Design Twitter

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets in the user's news feed. Your design should support the following methods:

1. **postTweet(userId, tweetId)**: Compose a new tweet.
2. **getNewsFeed(userId)**: Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent.
3. **follow(followerId, followeeId)**: Follower follows a followee.
4. **unfollow(followerId, followeeId)**: Follower unfollows a followee.

Example:

```
Twitter twitter = new Twitter();

// User 1 posts a new tweet (id = 5).
twitter.postTweet(1, 5);

// User 1's news feed should return a list with 1 tweet id -> [5].
twitter.getNewsFeed(1);

// User 1 follows user 2.
twitter.follow(1, 2);

// User 2 posts a new tweet (id = 6).
twitter.postTweet(2, 6);

// User 1's news feed should return a list with 2 tweet ids -> [6, 5].
// Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5.
twitter.getNewsFeed(1);

// User 1 unfollows user 2.
twitter.unfollow(1, 2);

// User 1's news feed should return a list with 1 tweet id -> [5],
// since user 1 is no longer following user 2.
twitter.getNewsFeed(1);
```

Solution 1

```
class Twitter(object):

    def __init__(self):
        self.timer = itertools.count(step=-1)
        self.tweets = collections.defaultdict(collections.deque)
        self.followees = collections.defaultdict(set)

    def postTweet(self, userId, tweetId):
        self.tweets[userId].appendleft((next(self.timer), tweetId))

    def getNewsFeed(self, userId):
        tweets = heapq.merge(*(self.tweets[u] for u in self.followees[userId] | {
userId}))
        return [t for _, t in itertools.islice(tweets, 10)]

    def follow(self, followerId, followeeId):
        self.followees[followerId].add(followeeId)

    def unfollow(self, followerId, followeeId):
        self.followees[followerId].discard(followeeId)
```

written by StefanPochmann original link here

Solution 2

```
private static class Tweet {
    int timestamp;
    int tweetId;

    public Tweet(int tweetId, int timestamp) {
        this.tweetId = tweetId;
        this.timestamp = timestamp;
    }
}

private Map<Integer, Set<Integer>> followMap = new HashMap<Integer, Set<Integer>>
();
private Map<Integer, List<Tweet>> tweetMap = new HashMap<Integer, List<Tweet>>();

private AtomicInteger timestamp;

/** Initialize your data structure here. */
public Twitter() {
    timestamp = new AtomicInteger(0);
}

/** Compose a new tweet. */
public void postTweet(int userId, int tweetId) {
    Tweet newTweet = new Tweet(tweetId, timestamp.getAndIncrement());

    if (!tweetMap.containsKey(userId)) {
        tweetMap.put(userId, new ArrayList<Tweet>());  //Assuming no deletion for
now?
    }

    tweetMap.get(userId).add(newTweet);
}

/**
 * Retrieve the 10 most recent tweet ids in the user's news feed. Each item
 * in the news feed must be posted by users who the user followed or by the
 * user herself. Tweets must be ordered from most recent to least recent.
 */
public List<Integer> getNewsFeed(int userId) {
    List<Integer> result = new ArrayList<Integer>(10);

    PriorityQueue<int[]> pq = new PriorityQueue<int[]>(new Comparator<int[]>() {
        public int compare(int[] it1, int[] it2) {
            return tweetMap.get(it2[0]).get(it2[1]).timestamp - tweetMap.get(it1[
0]).get(it1[1]).timestamp;
        }
    });

    Set<Integer> followeeSet = new HashSet<Integer>();
    followeeSet.add(userId);
    if (followMap.containsKey(userId)) {
        followeeSet.addAll(followMap.get(userId));
    }
```

```
    for (Integer followee : followeeSet) {
        if (tweetMap.containsKey(followee)) {
            List<Tweet> tweetList = tweetMap.get(followee);
            if (tweetList.size() > 0) {
                pq.add(new int[] { followee, tweetList.size() - 1 });
            }
        }
    }

    while (result.size() < 10 && pq.size() > 0) {
        int[] it = pq.poll();

        result.add(tweetMap.get(it[0]).get(it[1]).tweetId);
        it[1]--;
        if (it[1] >= 0) {
            pq.add(it);
        }
    }

    return result;
}

/**
 * Follower follows a followee. If the operation is invalid, it should be a
 * no-op.
 */
public void follow(int followerId, int followeeId) {
    Set<Integer> followSet = followMap.get(followerId);
    if (followSet == null) {
        followSet = new HashSet<Integer>();
        followMap.put(followerId, followSet);
    }

    followSet.add(followeeId);
}

/**
 * Follower unfollows a followee. If the operation is invalid, it should be
 * a no-op.
 */
public void unfollow(int followerId, int followeeId) {
    Set<Integer> followSet = followMap.get(followerId);
    if (followSet == null) {
        followSet = new HashSet<Integer>();
        followMap.put(followerId, followSet);
    }

    followSet.remove(followeeId);
}
```

written by lop original link here