Data Structures and Algorithms

Project: University Course Enrollment System

1. Overview

This project simulates a **University Course Enrollment System** that processes student enroll/drop requests under prerequisite, capacity, and timetable constraints. The simulation is event-driven over discrete timesteps, tracks waitlists, resolves conflicts, and generates statistics about enrollment efficiency and fairness.

2. Simulation Concept

Time advances in **discrete timesteps** from 1. At each timestep, the system processes queued events (enroll, drop, open/close sections, swap, waitlist promotions), updates course rosters and waitlists, detects time conflicts, and applies priority rules. After executing all actions due at the current timestep, the counter increments.

3. System Entities

3.1 Student

- **SID** (unique)
- **Level** (e.g., Senior/Junior/Sophomore/Freshman)
- **GPA** (for priority ties; optional)
- **Completed Courses** (set of CourseIDs)
- **Current Schedule** (set of SectionIDs with time slots)
- **MaxCreditLimit** (per term)

3.2 Course

- **CID** (identifier)
- **CreditHours**

- **Prereqs** (graph edges to other CourseIDs)
- **Sections** (collection of Section objects)

3.3 Section

- **SecID** (unique per course or global)
- **CID** (parent course)
- **InstructorID** (optional)
- **RoomID** (optional)
- **Capacity**
- **TimeSlot** (e.g., `MW 10:00–11:15` or numeric intervals)
- **Roster** (enrolled SIDs)
- **Waitlist** (priority queue)

3.4 Priority Policy

- **Primary**: Student Level priority (Senior > Junior > Sophomore > Freshman).
- **Secondary**: Earlier request timestamp (FIFO per level).
- **Tertiary** (optional): Higher GPA, or program-specific priority.

4. Constraints & Rules

- **Prerequisites**: a student may enroll if all prereqs are in `Completed Courses`.
- **Capacity**: a section cannot exceed `Capacity`; excess requests go to **Waitlist**.
- **Time Conflicts**: a student cannot enroll in overlapping time slots.
- **Credit Limit**: a student cannot exceed `MaxCreditLimit`.
- **Atomic Swap**: if supported, student can swap sections if both constraints are satisfiable at the same timestep.
- **Auto-Promotion**: when a seat becomes available, the top student on the waitlist (by priority policy) is enrolled automatically at that timestep.

5. Events

- **E** — Enroll request: student requests enrollment in a section.
- **D** — Drop request: student drops a section.
- **O** — Open section: create/activate a new section with capacity/time slot.
- **X** — Close section: close/cancel a section; enrolled students are dropped (and optionally moved).
- **W** — Waitlist promotion trigger: a seat opens; handled internally once capacity increases.
- **S** — Swap request: student requests atomic swap between two sections (same course or alternatives).

Events are read from the input file and executed when **TS = current timestep**.

6. Input / Output File Formats

6.1 Input File

```
C S                            # number of courses, students
P                              # number of prerequisite edges
u v                            # edge: u is a prereq of v  (repeat P ti

CS                             # number of sections
SecID CID Capacity TimeSlot    # CS lines

SINFO                          # S lines of student info
SID Level GPA MaxCredits K     # K = completed courses count
c1 c2 ... cK                   # course IDs (one line)

M                              # number of events
<event lines>
```

Event Lines

- **Enroll**
  `E TS SID SecID`
- **Drop**
  `D TS SID SecID`

- **Open**
  `O TS SecID CID Capacity TimeSlot`
- **Close**
  `X TS SecID`
- **Swap**
  `S TS SID SecID_from SecID_to`

TimeSlot can be encoded as numeric intervals (e.g., `start end`) or textual blocks; the system must consistently detect overlaps.

6.2 Output File

For each successful enrollment action (sorted by **TS ascending**):

```
TS Action SID SecID Result [Reason]
```

Where `Action` ∈ {Enroll, Drop, Promote, Swap, Close}, `Result` ∈ {OK, Waitlisted, Rejected}.

Aggregate statistics at the end:

- Total enroll requests; accepted vs waitlisted vs rejected
- Average **time-to-enroll** for waitlisted students
- Seat utilization per section (roster size / capacity)
- Waitlist churn (promotions processed)
- Fairness summary by level (acceptance rates per level)
- Conflicts prevented (time/prereq/credit violations)

7. Assignment & Promotion Logic

7.1 Enrollment Attempt

Given `(SID, SecID)` at TS:

1. Check **prereqs** via course graph reachability set; if unmet → `Rejected (Prereq)`.
2. Check **time conflict** with `Current Schedule`; if conflict → `Rejected (Conflict)`.

3. Check **credit limit**; if exceeded → `Rejected (Credits)` .
4. If **capacity available** → add to **Roster** (OK).
5. Else → push into **Waitlist** (priority by Level, then FIFO within level).

7.2 Drop & Auto-Promotion

When a student drops and a seat opens, **immediately promote** the top of the waitlist (if any) at the same TS, subject to time/credit checks at promotion time. If promotion fails (conflict), move to the next candidate.

7.3 Swap

Validate the destination section as in **Enrollment Attempt**. If valid, atomically drop from `SecID_from` and enroll into `SecID_to` . Otherwise, keep the original enrollment.

8. Program Modes

- **Interactive Mode**: at each timestep print:
    - Upcoming events (first N)
    - Per-section snapshots: roster sizes, waitlist heads, capacity
    - Per-student snapshots (optional): enrolled credits, pending swaps
    - Promotions executed this step
- **Silent Mode**: only generate the output file.

9. Suggested Class Design

- **Simulator**: event queue, global time, I/O, statistics.
- **Student**: completed courses set, current schedule, credits.
- **Course**: credit hours, prereq adjacency.
- **Section**: capacity, roster set, waitlist PQ, time slot.
- **Event (abstract)**: `TS` , `Execute()` ; derived: `EnrollEvent` , `DropEvent` , `OpenEvent` , `CloseEvent` , `SwapEvent` .
- **UI**: formatted console output (interactive).

10. Recommended Data Structures

Purpose	DS
Events in chronological order	Queue
Course prerequisites	Directed Graph (adjacency lists)
Section rosters	Hash set (SID)
Section waitlists	Priority Queue (by level, then FIFO)
Student schedule	Interval set / balanced tree (for overlap checks)
Student completed courses	Hash set
Output log	List

11. Implementation Notes

- Encode **TimeSlot** into comparable intervals for reliable conflict checks.
- Maintain **stable priority** in the waitlist (level first, then arrival order).
- Guarantee **atomicity** for swap operations.
- Prevent **duplication**: students cannot enroll in two sections of the same course unless allowed.
- Terminate when the event queue is empty and no pending promotions remain.

12. Sample Scenario

At TS=15:

- `E 15 S104 SEC7` → Waitlisted (capacity full; Senior at head retained).
- `D 15 S088 SEC7` → Seat opens; auto-promotion enrolls S104 (now OK).
- `S 15 S073 SEC3 SEC8` → Swap accepted if no conflict and prereqs satisfied.

- `X 15 SEC9` → Section closed; enrolled students dropped with `Close` actions logged.

13. Learning Outcomes

Students will:

- Model **constraint-based scheduling** with prerequisites and capacities.
- Use **graphs** (prereq relations), **priority queues** (waitlists), and **sets/maps** (rosters and schedules).
- Implement **event-driven simulation** with promotion and swap mechanisms.
- Design robust **file I/O** for reproducible experiments and auditing.
- Compute **utilization, fairness, latency**, and conflict statistics.
- Practice **OOP decomposition** and invariants for correctness.