6. Office Hours and STL Data Structures (18 pts.)

Suppose that each TA has their own **separate** queue that a student could join to wait for help. They would indicate the name of the TA and their own name and the system would add them to the appropriate queue. However, we do **NOT want to allow students to be in more than one TA queue at a time** and thus "game" the system. Furthermore, we should allow students to leave the queue early if they solve their problem while waiting. Finally, we will provide the ability for TAs to indicate they are done helping the student at the front of their queue and start helping the next waiting student, if any.

Implement the following `OfficeHours` class to meet the description and requirements described above and below:

- You may assume TAs and students all have unique names.
- You may use `map`, `set`, `queue`, `stack`, `singly` as defined at the start of the exam (pg 2).
- Ensure you meet the runtime requirements described in the comments

```
typedef std::queue<std::string> StudentQueue;
class OfficeHours {
public:
    OfficeHours();
    // Open a new queue for the given TA. Do nothing if TA already exists.
    // Runtime: log(t) where t = number of TAs
      void add_ta_on_duty(std::string ta);

    // If the student is not already in some TA's queue,
    //  adds the student to the given TA's queue and returns true.
    //  Returns false otherwise or if the TA is not on duty.
    // Runtime: log(t)+log(s) where t = # of TAs, s = # of students
      bool join_ta_hours(std::string ta, std::string student);

    // Assume this function is completed for you.
      void ta_done_with_stu(std::string ta);

    // Remove the given student from the given TA's queue
    // No runtime requirement but should not be overly inefficient
      void leave_queue_early(std::string ta, std::string student);
private:
    // Add your data members here
    // Answers may vary
    // Map of TA => Queue of students
    std::map<std::string, StudentQueue> taqueues;
    // Set of students in any (and all) queues
    std::set<std::string> students;
};
```

```
OfficeHours::OfficeHours()   // can be blank if no initialization necessary
{
  // no code necessary for solution data members
}
void OfficeHours::add_ta_on_duty(std::string ta) {
    if(taqueues.find(ta) == taqueues.end()){ // If TA not in map
        StudentQueue newQ;
        taqueues.insert(make_pair(ta, newQ));
    }
    // NOTE: BECAUSE C++ map::insert does not replace the
    // value if the key is already in the map, then the
    // if statement is not needed. Calling insert would be safe
    // even if the TA was present.
}
bool OfficeHours::join_ta_hours(std::string ta, std::string student) {
    if(students.find(student) == students.end())
    {
        if(taqueues.find(ta) != taqueues.end())
        {
            students.insert(student);
            taqueues[ta].push(student);
            return true;
        }
    }
    return false;
}
void OfficeHours::ta_done_with_stu(std::string ta) { /* Completed for you */ }

// Precondition:  ta is guaranteed to be added and the student is in the TAs queue
void OfficeHours::leave_queue_early(std::string ta, std::string student)
{
    StudentQueue temp;
    while(!taqueues[ta].empty())
    {
        // student is not the one leaving
        if(taqueues[ta].front() != student)
        {
            // save that student to be reinserted
            temp.push(taqueues[ta].front());
        }
        taqueues[ta].pop();
    }
    taqueues[ta] = temp;
    students.erase(student);  // student is no longer in any queue
}
```