CSCI 3901 Assignment 3

Fall 2023

Test Cases

Course Prerequisites:

Type of the test case	Property of the test case	Property of the input data	Expected result
Input validation	Null check	CourseID is null	Return false
	Empty check	CourseID is empty string	Return false
	Type mismatch	CourseID is not of string type	Return false
	Null Check	prerequisiteID is null	Return false
	Empty Check	prerequisiteID is empty	Return false
	Null check	demand is null	Return false
	Not integer check	Demand is not of type integer	Return false
Boundary case	Max demand	Demand is more than 100	Return false
	Min demand	Demand is less than 0 (negative value)	Return false
	Min courses	Total courses in the curriculum are less than 2	Return PrerequisiteCycleException
Control flow	Course not found	CourseID is not in the system	Return CourseNotFoundException
	Course not found	prerequisiteID not in the system	Return CourseNotFoundException
	Cycle	CourseID is same as prerequisiteID	Return PrerequisiteCycleException
Data flow	Prerequisite not scheduled	prerequisiteID is not yet scheduled (do not have the capacity) i.e., calling coursePrerequisite without calling scheduleCourse method	Return PrerequisiteNotScheduledYetException
	No Courses in	Not adding any	Return CourseNotFoundException

	the curriculum	courses in Curriculum object i.e., calling coursePrerequisite without calling addCourse method	

Schedule Course:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Input validation	Null check	Passing courseID as null	Return false
	Empty check	Passing courseID as empty string	Return false
	Type mismatch	Passing courseID anything other than string	Return false
	Non integer value	Passing non integer value for capacity	Return false
Boundary case	Min capacity	Passing capacity less than or equal to 0	Return false
Control flow	Course not found	CourseID is not in the system	Return CourseNotFoundException
	Course already scheduled	Trying to schedule the course for the second type in the same academic year	Return CourseAlreadyScheduledException
Data flow	No Courses in the curriculum	Not adding any courses in Curriculum object i.e., calling scheduleCourse without calling addCourse method	Return CourseNotFoundException

Has Prerequisite Cycle:

Type of the test case	Property of the test case	Property of the input data	Expected result
Boundary case	Single cycle	There is only one cycle in the curriculum	Return true
	Multiple cycles	Multiple independent cycles in the curriculum	Return true
Control flow	Cycle in curriculum	A course has a prerequisite such that that prerequisite's prerequisite is that course. A - B B - A	Return true
	No prerequisites	All courses with no prerequisites	Return false
	A complete graph	Every vertex is connected to every other vertex	Return true
Data flow	No Courses in the curriculum	Not adding any courses in Curriculum object i.e., calling hasPrerequisiteCycle before addCourse, scheduleCourse and coursePrerequisite	Return CourseNotFoundException

Bottleneck courses:

Type of the test case	Property of the test case	Property of the input data	Expected result
Input validation	Negative value	Course with negative demand	Return empty set
	Negative value	Course with negative capacity	Return PrerequisiteNotScheduledYetException
Boundary Cases	Max value	Multiple courses with high demand and capacity, including cases where some are bottlenecks and others are not.	Return set of courses which are bottleneck

Control flow	Demand and capacity	Demand of a course is greater than capacity	Return that course as bottleneck
	Demand and capacity	No courses in curriculum have demand greater than capacity respectively	Return empty set
	Demand and capacity	Course with same demand and capacity	Return empty set
Data flow	Flow of the function call	Calling bottleneck method before scheduling the courses i.e., before scheduleCourse method	Return PrerequisiteNotScheduledYetException

Course Path:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Input validation	Empty check	endCourseID is empty	Return null
	Null check	endCourseID is null	Return null
	Cycle	There is a cycle in the curriculum	Return PrerequisiteCycleException
	Bottleneck	There is a bottleneck course in the system	Return BottleneckCourseException
Boundary case	Single course	endCourseID has only one course	Return path of all courses which leads to target course including the target course
	Multiple courses	endCourseID has multiple courses	Return path of all courses which leads to all target courses including the target courses
Control flow	A course with no prerequisite	courseID is the course with no prerequisite	Return empty list
	A course which is not a part of curriculum	courseID is the course not in curriculum	Return CourseNotFoundException
	Indirect prerequisites	courseID is the course which have prerequisites which further have their own prerequisites	Return List of all the courses which lead to target course/courses.

Data flow	Course not scheduled	courseID is a course which is not scheduled yet. i.e., calling coursePath before scheduleCourse	Return PrerequisiteNotScheduledYetException
	Course is yet to have prerequisite	courseID is course which is yet to have its prerequisites i.e., calling coursePath before coursePrerequisite	Return list with the target course itself

Equivalent Courses:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Input Validation	Cycle	There is a cycle in the curriculum	Return PrerequisiteCycleException
	Bottleneck	There is a bottleneck course in the system	Return BottleneckCourseException
Boundary case	Single course	Curriculum with only one course	Return empty set
	Multiple courses	Curriculum with multiple courses	Return set of interchangeable courses set if interchangeable is true else return set of all equivalent courses set
Control flow	Interchangeable true	When interchangeable is true and if courses appear together as prerequisites for multiple courses and transferring the material/content of one course into another won't affect the curriculum	Return set of all those courses set
	Interchangeable false	When interchangeable is false and if course appear together as prerequisites in multiple courses but here transferring content of one to	Return set of all those courses set

		another can affect curriculum	
Data flow	Courses don't have prerequisites yet	Calling equivalentCourses before coursePrerequisite method	Return empty set

Longest Chain:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Input Validation	Cycle	There is a cycle in the curriculum	Return PrerequisiteCycleException
	Bottleneck	There is a bottleneck course in the system	Return BottleneckCourseException
Boundary Case	Single course	Only one course in the curriculum	Return Set with one list with one item (that particular course)
Control Flow	Single chain	A single longest chain of courses exists in the curriculum	Return Set with one list
	Multiple chain	Multiple chains with same length exist in the curriculum	Return Set with all possible longest chains list
Data flow	Courses don't have prerequisite yet	Calling longest chain method before coursePrerequisite	Return Set with all the list (one single course) of all the courses as they all have the same length initially

External Documentation

Overview

The program is a collection of classes that we've put together to perform a process which includes following step:

1. Add the courses into the curriculum. Each course will have a unique course ID and a course name. The course will also have more information such as course capacity (number of students it can accommodate) and demand value (number of students interested in this course based on the prerequisite of the course).
2. Schedule the course. Here, we register for the course and set the capacity of the course.
3. The final step is to add prerequisites of the courses in the curriculum. We also set the demand value of the course based upon the % of students which have shown interest in the course from the previous/prerequisite course.

The goal is to build a Curriculum which would have courses and their prerequisites. We will be working with the inter relation of the data between the courses in the curriculum.

Files and external data

All in all, there are 8 classes and 5 exception classes in the project:

1. Main.java:
    a. Contains the main method and basically holds the driver code.
    b. Handles adding a course to the curriculum, scheduling the course, and setting prerequisites for the scheduled courses later.
2. Course.java:
    a. This holds the basic structure of how a course would look like in the curriculum.
    b. It has a few course properties including course name, ID, capacity, demandPercentage and demandValue.
3. CourseHelper.java -
    a. This class holds all the common methods and their implementation which are used throughout the curriculum globally.
    b. It consists of methods like:
        i. getCourse - gets the course object from the ID.
        ii. getPrerequisiteCourseIDs – gets a set of all prerequisite's IDs of the courseID passed.
4. CoursePath.java –
    a. This class is used to find the course path to the given End Course IDs.
    b. It holds the logic related to computing the course path to the target courses.
5. Curriculum.java –
    a. Implementation of all the methods which are used in the curriculum.
    b. Also include the logic to initialize and build the graph.
6. EquivalentCourses.java –
    a. This class holds all the logic related to finding all the interchangeable and equivalent courses in the curriculum.
7. Graph.java –

    a. This is an adjacency List implementation where courses are vertex and prerequisites are the edges.
8. LongestChain.java –
    a. This class is used to identify the longest chain/chains of prerequisites in the curriculum.
9. CourseAlreadyScheduledException.java: This exception is triggered when a course is already scheduled, and user tries to schedule it again.
10. CourseNotFoundException.java: This exception is triggered when a user will try to access the course which does not exist in the system.
11. PrerequisiteCycleException.java: This exception is triggered when a cycle in the prerequisite is detected.
12. PrerequisiteNotScheduledYetException: This exception is triggered when a user tries to add a course as a prerequisite, but that course has not been scheduled yet.
13. BottleneckCourseException: This exception is triggered when a there is a bottleneck course detected in the curriculum.

Data structures and their relations to each other

Mostly, the entire project consists of primitive data structures.

Graph –

1. The Graph class is a generic class that can store and manage a directed graph of vertices and edges, where T is the type of data representing vertices (courses in this context).
2. adjacencyList is a private member variable of the class, representing the graph's adjacency list. It's a mapping of vertices (courses) to sets of adjacent vertices (prerequisites).
3. addCourse method allows adding a new vertex (course) to the graph by associating it with an empty set of prerequisites.
4. addPrerequisite method adds a directed edge from a source vertex (course) to a destination vertex, representing a prerequisite relationship between courses.
5. getAllCourses method returns a set containing all the courses (vertices) in the graph.
6. getPrerequisites method returns a set containing all the prerequisites (adjacent vertices) of a given course.
7. hasCourse method checks whether a course exists in the graph by looking it up in the adjacency list.
8. hasPrerequisite method checks if there's a directed edge from a source course to a destination course, indicating a prerequisite relationship.
9. hasCycle method checks if the graph contains any cycles, which would imply circular dependencies among courses. It uses depth-first search (DFS) to identify cycles.
10. checkingCycleDFS is a private helper method used by hasCycle() to perform DFS traversal to detect cycles. It maintains two sets, one for visited vertices and another for vertices in the current recursion stack, to identify cycles.

Assumptions

1. Allowed adding of the bottleneck courses in coursePrerequisite method as we have a method bottleneckCourses which returns a set of bottleneck courses and if we didn't allow adding bottleneck courses then we won't be able to find which courses are bottleneck courses and also this method would be of no use.
2. When there is a cycle or a bottleneck course in the curriculum, coursePath, equivalentCourses and longestPrerequisiteChain methods would throw PrerequisiteCycleException for cycle and BottleneckCourseException for bottleneck course.
3. When a course would have multiple pre-requisite courses added calling the prerequisiteCourse method multiple times, it would update the demand percentage and demand value attributes of current course to the minimum demand pre-requisite. The rationale behind this approach is to ensure that the course's demand percentage is not unrealistically high, which could lead to overestimating the demand for the course and potentially causing issues. By taking the minimum demand percentage of its prerequisites, you ensure that the course's demand is not artificially inflated. This approach reflects a more realistic expectation of how many students are likely to be interested in the course, as it depends on the availability and popularity of its prerequisite courses.
4. The return type of coursePrerequisite method is Boolean because that way we can identify if a prerequisite was added to the course or not and test it accordingly.
5. If the course is already scheduled for the current year, it cannot be scheduled again until the year ends.
6. The capacity for a course should be greater than 0 as there is no point in scheduling a course with 0 capacity.
7. The return type of scheduleCourse method is Boolean because that way we can identify if a course was scheduled successfully or not and test it accordingly.
8. If we pass null as the courseID or prerequisiteID in coursePrerequisite method, that will return false, and no prerequisite would be set for that particular case and won't be considered in all other methods. This is because we don't want to restrict other methods just because there was a wrong input for one or few courses. Those methods would just skip the consideration for the values which return false for coursePrerequisite method.
9. When the courseID in schedule course would be null or when the output of scheduleCourse would be false, that would essentially mean that course was not scheduled and so when we would try to set that course as prerequisite, we would get PrerequisiteNotScheduledYet exception.

Choices

Adjacency List for course-prerequisite representation:

1. Allowed me quick access to a course's prerequisites, essential for curriculum analysis.
2. Efficiency in representing and traversing sparse graphs, such as curriculum prerequisites, where not all courses have many prerequisites, resulting in a more memory-efficient and scalable solution.
3. Suitability for detecting cycles in a graph efficiently. It's well-suited for cycle detection algorithms, and in the context of curriculum prerequisites, this is crucial for ensuring that students don't get stuck in an endless loop of prerequisites.

Key algorithms and design elements

Equivalent Courses –

1. findInterchangeable:
    a. Identify interchangeable courses by comparing subsets of course prerequisites.
    b. Iterate through prerequisites, generating and checking subsets.
    c. If a subset is found in another course's prerequisites, check for remaining unique prerequisites.
    d. If they exist, mark the courses as interchangeable and add them to the result.
2. findEquivalent:
    a. Identify equivalent courses based on the findings from findInterchangeable.
    b. Iterate through interchangeable sets.
    c. Compare each set with all prerequisites.
    d. If one set is a subset of another with unique prerequisites, mark them as equivalent.

Longest Chain:

1. A Dynamic Programming (recursive) method to find the longest chain leading to a course, given its course ID.
2. It utilizes a map, longestChainsMap, to store previously computed longest chains.
3. It returns the longest chain leading to the current course.
4. Iterates through all courses in the curriculum.
5. For each course, computes the longest chain using findLongestChain.
6. Compares the length of the computed chain with the maximum chain length seen so far.
7. If a longer chain is found, update the maximum chain length, and clear the result set.
8. If multiple chains with the same length are found, add them all to the result set.

Limitations

1. Allowing scheduling of the course for the entire year and not term wise. Future versions of the system may consider scheduling courses on a term-by-term basis as it’s a more realistic approach.