חלק יבש

מבנה הנתונים שלנו מכיל:

- אלמנט מסוג boom אשר בתוכו שני אלמנטים מסוג AVLtree ו-int כמפורט:
  - עץ AVL courses. האלמנטים בעץ זה הם מסוג AVLnode<course,int>. כאשר course הוא אלמנט מחלקת course, המפורט בהמשך. הint בצמד הוא courseID. המיון נעשה באמצעות מספר מזהה של הקורסים. לכל צומת בעץ, בן שמאלי קטן ממנה ובן ימני גדול ממנה.
  - עץ AVL lectures. האלמנטים בעץ זה הם מסוג AVLnode<lecture,lectureKey>. כאשר lecture הוא אלמנט מחלקת lecture, המפורט בהמשך, ובנוסף lectureKey הוא struct המפורט בהמשך. המיון בעץ זה מתבצע על פי lectureKey כפי שיופרט בחלק שיעסוק במבנה זה.
  - class_counter מסוג int. תפקדו לספור את מספר ההרצאות הכולל שהועלו.

אובייקטים:

- AVLnode של מחלקת AVLnode:

T info	
right_son	M key
left_son	parent
height	balance

אלמנט template-י אשר יודע לעבוד עם אלמנטים של מחלקות אחרות. זהו האלמנט הבסיסי המהווה צומת בעץ AVL
info- המידע אותו נרצה לשמור בצומת.
key- המידע לפי נכיל יחס סדר על העץ (נניח כי אופרטור <,>,== ממומשים על ידי טיפוס M).

- course של מחלקת course:

courseID	
num_of_classes	lectures

כאשר lectures הוא מערך מסוג *lecture בגודל num_of_classes, מצביע להרצאה ה-j יושבת בתא ה-j במערך.

lecture של מחלקת lecture:

lectureKey
parent

כאשר parent הוא מצביע להרצאה שמעליה בעץ ההרצאות.

:struct lectureKey

classID
coursed
viewTime

אלמנט זה מכיל את המידע אשר על בסיסו ניתן לערוך השוואות בין אלמנטים מסוג lecture.

פונקציות:

הערה כללית: שגיאות אומנם נכתבו בסוף הפונקציות אך במימוש הן יבדקו ראשונות.

- **Init():**
  - נקים אלמנט מסוג boom.
    - תחתיו קמים שני עצים ריקים. עץ courses ועץ lectures (הקמת עצים עולה לנו O(1), שכן נוצרים שלושה מצביעים באלמנט מסוג AVLtree המאותחלים כ-NULL:
      - מצביע biggest לאיבר הגדול ביותר בעץ.
      - מצביע smallest לאיבר הקטן ביותר בעץ.
      - מצביע root לשורש.

סיבוכיות זמן: O(1) מבצעים מספר סופי של פעולות.

- **StatusType AddCourse(void *DS, int courseID, int numOfClasses):**
  - נחפש בעץ courses את המיקום של הקורס החדש. אם נמצא שהקורס שלנו כבר קיים, נזרוק שגיאת FAILURE. החיפוש עולה לנו O(log(n)) – חיפוש בעץ AVL כפי שנלמד בכיתה.
  - אחרת – נקים אובייקט מסוג course (יש לשים לב שזה כולל את ההקמה של מערך מסוג lecture*), נקים numOfClasses אובייקטים מסוג AVLnode<lecture,lectureKey> כך שלכל תא במערך lectures נכניס את המצביע ל lecture ה-i בתא ה-i. פעולה זו עולה לנו O(m) כאשר m=numOfClasses.
  - ניצור אובייקט מסוג AVLnode<course,int>.
  - נכניס את האובייקט הנ"ל למקומו בעץ courses אם יש צורך בתיקון העץ, נתקן עץ AVL באמצעות גלגולים כפי שנלמד בהרצאה. עולה לנו במקרה הגרוע O(log(n)) (עבור התיקון).
  - אם ה courseID של הקורס החדש גדול מהcourseID שהמצביע biggest מצביע עליו. נבצע השמה של הקורס החדש כך ש-biggest יצביע עליו.
  - אם ה courseID של הקורס החדש קטן מהcourseID שהמצביע smallest מצביע עליו. נבצע השמה של הקורס החדש כך ש-smallest יצביע עליו.
  - נבצע עדכון ל class_couner כך: class_counter+=num_of_classes.
  - שגיאות:
    - כשלון בהקצאת זיכרון – יוחזר ערך ALLOCATION_ERROR.
    - שגיאת קלט – אם הקלט אינו עומד בתנאים הנדרשים כלהלן:

- DS=NULL
- numOfClasses<=0
- courseID<=0

- יוחזר ערך INVALID_INPUT.
- במקרה של הצלחת יוחזר SUCCESS.

<u>סיבוכיות זמן:</u> הקמת אובייקט מסוג course למעט תא המערך, מספר סופי של פעולות.
את תא המערך מקימים ב O(num_of_classes=m)=O(m).
הקמת אובייקט מסוג class עולה O(1).
מציאת מקום ה-course והכנסתו לעץ courses עולה O(log(n)) כפי שנלמד בהרצאה.
סה"כ O(log(n)+m).

- **<u>StatusType RemoveCourse(void *DS, int courseID):</u>**
  - אם הקורס שאנו מוציאים הוא הcourseID ש – smallest מצביע עליו. נבצע השמה כך ש- smallest יצביע להבא הקטן ביותר (במידה ויש כזה) של הקורס אותו מוציאים.
  - אם הקורס שאנו מוציאים הוא הcourseID ש – biggest מצביע עליו. נבצע השמה כך ש- biggest יצביע להבא הגדול ביותר (במידה ויש כזה) של הקורס אותו מוציאים.
  - נמצא את courseID בעץ courses. (O(log(n))). נעבור על המערך lectures עבור כל lecture (סה"כ m הרצאות):
    - נוציא את האלמנט AVLnode<lecture,lectureKey> מהעץ lectures. עולה לנו O(1) למצוא את ההרצאה כיוון שיש לנו מצביע אליה.
    - נתקן את העץ. עולה לנו O(log(M)) במקרה הגרוע. כאשר M הוא מספר כלל ההרצאות בעץ.
  - נוציא את הקורס מעץ הקורסים ונתקן. התיקון עולה O(log(n)) במקרה הגרוע.
  <u>סיבוכיות זמן:</u> סה"כ קיבלנו O(log(n)+mlog(M))=O(mlog(M))

- **<u>StatusType WatchClass(void *DS, int courseID, int classID, int time):</u>**
  - נמצא את הcourse שנתון לנו הcourseID שלו בעץ courses, עולה לנו O(log(n).
  - באמצעות מערך המצביעים להרצאות שמחזיק אלמנט מסוג course אנו יכולים להגיע לnode של ההרצאה בעץ ההרצאות בO(1).
  - נוציא את ההרצאה הנדרשת מהעץ lectures. עולה לנו O(log(M)).
  - נתקן לאותה הרצאה את זמן הצפייה שלה. O(1).
  - נחזיר לעץ lectures ונתקן את העץ. (כעת היא תכנס למיקומה הנכון עבור הזמן המעודכן). עולה לנו O(log(M))+O(log(M)).
  - שגיאות:
    - במקרה של כשל בהקצאת זכרון יוחזר ALLOCATION_ERROR.
    - אם time<=0 או classID<=0 או courseID<=0 או DS==NULL או classID>numOfClasses יוחזר INVALID_INPUT.
    - אם לא קיים courseID כזה יוחזר FAILURE.
    - במקרה של הצלחה יוחזר SUCCESS.
  - **<u>סיבוכיות זמן:</u>** $O(\log(M))$

- **StatusType TimeViewed(void *DS, int courseID, int classID, int *timeViewed):**
  - נחפש את הקורס המתאים ל-courseID הנתון בעץ courses. עולה O(log(n)).
  - באמצעות מערך המצביעים של הקורס נוכל לגשת להרצאה זו ב O(1).
  - נוציא את המידע הנדרש מאלמנט ההרצאה.
  - שגיאות:
    - במקרה של כשל בהקצאת זכרון יוחזר ALLOCATION_ERROR.
    - אם classID<=0 או courseID<=0 או DS==NULL או classID>numOfClasses יוחזר INVALID_INPUT.
    - אם לא קיים courseID כזה יוחזר FAILURE.
    - במקרה של הצלחה יוחזר SUCCESS.
  - **סיבוכיות זמן:** O(log(n)) – רק החיפוש בעץ courses הוא העלות כאן.

- **StatusType GetMostViewed(void *DS, int numOfClasses, int *courses, int *classes):**
  - נתבונן בשני מקרים.
    - צפו במספר שווה או גדול של ההרצאות מהמספר המבוקש:
      - ניגש באמצעות המצביע biggest של עץ ההרצאות להרצאה שנצפתה בזמן הגדול ביותר ונבצע הכנסות למערכים על ידי הפונקציה applyFromRight.
        - הפונקציה applyFromRight:
          - מתחילה מהאיבר הגדול ביותר באמצעות המצביעה biggest (מגיע אליו ב(O(1).
          - זו פונקציה רקורסיבית הפועלת כך-
          - ביצוע פעולה על האיבר הנוכחי.
          - סיור reverseInOrder על הבן השמאלי של האיבר.
          - זימון מחדש על הparent של האיבר הנוכחי.
          - זמן הפעולה שלה עבור קלט בגודל k הוא O(k) שכן היא עובר על כל האיברים בקלט.
          - את מספר הפעולות של פונקציה זו ניתן להגביל, וכך למנוע שתעבור על כל עץ ההרצאות אם מבוקשות פחות הרצאות מ-M.
    - צפו בפחות הרצאות מהמספר המבוקש:
      - ראשית נבצע את עץ ההרצאות את הכתוב לעיל. לאחר מכן עבור class_counter – numOfClasses ההרצאות שנותרו, נפנה לעץ הקורסים. נגיע ב O(1) לקורס בעל המספר המזהה הקטן ביותר על ידי המצביע smallest.
      - נעבור על מערך ההרצאות שלו. נכניס למערכים רק את ההרצאות שזמן הצפיה שלהן הוא 0.
      - ונמשיך לקורס הבא בעזרת הפונקציה applyFromLeft עד שנסיים את המכסה המבוקשת.
        - הפונקציה applyFromLeft עובדת בצורה סימטרית לפונקציה applyFromRight שתארנו לעיל.
        - **הערת נכונות:** מספר האיברים **"המיותרים"** (ההרצאות שכבר הוכנסו למערכים כי כן צפו בהן) שאנו עוברים עליהם הוא לכל היותר numOfClasses-1.

- שגיאות:
  - כשלון בהקצאת זיכרון – יוחזר ערך ALLOCATION_ERROR.
  - שגיאת קלט – אם הקלט אינו עומד בתנאים הנדרשים כלהלן:
    - DS=NULL
    - numOfClasses<=0
  - courseID<=0 יוחזר ערך INVALID_INPUT.
  - אם numOfClasses>class_counter יוחזר FAILURE
  - במקרה של הצלחת יוחזר SUCCESS.
- **<u>סיבוכיות זמן:</u>** בעזרת המצביעים ישנה גישה מהירה לאיברים הרלוונטיים ללא צורך בחיפוש. לכן עלות הפעולות היא ככמות האיברים אותה נדרש להדפיס O(m) כאשר m=numOfClasses. נעיר שהמקרה הגרוע ביותר הוזכר קודם "לכן לכל היותר נעבור על $2 \cdot numOfClasses - 1$ כיתות" וגם הוא כמובן O(m).

- **<u>void Quit(void **DS):</u>**
  - נקרא ל destructor של boom.
  - נעבור ב post-Order על עץ courses, ונשחרר את כל הקצאותיו הדינאמיות O (n+m).
  - נשחרר את עץ courses (נשים לב שאחרי השלב הקודם עץ lectures לא קיים יותר). O(n)
  - **<u>סיבוכיות זמן:</u>** O(n+m).