设计一个Twitter

分析： 对获取当前10个最新的twtterId本质上是一个k个有序链表合并。

```
struct Tweet {
  Tweet(int tweetId, int timeId)
      : tweetId(tweetId), timeId(timeId), next(nullptr) {
  }

  int    timeId;
  int    tweetId;
  Tweet *next;
};

class User {
public:
  User(int userId) : userId(userId), head(nullptr) {
    follow(userId);  // 关注他自己
  }

  void post(Tweet *tw) {
    // 放入一个tweet
    if (head == nullptr) {
      head = tw;
      return;
    }
    tw->next = head;
    head     = tw;
  }

  void follow(int userid) {
    // 关注一个用户
    followee.insert(userid);
  }

  void unfllow(int userid) {
    // 取关一个用户
    if (userid != this->userId) {
      followee.erase(userid);
    }
  }

  // 返回关注列表
  std::set<int> getFollow() const {
    return followee;
  }

  Tweet *getTweetList() const {
    return head;
  }

private:
```

```
    int           userId;
    std::set<int> followee;  // 该用户的关注列表
    Tweet *       head;      // 最新的一条tweet
  };

  class Twitter {
  public:
    Twitter() {
    }

    void postTweet(int userId, int tweetId) {
      if (!map.count(userId)) {
        map.insert(std::make_pair(userId, new User(userId)));
      }
      Tweet *tw = new Tweet(tweetId, timeId++);
      map[userId]->post(tw);
    }

    std::vector<int> getNewsFeed(int userId) {
      std::vector<int> res;
      if (!map.count(userId)) {
        return res;
      }

      std::priority_queue<Tweet *, std::vector<Tweet *>, less> pq;  // 小根堆
      std::set<int> s = map[userId]->getFollow();

      for (auto item : s) {
        Tweet *tw = map[item]->getTweetList();
        if (tw != nullptr) {
          pq.push(tw);
        }
      }

      while (!pq.empty()) {
        if (res.size() == 10) {
          break;
        }
        Tweet *tw = pq.top();
        pq.pop();

        res.push_back(tw->tweetId);

        if (tw->next) {
          pq.push(tw->next);
        }
      }

      return res;
    }

    void follow(int followerId, int followeeId) {
      if (!map.count(followerId)) {
        User *user = new User(followerId);
```

```
      map.insert(std::make_pair(followerId, user));
    }

    if (!map.count(followeeId)) {
      User *user = new User(followeeId);
      map.insert(std::make_pair(followeeId, user));
    }

    map[followerId]->follow(followeeId);
  }

  void unfollow(int followerId, int followeeId) {
    if (map.count(followerId)) {
      map[followerId]->unfllow(followeeId);
    }
  }

  void print() {
    for (auto it : map) {
      std::cout << "user " << it.first << " : ";
      for (auto item : it.second->getFollow()) {
        std::cout << item << ' ';
      }
      std::cout << '\n';
    }
  }

private:
  struct less {
    bool operator()(const Tweet *t1, const Tweet *t2) const {
      return t1->timeId < t2->timeId;
    }
  };

  std::unordered_map<int, User *> map;  // userid 与 User的对应关系
  static int                      timeId;
};

int Twitter::timeId = 1;
```