Project: Implementing a Sparse Table for University Course Registration (Spring 2024-2025)

Assigned: Saturday, April 5, 2025.

Due: May 3, 2025.

Graded Points: 20

Team: 5 - 10 students

Objectives

1. **Efficient Space Management**: Store only non-empty enrollments using **linked lists**.
2. **Improve Data Retrieval**: Provide **fast queries** for student-course relationships.
3. **Apply Dynamic Memory Techniques**: Use **linked lists** instead of static arrays.
4. **Introduce Additional Features**: Enable **adding/removing students and courses dynamically**. (you will find the list of the required function below)
5. **Bonus: Stack Integration**: Implement **Undo/Redo functionality** for enrollment actions.

Description & Requirements

This assignment requires you to develop a **University Course Registration System** that efficiently stores and retrieves student-course enrollments using **linked lists** instead of arrays. The goal is to implement a **sparse storage structure** that only stores the necessary student-course relationships dynamically.

Core Requirements:

1. **Dynamic Storage using Linked Lists**
   - Each **student** has a linked list of enrolled **courses**.
   - Each **course** has a linked list of **enrolled students**.
   - A **separate linked list** keeps track of all students and another for all courses.
2. **Functionalities to Implement**
   - **Student & Course Management**
     - Add/Remove students dynamically.
     - Add/Remove courses dynamically.
     - Keep track of the **last student added** and **last course added**.
   - **Enrollment Management**
     - Enroll a student in a course.
     - Remove a student from a course.
   - **Data Retrieval & Sorting**
     - List all courses taken by a student.

- List all students enrolled in a course.
- Sort and display the **students list by ID**.
- Sort and display the **courses list by ID**.

3. **Bonus Features (Optional but Recommended) 5 points**
    - Implement **Undo/Redo** for enrollment operations using **stacks**.

Functions to Implement

Function	Description
`addStudent(int studentID)`	Adds a new student to the system (linked list).
`addCourse(int courseID)`	Adds a new course to the system (linked list).
`removeStudent(int studentID)`	Removes a student from the system.
`removeCourse(int courseID)`	Removes a course from the system.
`getLastStudentAdded()`	Returns the ID of the last student added.
`getLastCourseAdded()`	Returns the ID of the last course added.
`enrollStudent(int studentID, int courseID)`	Enrolls a student in a course (updates both lists).
`removeEnrollment(int studentID, int courseID)`	Removes a student's enrollment from a course.
`listCoursesByStudent(int studentID)`	Displays all courses a student is enrolled in.
`listStudentsByCourse(int courseID)`	Displays all students enrolled in a course.
`sortStudentsByID(int courseID)`	Sorts the list of students in ascending order by ID.
`sortCoursesByID(int studentID)`	Sorts the list of courses in ascending order by ID.
`isfullCourse(int courseID)`	Checks if a course if complete or not.
`isnormalstudent(int courseID)`	Checks if a student registers 2-7 courses or not.

University Structure Constraints

- Each student can register **2 – 7 courses per semester**.
- Each course has **20 – 30 students**.
- **Sparse storage**: Store only the necessary student-course enrollments.

How Undo/Redo Works in This System

🔻 **When a Student Enrolls in a Course:**

- The action **"enroll studentID courseID"** is **pushed** onto the **undo stack**.
- The **redo stack** is cleared (since we can't redo after a new action).

🔻 **When Undo is Called:**

- The last action is **popped from the undo stack**.
- It is **executed in reverse** (i.e., removing the student from the course).
- The action is then **pushed onto the redo stack**.

🔻 **When Redo is called:**

- The last undone action is **popped from the redo stack**.
- It is **executed again** (i.e., enrolling the student back into the course).
- The action is **pushed back onto the undo stack**

Delivery Requirements

- Create a zip file that contains (Source code (java file(s)) + documentation(pdf))
- Documentation contains
    - Names of your team, IDs, role of each one
    - Screenshots for the output for the required functions