חלק יבש:

מבנה הנתונים שלנו מכיל:

* עץ AVL של courses. האלמנטים בעץ זה הם מסוג course. courses הוא מצביע לשורש העץ. המיון נעשה באמצעות מספר מזהה של הקורסים. לכל צומת בעץ, בן שמאלי קטן ממנה ובן ימני גדול ממנה.
* עץ AVL של hubs. האלמנטים בעץ זה הם מסוג hub. hubs הוא מצביע לשורש העץ. המיון נעשה לפי מספר הhub (שהוא מייצג את משך הצפיה בהרצאה -time) כאשר בן שמאלי של צומת מייצג זמן קצר ממנה ובן ימני מייצג זמן ארוך ממנה.
* עצי AVL של sub\_hubs. האלמנטים בעצים אלו הם מסוג class. Sub\_hubs הוא מצביע לשורש העץ. המיון נעשה לפי מספר ההרצאה (פנים קורסי) במקרה של תיקו, יוכרע באמצעות מספר הקורס. לכל צומת בן שמאלי מייצג כיתה שקטנה ממנה במספר הכיתה ו-או במספר הקורס, בן ימני מייצג כיתה שגדולה ממנה במספר הכיתה ו-או במספר הקורס.
* most\_viewed - מצביע מסוג \*hub שיצביע לhub- בעל hub\_ID הגדול ביותר.
* first\_course\_to\_print מצביע מסוג \*course לcourse בעל הcourse\_ID הנמוך ביותר בעץ courses.
* class\_counter מסוג int. תפקדו לספור את מספר ההרצאות הכולל שהועלו.

אובייקטים:

* course:

|  |  |
| --- | --- |
| course\_ID | |
| class\_array | right\_son |
| Prev | left\_son |
| num\_of\_classes | |

כאשר classes\_array הוא מערך מסוג class\* בגודל j כאשר j הוא מספר ההרצאות המוקלטות בקורס course ID. הכיתה ה j יושבת בתא הj- במערך.

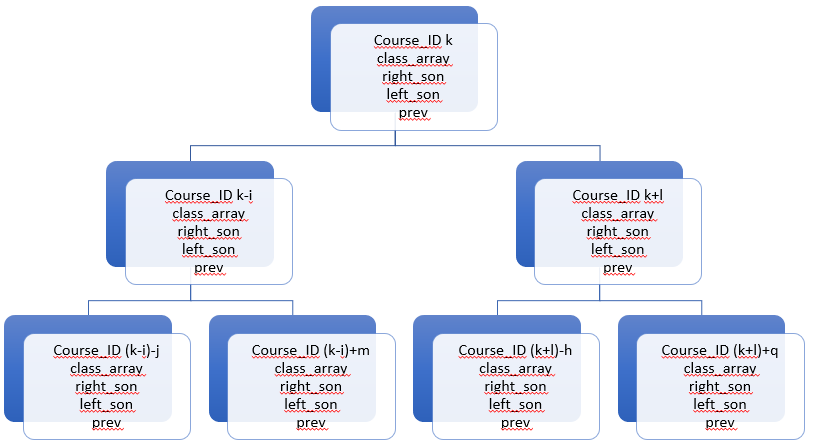
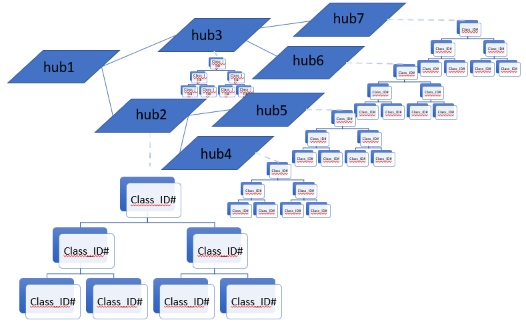
|  |  |
| --- | --- |
| class\_ID | |
| course\_ID | right\_son |
| Prev | left\_son |
| hub\_ID | |

* class:
* hub:

|  |  |
| --- | --- |
| hub\_ID | right\_son |
| Prev | left\_son |
| sub\_hub | smallest |

כאשר sub\_hub הוא מצביע לשורש של העץ של כל ה classes בעלי אותו ייצוג זמן (אותו מייצג ה hub).  
smallest הוא מצביע לאיבר בעל ה courseID הקטן ביותר בhub (במקרה של תיקו – גם בעל ה classID הקטן ביותר).

עץ courses:



עץ hubs ומתחתיו עצי ה -sub\_hubs

הסבר מילולי למבנה הנתונים: מבנה הנתונים מתחלק על פני שני עצים. שני העצים הנ"ל הם עצי AVL. עץ אחד הוא עץ courses שתפקידו לשמור ולנהל מידע עבור הקורסים עצמם כמו גם היכולת לקשר בין קורס להרצאות שלו (על ידי מערך מצביעים). העץ ממיין את הקורסים על פי ה courseID ומציע דרך נוחה לגישה להרצאות ששמורות בעץ השני. העץ השני הוא עץ של hubs, בעצם כל hub מייצג זמן. זה הזמן בו צפו בכל ההרצאות שנמצאות "בתוך הhub". הכוונה ל"בתוך ה"hub היא שכל hub הוא מצביע לעץ (sub\_hub) AVL של הרצאות, כאשר הוא מצביע למעשה לשורש העץ. אם אין הרצאות בhub זה אזי אין הרצאות שנצפו בהם במספר הדקות שהhub מייצג ולכן הוא נמחק. בתוך ה sub\_hub ההרצאות ממוינות על פי מספר הקורס ומספר ההרצאה (עבור מצב של תיקו).

פונקציות:

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

* **Init():** 
  + נקים מצביע ל course מסוג course\*, נקרא לו courses והוא יאותחל עם ערך NULL.
  + נקים את המצביע first\_course\_to\_print מסוג course\* שיאותחל להצביע על courses.
  + נקים מצביע לhub מסוג hub\* נקרא לו hubs והוא יאותחל עם הערך NULL.
  + נקים את המצביע most\_viewed מסוג hub\* והוא יצביע לhubs.  
    **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.
  + אחרת – נקים אובייקט מסוג course (יש לשים לב שזה כולל את ההקמה של מערך מסוג \*class), נקים numOfClasses אובייקטים מסוג class כך שלכל תא במערך classes\_array נכניס את המצביע לclass הi- בתא הi-. (לדוגמא נקים אובייקט מסוג class שנאתחל עם course\_ID = courseID ואת class\_ID=5, מצביע לכיתה זו יכנס לתא 5 במערך).
  + נכניס את האובייקט course למקומו בעץ courses אם יש צורך בתיקון העץ, נתקן עץ AVL באמצעות גלגולים כפי שנלמד בהרצאה.
  + אם ה courseID של הקורס החדש קטן מהcourseID שהfirst\_course\_to\_print מצביע עליו. נבצע השמה של הקורס החדש כך ש-first\_course\_to\_print יצביע עליו.
  + נבצע עדכון ל class\_couner כך: class\_counter+=numOfClasses.
  + 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.

**סיבוכיות זמן:** 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 הcourse- והכנסתו לעץ courses עולה O(log(n)) כפי שנלמד בהרצאה.  
סה"כ O(log(n)+m).

* **StatusType RemoveCourse(void \*DS, int courseID):**
  + אם הקורס שאנו מוציאים הוא הcourseID ש – first\_course\_to\_pring מצביע עליו. נבצע השמה כך ש- first\_course\_to\_print יצביע לprev של הקורס אותו מוציאים.
  + נמצא את courseID בעץ courses. (O(log(n))). נעבור על המערך classes\_array עבור כל class:
    - אם ערך הhub שלו הוא NULL, נמחק את האובייקט.
    - אחרת –
      * נוציא את האובייקט מהעץ sub\_hub בוא הוא נמצא. (ניתן להגיע לעץ ב O(1) כיוון שהclass- שומר מצביע לhub- שמצביע לsub\_hub שמכיל אותו). מחיקת האובייקט מהsub\_hub- עולה במקרה הגרוע ביותר O(log(M)) (הוצאת איבר מעץ AVL כפי שנלמד בהרצאה).
        + אם הsub\_hub- ריק לאחר ההוצאה (זאת אומרת לא קיים יותר עץ sub\_hub), כלומר הhub- שמצביע אליו הclass- מצביע כעת לNULL, נוציא את הhub- מעץ hubs ונשחרר אותו. זה עולה במקרה הגרוע O(log(M)) לפי הוצאת צומת מעץ AVL כפי שנלמד בהרצאה (עץ hubs).
  + כעת נשחרר את האובייקט class.
  + לאחר שבצענו זאת לכל המערך classes\_array, נוציא את האובייקט course הנדרש מעץ courses על פי הוצאת איבר מעץ AVL כפי שנלמד בהרצאה. ונשחררו. במקרה הגרוע פעולה זו עולה לנו O(log(n))
  + נבצע עדכון ל class\_couner כך: class\_counter-=numOfClasses.
  + הערה: בכל פעם שנוציא class מ sub\_hub- והוא אינו נשאר ריק, אז אם הclass שהוצאנו היה הקטן ביותר. נעדכן את smallest שהצביע עליו, להצביע על הprev שלו.

**סיבוכיות זמן:** סה"כ קיבלנו O(log(n)+mlog(M)+log(M)+log(n))=O(mlog(M))

* **StatusType WatchClass(void \*DS, int courseID, int classID, int time):**
  + נמצא את הcourse שנתון לנו הcourseID שלו בעץ courses.
  + נחלק לשני מקרים:
    - עוד לא צפו ב classID זה עד כה (classes\_array[classID]->hub==NULL)
      * נבדוק האם קיים hub כך ש hub\_ID==t בעץ hubs)
        + אם כן – נוסיף את ה- class הנ"ל ל-sub\_hub ש-hub זה מצביע עליו. במקרה הגרוע O(log(M)) – הסבר: log(M) למצוא את הhub ועוד log(M) לצרף את הclass ל sub\_hub.
        + אם לאו- נקים hub כזה, נצרף אותו לעץ hubs עולה O(log(M)), הhub החדש יצביע לclass החדש (הוא כרגע שורש ה sub\_hub).
    - אם כן צפו כבר בכיתה זו עד כה:
      * נוציא את הclass הנ"ל מהעץ sub\_hub נו הוא נמצא כרגע. עולה O(log(M)).
      * נבדוק האם הוא היה לבד בsub\_hub זה:
        + אם כן נשחרר את הhub שמעליו. עולה O(log(M)).
        + אם לאו, נתקן את sub\_hub זה לפי צורך. עולה O(log(M)).
      * נבדוק האם קיים hub כך ש hub\_ID==t(old)+t (בעץ hubs)
        + אם כן – נוסיף את ה- class הנ"ל ל-sub\_hub ש-hub זה מצביע עליו. במקרה הגרוע O(log(M)) – הסבר: log(M) למצוא את הhub ועוד log(M) לצרף את הclass ל sub\_hub.
        + אם לאו- נקים hub כזה, נצרף אותו לעץ hubs עולה O(log(M)), הhub החדש יצביע לclass החדש (הוא כרגע שורש ה sub\_hub).
  + הערה: בכל פעם שנוציא class מ sub\_hub- והוא אינו נשאר ריק, אז אם הclass שהוצאנו היה הקטן ביותר. נעדכן את smallest שהצביע עליו, להצביע על הprev שלו.  
    בכל פעם שנכניס class ל- sub\_hub, נבדוק אם הזוג הסדור (courseID,classID) שלו הוא הקטן ביותר. אם כן נעדכן את smallest של ה hub להצביע עליו.
  + 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.
  + **סיבוכיות זמן:**
* **StatusType TimeViewed(void \*DS, int courseID, int classID, int \*timeViewed):**
  + נחפש את הקורס המתאים לcourseID- הנתון בעץ courses. עולה O(log(n)).
  + נבצע השמה של ה hub\_ID של ה hub שתחתיו מתארחת ה class ב- sub\_hub אליו הוא מצביע לתוך timeViewed. לפי מבנה הנתונים המתואר יש לנו גישה למידע זה בעזרת classes\_array ושמירת כתובת ה hub שתחתיו הclass נמצא ב O(1).
  + במידה ואין צפיות עוד בכיתה זו, ה hub\_ID שלה NULL ולכן נבצע השמה של 0 לtimeViewed.
  + 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):**
  + נחלק לשני מקרים:
    - most\_viewed==NULL (לא צפו עדיין באף הרצאה):
      * נעבור ל\*first\_course\_to\_print, ונדפיס את איבריו לפי מערך classes\_array שלו בסדר עולה לתוך המערכים courses ו classes בהתאמה. נשים לב כי כל עוד אנחנו באותו courseID יודפס לתוך מערך courses אותו courseID. במידה ו-numOfClasses>m כאשר m הוא כמות הclasses בקורס זה. נעבור לקורסים הבאים בסדר inOrder, עד שנדפיס לתוך המערכים הנתונים numOfClasses נתונים.
    - most\_viewed!=NULL
      * נעבור לmost\_viewedf ונדפיס את האיברים שבו החל מהאיבר smallest בסדר inOrder.
      * במידה וסיימנו לעבור על כל הhub הנוכחי ולא סיימנו למלא numOfClasses נתונים למערכים, נעבור לhub שקודם לו בעץ hubs (בצורת inOrder מהופך – כך שקודם מדפיסים ענף ימין – אז שורש – אז ענף שמאל).
      * במידה ונגמרו הhubs אך עדיין לא הגענו למכסה הנדרשת, נעבור להדפיס לתוך המערכים הרצאות שלא צפו בהן עדיין. נעשה זאת מעץ courses, ע"י כך שנתחיל ב first\_course\_to\_print.
        + בכל קורס – נעבור בכל תא במערך classes\_array. נבדוק האם קיימים בו classes שמצביעים ל NULL ב hub\_ID (משמע שזמן הצפיה בהם הוא 0). אם כן נדפיס אותם לתוך המערכים. אם לא נמשיך הלאה. **הערה:** המעבר בשיטה זו על העץ אינו פוגע בסיבוכיות הזמן. איננו עושים יותר מnumOfClasses השוואות מיותרות (שבסופן מתגלה שהאיבר כבר הודפס למערכים כי צפו בו) כי לא קיימים יותר מ numOfClasses כיתות שזמן הצפיה בהן גדול מ0 כי אחרת לא היינו מגיעים בכלל לחפש בעץ courses. (לכן לכל היותר נעבור על כיתות).
        + נעבור על העץ בשיטת inOrder עד שנמלא את המכסה (קרי numOfCLasses נתונים).
  + שגיאות:
    - כשלון בהקצאת זיכרון – יוחזר ערך ALLOCATION\_ERROR.
    - שגיאת קלט – אם הקלט אינו עומד בתנאים הנדרשים כלהלן:
      * DS=NULL
      * numOfClasses<=0
    - courseID<=0יוחזר ערך INVALID\_INPUT.
    - אם numOfClasses>class\_counter יוחזר FAILURE
    - במקרה של הצלחת יוחזר SUCCESS.
  + **סיבוכיות זמן:** בעזרת המצביעים : most\_viewed, first\_course\_to\_print,smallest 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 כיתות" וגם הוא כמובן O(m).
* **void Quit(void \*\*DS):**
  + נקרא ל destructor של מבנה הנתונים.
  + נעבור ב post-Order על עץ courses, ונשחרר את כל איבריו O (n).
  + בצורה רקורסיבית נעבור על עץ hubs ב post-Order. O(m)
  + בתוך כל hub נעבור על עץ ה sub\_hub בpost-Order ונשחרר את כל איבריו.
  + לאחר מכן נשחרר את הhub- שהצביע ל sub\_hub הנוכחי.
  + **סיבוכיות זמן:** O(n+m).