Identify the order of CS courses

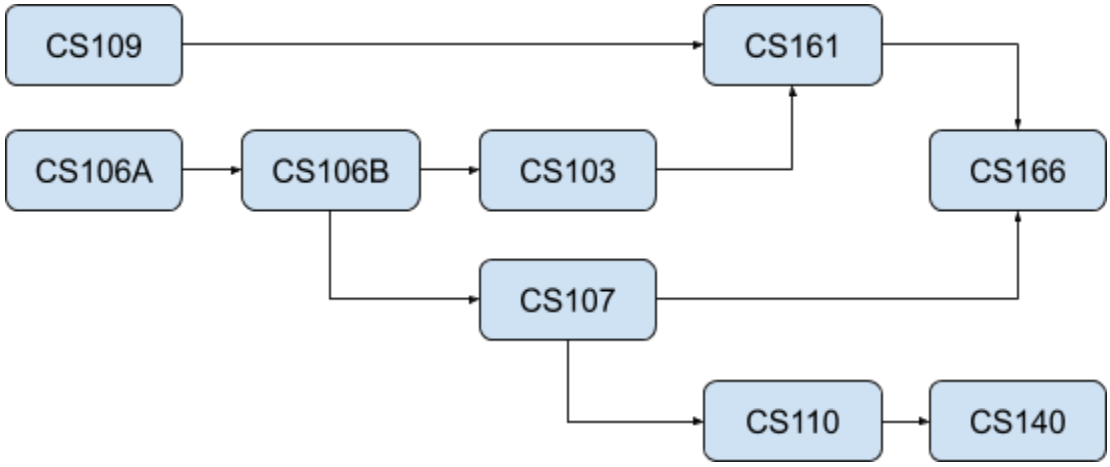
Author: Dima Timofeev (thedima@stanford.edu)

Problem description

Suppose you just started your journey in Computer Science as a Stanford Computer Science department student. Stanford Engineering School offers a lot of classes, and some of them have some prerequisites. It is tough to identify in which order you should take them to satisfy all requirements. For example, you have a list of courses you want to complete and their prerequisites:

Course	**Prerequisites**
CS109: Probability for Computer Scientists	
CS106A: Programming Methodologies	
CS106B: Programming Abstractions	CS106A
CS103: Mathematical Foundations of Computing	CS106B
CS107: Computer Organization and Systems	CS106B
CS110: Principles of Computer Systems	CS107
CS161: Design and Analysis of Algorithms	CS109, CS103
CS166: Data Structures	CS107, CS161
CS140: Operating Systems	CS110

Once you decided what you want to attend, you carefully crafted the table of those courses and its dependencies. You need to identify the order in which you are going to take them. For example, if you're going to take "CS161: Design and Analysis of Algorithms", you first need to choose "CS 109: Probability for Computer Scientists" and "CS103: Mathematical Foundations of Computing". However, CS103 has its dependency: "CS103: Mathematical Foundations of Computing," and so on. You can think about it as a directed acyclic graph (DAG):



Where one of the correct sequences of taking them is: *CS109 → CS106A → CS016B → CS103 → CS107 → CS110 → CS161 → CS166 → CS140*.

Write a function

```
Vector<string> SequenceOfCourses(Map<string, Set<string>>&);
```

that accepts as input a **`Map<string, Set<string>>&`** that has courses as keys are associated with the course prerequisites and then returns a **`Vector<string>`** holding courses in a sequence you can take them and don't miss any prerequisites.

Your function should also fulfill the following requirements:

- If there are multiple sequences - return any of them
- If a course has no prerequisites and is not a prerequisite for any other, it is OK to put it either in the beginning or end of the resulting vector

```
Vector<string> SequenceOfCourses(Map<string, Set<string>>&) {
      /* TODO: Fill me in! */
      return {};
}
```

Solutions

Solution 1

```
void SequenceOfCoursesHelper(
       Map<string, Set<string>>& courses,
       string& course, Set<string>& explored,
       int& currentTime, Map<string, int>& visitTime) {
   explored += course;
```

```
    for (string dep : courses[course]) {
        if (!explored.contains(course)) {
            SequenceOfCoursesHelper(
                courses, dep, explored, currentTime, visitTime);
        }
    }
    visitTime[course] = currentTime;
    currentTime++;
}

Vector<string> SequenceOfCourses(Map<string, Set<string>>& courses) {
    Set<string> explored;
    Map<string, int> visitTime;
    int currentTime = 0;
    for (string course : courses) {
        if (!explored.contains(course)) {
            SequenceOfCoursesHelper(
                courses, course, explored, currentTime, visitTime);
        }
    }

    Vector<string> order(visitTime.size(), "");
    for (string course : visitTime) {
        order[visitTime[course]] = course;
    }
    return order;
}
```

Solution 1 takes advantage of a helper function to satisfy the interface described in the problem description. **SequenceOfCourses** traverses all vertices. This works for any starting vertex. **SequenceOfCourses** and **SequenceOfCoursesHelper** receive parameters by reference: they help maintain the state between recursive calls and avoid unnecessary copying. The solution to this problem is known as Topological sorting. Let *n* be the number of vertices and *m* - the number of edges. The running time of this solution is *O(m+n)*.

Solution 2

```
int SequenceOfCoursesHelper(
        Map<string, Set<string>>& courses,
        string& course, Set<string>& explored,
        int currentTime, Map<string, int>& visitTime) {
    explored += course;
```

```
    for (string dep : courses[course]) {
        if (!explored.contains(course)) {
            currentTime = SequenceOfCoursesHelper(
                courses, dep, explored, currentTime, visitTime);
        }
    }
    visitTime[course] = currentTime;
    currentTime++;
    return currentTime;
}

Vector<string> SequenceOfCourses(Map<string, Set<string>>& courses) {
    Set<string> explored;
    Map<string, int> visitTime;
    int currentTime = 0;
    for (string course : courses) {
        if (!explored.contains(course)) {
            currentTime = SequenceOfCoursesHelper(
                courses, course, explored, currentTime, visitTime);
        }
    }

    Vector<string> order(visitTime.size(), "");
    for (string course : visitTime) {
        order[visitTime[course]] = course;
    }
    return order;
}
```

Solution 2 is very similar to solution 1, except it does not pass the current time (*currentTime*) by reference but pass it by value instead. Solution 2 is a more common practice in recursive algorithms descriptions. In many popular programming languages (Java, Python, JavaScript, etc.), primitive types pass by value. There is no control in those languages to pass by reference. The running time is the same as in solution 1: *O(m+n)*.

Problem motivation

Concept coverage

This question is designed to get students to extend their knowledge and understanding of graphs representation and how graphs can be modeled and traversed with ADTs (here, Map, Set, and Vector.) The problem requires an understanding of:

- How to iterate over maps
- How recursion works and how to combine recursion with iterations
- The role of helper functions in recursive algorithms
- Passing data structures by reference vs. returning them
- The ability to manipulate with sets and maps
- The ability to select proper ADT for the best running time

Modeling graphs with ADTs and making recursive calls in the loop at the same time can be complicated. The problem specifies public interface: the entry point is not given. It adds a level of complexity but, at the same time, provides an implicit hint to think about helper function. ADTs have to be carefully selected. If they are not, the running time can jump to non-linear. Examples are: track explored vertices with Vector instead of Set, sort output vector with any $O(n^2)$ algorithm, etc.

Personal significance

From a personal perspective, the most complicated part of the class for me is recursion. I struggled to imagine how the base case will be met and how it helps build a solution specifically when recursive calls happen in a loop. So, I used the opportunity to develop my understanding of how recursive functions can traverse graphs, how to model graphs with ADTs, and how to store some useful information on the way. It was interesting to think about different programming languages with less control over references and algorithm implementation.

I have heard about ways to resolve dependencies on DAG, but have never tried to design an effective algorithm myself. My daily job includes updating a multi-layer distributed system in a proper sequence. It is excellent to understand how updater internals work.

Common misconceptions

Misconceptions or bugs could come in one of three main areas in this problem:

- Bad choice of ADTs
- Passing data structures by reference vs. by value
- Building up the resulting vector of courses

It is essential to select proper ADTs to achieve *O(m+n) overall running time*. The solution tracks explored vertices in a set. The critical property of sets we utilize here is the running time of lookup: *O(1)*. A student might try to use a vector for this. In this case, the lookup will take *O(n)* instead of *O(1)*.

The description of a problem explicitly takes an input map by reference. However, a student can forget to pass the map into the helper function by reference. The algorithm will work and

produce the correct result because it does not change the state of the map. However, it will make many copies and for big graphs can eat all allocated memory pretty fast.
If a student selects to pass the current time by the value, they can forget to return updated time from the function. Very often, recursive algorithms crash programs when something went wrong, for example, stack overflow. It might take a lot of time to find out why it happens.

Once a student is assigned a sorting order to courses, they need to output them based on sorting order numbers. There are many ways to do so, but any solution slower than *O(n)* can blow up the overall *O(m+n)* running time. The student will have to elaborate on the sorting algorithm running time and topological sorting based on *m* and *n*. Hence can lose the efficiency of depth-first search.

Appendix 1: Input data

```
Map<string, Set<string>> courses;
courses["CS109: Probability for Computer Scientists"] = {};
courses["CS106A: Programming Methodologies"] = {};
courses["CS106B: Programming Abstractions"] = {
    "CS106A: Programming Methodologies"};
courses["CS103: Mathematical Foundations of Computing"] = {
    "CS106B: Programming Abstractions"};
courses["CS107: Computer Organization and Systems"] = {
    "CS106B: Programming Abstractions"};
courses["CS110: Principles of Computer Systems"] = {
    "CS107: Computer Organization and Systems"};
courses["CS161: Design and Analysis of Algorithms"] = {
    "CS109: Probability for Computer Scientists",
    "CS103: Mathematical Foundations of Computing"};
courses["CS166: Data Structures"] = {
    "CS107: Computer Organization and Systems",
    "CS161: Design and Analysis of Algorithms"};
courses["CS140: Operating Systems"] = {
    "CS110: Principles of Computer Systems"};
```