obtain the ids of those logs which lie within the required range. We keep on adding these ids to a res list which is returned at the end of this function call.

Java

```
public LogSystem() {
    list = new ArrayList < long[] > ();
}

public void put(int id, String timestamp) {
    int[] st = Arrays.stream(timestamp.split(":")).mapToInt(Integer::parseInt).toArray();
    list.add(new long[] {convert(st), id});
}
public long convert(int[] st) {
    st[1] = st[1] - (st[1] == 0 ? 0 : 1);
    st[2] = st[2] - (st[2] == 0 ? 0 : 1);
    return (st[0] - 1999L) * (31 * 12) * 24 * 60 * 60 + st[1] * 31 * 24 * 60 * 60 + st[2] * 24 * 60 *
60 + st[3] * 60 * 60 + st[4] * 60 + st[5];
}
public List < Integer > retrieve(String s, String e, String gra) {
    ArrayList < Integer > res = new ArrayList();
    long start = granularity(s, gra, false);
    long end = granularity(e, gra, true);
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i)[0] >= start && list.get(i)[0] < end)
            res.add((int) list.get(i)[1]);
    }
    return res;
}

public long granularity(String s, String gra, boolean end) {
    HashMap < String, Integer > h = new HashMap();
    h.put("Year", 0);
```

Performance Analysis

- The `put` method takes $O(1)$ time to insert a new entry into the given set of logs.
- The `retrieve` method takes $O(n)$ time to retrieve the logs in the required range. Determining the granularity takes $O(1)$ time. But, to find the logs lying in the required range, we need to iterate over the set of logs atleast once. Here, n refers to the number of entries in the current set of logs.

Approach #2 Better Retrieval [Accepted]

This method remains almost the same as the last approach, except that, in order to store the timestamp data, we make use of a TreeMap instead of a list, with the key values being the timestamps converted in seconds form and the values being the ids of the corresponding logs.

Thus, the `put` method remains the same as the last approach. However, we can save some time in `retrieve` approach by making use of `tailMap` function of TreeMap, which stores the entries in the form of a sorted navigable binary tree. By making use of this function, instead of iterating over all the timestamps in the storage to find the timestamps lying within the required range(after considering the granularity as in the last approach), we can reduce the search space to only those elements whose timestamp is larger than(or equal to) the starting timestamp value.

Java

```
public class LogSystem {
    TreeMap < Long, Integer > map;
    public LogSystem() {
        map = new TreeMap < Long, Integer > ();
    }

    public void put(int id, String timestamp) {
        int[] st = Arrays.stream(timestamp.split(":")).mapToInt(Integer::parseInt).toArray();
        map.put(convert(st), id);
    }
    public long convert(int[] st) {
        st[1] = st[1] - (st[1] == 0 ? 0 : 1);
        st[2] = st[2] - (st[2] == 0 ? 0 : 1);
        return (st[0] - 1999L) * (31 * 12) * 24 * 60 * 60 + st[1] * 31 * 24 * 60 * 60 + st[2] * 24 * 60 *
60 + st[3] * 60 * 60 + st[4] * 60 + st[5];
    }
    public List < Integer > retrieve(String s, String e, String gra) {
        ArrayList < Integer > res = new ArrayList();
        long start = granularity(s, gra, false);
        long end = granularity(e, gra, true);
        for (long key: map.tailMap(start).keySet()) {
            if (key >= start && key < end)
                res.add(map.get(key));
        }
        return res;
    }

    public long granularity(String s, String gra, boolean end) {
```

Performance Analysis

- The TreeMap is maintained internally as a Red-Black(balanced) tree. Thus, the `put` method takes $O(log(n))$ time to insert a new entry into the given set of logs. Here, n refers to the number of entries currently present in the given set of logs.
- The `retrieve` method takes $O(m_{start})$ time to retrieve the logs in the required range. Determining the granularity takes $O(1)$ time. To find the logs in the required range, we only need to iterate over those elements which already lie in the required range. Here, m_{start} refers to the number of entries in the current set of logs which have a timestamp greater than the current $start$ value.

Analysis written by: @vinod23

Rate this article:

Previous Next

Comments: 6

Sort By

Type comment here... (Markdown is supported)

Preview Post

saikrishnatvt ★12 May 2, 2019 9:02 PM Report

What is the need to convert the time stamp into seconds?
Why cant we straight away use it? For instance, why can't we represent the timestamp 2019:03:04:15:16:18 as 20190304151618 (or 190304151618)?

12 Share Reply

llalies ★13 July 6, 2018 7:35 AM

Can you please explain the following lines?

```
st[1] = st[1] - (st[1] == 0 ? 0 : 1);
st[2] = st[2] - (st[2] == 0 ? 0 : 1);
```

12 Share Reply

SHOW 2 REPLIES

Explorer123 ★6 September 20, 2018 11:19 AM

how about using subMap function TreeMap? In that case, the complexity will be reduced to O(logN + k), where k is the number of elements within range and N is the total number of elements in treeMap

6 Share Reply

whyseahike ★4 July 6, 2017 12:31 AM

2017:01:31:23:59:59 can also represented as 180131235959, this way can save lots of calculation at put() /retrieval() where only year needs a subtract.

1 Share Reply

SHOW 3 REPLIES

mzchen ★3771 July 5, 2017 10:18 AM

The code is wrong for case:
["LogSystem","put","put","put","retrieve"]
[[],[1,"2017:01:31:23:59:59"],[2,"2017:02:01:23:59:59"],[3,"2017:01:31:00:00:00"],
["2017:01:30:00:00:00","2017:02:01:00:00:00","Hour"]]

Read More

1 Share Reply

vinod23 ★425 July 5, 2017 10:02 PM Report

@mzchen It is assumed that every month contains max of 30 days. Anyways I have changed the formula lin convert function Now the code works fine for your test case also. Thanks.

-2 Share Reply