

355. Design Twitter

Medium 1239 232 Add to List Share

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);
```

Tweet

id	5
time	7
next	→ null

User

id	2
followed	→ {User 5, User 3}
head	→ Tweet 5 → Tweet 1 → null

Priority queue.

```

1  class Tweet {
2  private:
3      int id;
4      int time;
5
6  public:
7      Tweet *next;
8
9      Tweet(int id, int time) {
10         this->id = id;
11         this->time = time;
12         this->next = nullptr;
13     }
14
15     int getTime() {
16         return this->time;
17     }
18
19     int getId() {
20         return this->id;
21     }
22
23 };
24
25 struct cmp{
26     bool operator()(Tweet *a, Tweet *b){
27         return (b->getTime() - a->getTime()) > 0;
28     }
29 };
30
31 class User {
32 private:
33     int id;
34
35 public:
36     unordered_set<int> *followed;
37     Tweet *head;
38
39     User(int userId) {
40         this->followed = new unordered_set<int>();
41         this->id = userId;
42         this->head = nullptr;
43         // follow self
44         follow(this->id);
45     }
46
47     void follow(int userId) {
48         this->followed->insert(userId);
49     }
50
51     void unfollow(int userId) {
52         // cant unfollow self
53         if (userId != this->id) {
54             this->followed->erase(userId);
55         }
56     }
57
58     void post(int tweetId, int &timestamp) {
59         Tweet *tweet = new Tweet(tweetId, timestamp);
60         timestamp++;
61         // insert tweet in time order
62         tweet->next = this->head;
63         this->head = tweet;
64     }
65 };
66

```



```

66 class Twitter {
67 private:
68     int timestamp;
69     unordered_map<int, User*> *userMap;
70
71 public:
72     /** Initialize your data structure here. */
73     Twitter() {
74         timestamp = 0;
75         userMap = new unordered_map<int, User*>();
76     }
77
78     /** Compose a new tweet. */
79     void postTweet(int userId, int tweetId) {
80         if (!this->userMap->count(userId)) {
81             this->userMap->insert(pair<int, User*>(userId, new User(userId)));
82         }
83         this->userMap->at(userId)->post(tweetId, this->timestamp);
84     }
85
86     /** Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by
87     users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent. */
88     vector<int> getNewsFeed(int userId) {
89         auto *res = new vector<int>();
90         if (!this->userMap->count(userId)) {
91             return *res;
92         }
93
94         unordered_set<int> *users = this->userMap->at(userId)->followed;
95
96         auto *pq = new priority_queue<Tweet*, vector<Tweet*>, cmp>();
97
98         for (auto user : *users) {
99             Tweet *tweet = this->userMap->at(user)->head;
100             if (tweet == nullptr) {
101                 continue;
102             }
103             pq->push(tweet);
104         }
105
106         while (!pq->empty()) {
107             if (res->size() == 10) {
108                 break;
109             }
110             Tweet *tweet = pq->top();
111             pq->pop();
112             res->push_back(tweet->getId());
113             if (tweet->next != nullptr) {
114                 pq->push(tweet->next);
115             }
116         }
117
118         return *res;
119     }
120
121     /** Follower follows a followee. If the operation is invalid, it should be a no-op. */
122     void follow(int followerId, int followeeId) {
123         if (!this->userMap->count(followerId)) {
124             this->userMap->insert(pair<int, User*>(followerId, new User(followerId)));
125         }
126         if (!this->userMap->count(followeeId)) {
127             this->userMap->insert(pair<int, User*>(followeeId, new User(followeeId)));
128         }
129         this->userMap->at(followerId)->follow(followeeId);
130     }
131 }

```

```
131
132 ▾ /** Follower unfollows a followee. If the operation is invalid, it should be a no-op. */
133 ▾ void unfollow(int followerId, int followeeId) {
134 ▾     if (!this->userMap->count(followerId)) {
135         this->userMap->insert(pair<int, User*>(followerId, new User(followerId)));
136     }
137 ▾     if (!this->userMap->count(followeeId)) {
138         this->userMap->insert(pair<int, User*>(followeeId, new User(followeeId)));
139     }
140     this->userMap->at(followerId)->unfollow(followeeId);
141 }
142 };
143
```