***מבנה הנתונים מכיל:***

***1 .******coursesData*** *– טבלת ערבול דינמית מבוססת chain-hashing שמכילה את הקורסים ע"פ פעולת ערבול modN כאשר N הוא הגודל הנוכחי של טבלת הערבול. הטבלה מכילה:*

*א. פקטור עומס.*

*ב. גודל המערך הנוכחי.*

*ג. כמות האיברים בטבלה.*

*ד. מערך דינמי, כך שכל תא במערך מכיל רשימה מקושרת. כל תא ברשימה המקושרת מייצג קורס ויכיל מערך דינאמי בהתאם לכמות השיעורים שהוכנסו אליו המערך יגדל בהתאם לפקטור העומס.*

***2. timeTree*** *– עץ דרגות, מסוג AVL\_v2 ממויין לפי מפתחות מסוג trio (הסבר על AVL\_v2 ועל trio בעמוד הבא). כאשר כל תא בעץ מכיל:*

*א. מספר הבנים של הצומת (rank) – דרגת עלה תהיה שווה ל1.*

*ב. מצביע לאבא.*

*ג. בן ימני.*

*ד. בן שמאלי.*

*ה. גובה הצומת הנוכחי.*

***- עץ AVL\_v2*** *הוא עץ AVL מסוג עץ דרגות, שמתעדכן בדיוק ע"פי האלגוריתם שלמדנו בכיתה עבור עץ דרגות.*

*השורש (הצומת העליונה): יחזיק NULL במצביע parent.*

*בגלגולים: נדאג לעדכן בנוסף לleft וright את מצביע parent, העדכון הוא חיבור וניתוק של עד 3 מצביעים ולכן אין שינוי בסיבוכיות הזמן.*

*בעת הכנסה של איבר: נבצע הכנסה כפי שלמדנו ונעדכן גם את parent של הצומת החדשה + את דרגת הצמתים במסלול.*

*במקרה של הסרה: נבצע הסרה כפי שלמדנו, נעביר (במקרה הצורך) את הparent של הנמחק לparent של הבנים שלו + נעדכן את דרגת הצמתים במסלול.*

*סיבוכיות מקום: נשארת O(n) כאשר n מס' הצמתים בעץ - הוספנו רק מצביע + דרגה לכל צומת.*

***- trio*** *הוא מבנה שמכיל 3 שדות מסוג int – 1. זמן הצפייה, 2. מספר הקורס, 3. מספר השיעור.*

*למבנה יהיו אופרטורי השוואה (<, >, == וכו') שיעבדו באופן הבא:*

*trio1 יהיה קטן מtrio2 כאשר (בגדול בדיוק להפך בכל אח מהשלבים):*

*1. זמן הצפייה של trio1 קטן יותר משל trio2.*

*2. אם זמני הצפייה של trio1 ו trio2 שווים, אז מספר הקורס של trio1 גדול ממספר הקורס של trio2.*

*3. אם גם זמני הצפייה וגם מספרי הקורסים של trio1 ו trio2 שווים, אז מספר הכיתה של trio1 גדול ממספר הכיתה של trio2.*

*4. אם כל השדות בtrio1 וtrio2 שווים - מתקיים שוויון.*

***- רשימה מקושרת*** *מכילה בכל תא:*

*א. מספר קורס.*

*ב. מערך שכל אינדקס תא בו מייצג שיעור, המערך דינמי – כך שכשמוסיפים יותר שיעורים מגודלו, הוא יגדל כפי שנלמד.*

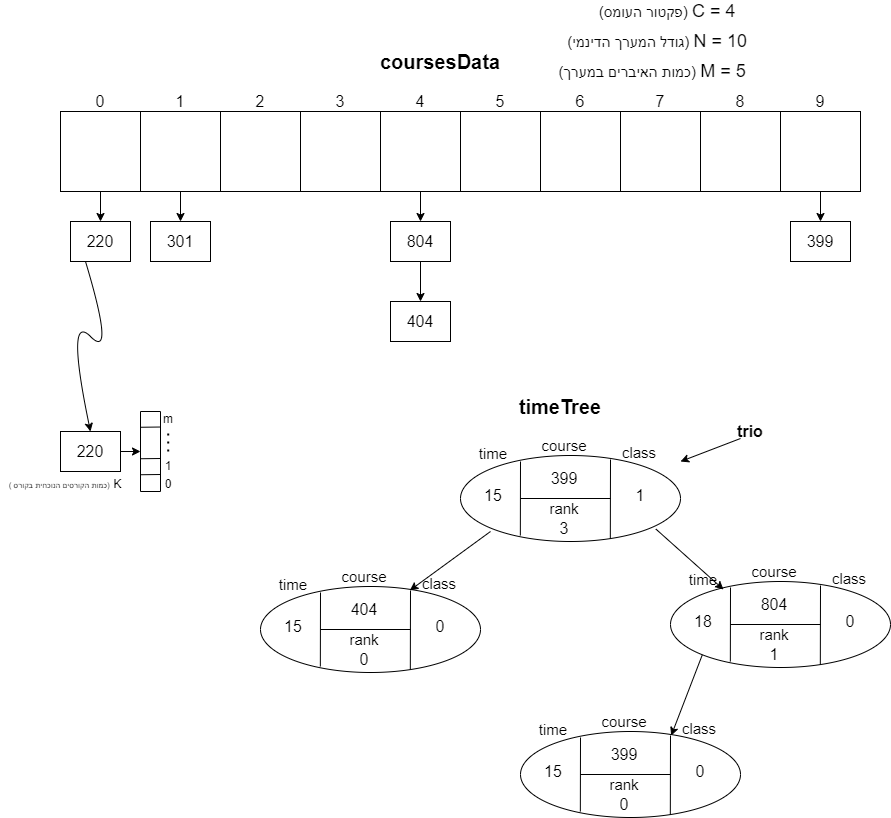
*ג. כמות הקורסים הנוכחית.*

*ד. כל תא במערך יכיל מצביע לתא בעץ שבו השיעור מאוכסן (לפי התא השלישי בtrio), בdefault יש בכל תא NULL.*

*ה. מצביע לתא הבא.*

***איור להמחשה:***

*- עבור getIthWatchedClass(2), נקבל למשל את קורס 399, שיעור 0 (ראו מימושים לפונקציות + סיבוכיות)*

******

***מימושים לפונקציות:***

***- Init() .1***

***א.*** *צור טבלת hash בשם coursesData (מתחילה בגודל 2, ריקה).*

***ב.*** *צור עץ AVL\_v2 מסוג trio, בשם timeTree.*

*-* ***AddCourse(void\* DS, int courseID) .2***

***א.*** *צור תא חדש עם הערך courseID והכנס את התא לcoursesData בעזרת פונקציית הערבול של טבלת הhash.*

***ב.*** *צור מערך דינמי שישמש את הקורס החדש (המערך יהיה בגודל התחלתי 2, ו0 בכמות הקורסים הנוכחית).*

***הערה:*** *אם פקטור העומס כפול גודל המערך של coursesData יהיה קטן או שווה לכמות האיברים בטבלה, הטבלה תגדיל את עצמה פי 2, תעדכן את גודל המערך, ותדאג לכך שהאיברים יוכנסו בהתאם לפונקציית הערבול החדשה (שהשתנתה בעקבות השינוי בגודל המערך) – כפי שנלמד בכיתה.*

***- RemoveCourse(void\* DS, int courseID) .3***

***א.*** *גש בעזרת פונקציית הערבול לתא המתאים (בעל ערך courseID) בcourseData (אם לא קיים כזה החזר FAILURE), והחל בסריקה של מערך הפונטרים שבצומת הנ"ל.*

***ב.*** *אם הכתובת שבתא במערך היא NULL המשך לתא הבא במערך. אחרת, גש לכתובת שבתא במערך (בכתובת יש את הצומת של השיעור הנ"ל).*

***ג.*** *מחק את הצומת הנ"ל מהעץ timeTree.*

***ד.*** *בסיום הסריקה – מחק את המערך הדינמי ואת הקורס מcoursesData.*

***הערה:*** *אם פקטור העומס כפול גודל המערך של coursesData יהיה גדול פי 2 מכמות האיברים בטבלה, הטבלה תקטין את עצמה פי 2, תעדכן את גודל המערך, ותדאג לכך שהאיברים יוכנסו בהתאם לפונקציית הערבול החדשה (שהשתנתה בעקבות השינוי בגודל המערך) – כפי שנלמד בכיתה.*

***- AddClass(void\* DS, int courseID, int\* classID) .4***

***א.*** *גש בעזרת פונקציית הערבול לתא המתאים (בעל ערך courseID) בcourseData (אם לא קיים כזה החזר FAILURE).*

***ב.*** *הכנס למצביע classID את הערך שבכמות הקורסים הנוכחית שבתא.*

***ג.*** *הגדל ב1 את הערך שבכמות הקורסים הנוכחית שבתא.*

***הערה****: אם כמות הקורסים הנוכחית שווה לגודל המערך, המערך יגדיל את עצמו פי 2 ויעדכן את השדות בהתאם – כפי שנלמד בכיתה לגבי מערכים דינמיים.*

***- WatchClass(void\* DS, int courseID, int classID, int time) .5***

***א.*** *גש בעזרת פונקציית הערבול לתא המתאים (בעל ערך courseID) בcourseData (אם לא קיים כזה החזר FAILURE).*

***ב.*** *גש לתא classID במערך הדינמי שבתא.*

***ב.1.*** *אם הערך שבתא הוא NULL:*

***ב.1.1.*** *הכנס צומת חדשה לעץ timeTree, עם trio חדש בעל הערכים: 1. time, 2. courseID, 3. classID.*

***ב.2.1.*** *שמור בתא כתובת לצומת החדשה הנ"ל (שהוכנסה לעץ).*

***ב.2.*** *אחרת (יש מצביע לצומת בתא):*

***ב.1.2.*** *הוצא את הצומת מהעץ והוסף לערך המתאים בtrio את time.*

***ב.2.2.*** *הכנס את הצומת לעץ.*

***- TimeViewed(void\* DS, int courseID, int classID, int \*timeViewed) .6***

***א.*** *גש בעזרת פונקציית הערבול לתא המתאים (בעל ערך courseID) בcourseData (אם לא קיים כזה החזר FAILURE).*

***ב.*** *גש לתא classID שבמערך הדינמי שבתא שניגשנו אליו בסעיף א.*

***ג.*** *אם ערך התא NULL עדכן את timeViewed להיות שווה 0. אחרת, עדכן אותו להיות שווה לערך הtime שבtrio שבכתובת.*

***- GetIthWatchedClass(void\* DS, int i, int \*courseID, int \*classID) .7***

***א.*** *אם i גדול מrank (כמות הבנים) של שורש timeTree, החזר FAILURE.*

***ב.*** *התחל לולאת while שתרוץ כל עוד i>0.*

***ג.*** *אם i-1 שווה לrank הבן הימני של הצומת הנוכחי:*

***ג.1.*** *הכנס את ערך הclass שבtrio של הצומת הנוכחי לclassID ואת ערך הcourse שבtrio של הצומת הנוכחי לcourseID.*

***ג.2.*** *החזר SUCCESS.*

***ד.*** *אחרת, אם i-1 קטן מrank הבן הימני:*

***ד.1.*** *גש מהצומת הנוכחי לבן הימני.*

***ה.*** *אחרת (i-1 גדול מrank הבן הימני):*

***ה.1.*** *החסר מi את rank הבן הימני + 1. (כלומר, i = i-rank-1).*

***ה.2.*** *גש מהצומת הנוכחי לבן השמאלי.*

***- Quit(void\*\* DS) .8***

***א.*** *הפוך את DS לNULL כך שיקראו הדיסטרקטור של מבנה הנתונים.*

*\*הדיסטרקטור – מוחק סך הכל סדר גודל של כ-4M איברים:*

*את העץ הראשי שמחזיק את המערכים עם המצביעים ואת הרשימה המקושרת שמחזיקה את העץ בתוך עץ. כפי שנראה בסיבוכיות המקום, מכיוון שאנו דואגים לכך שלא יהיו כפילויות בעצים וברשימה המקושרת, סיבוכיות המקום תהיה במקרה הגרוע 3M, (M תאי רשימה מקושרת, M תאי קורסים, M תאי שיעורים).*

***נכונות:***

***- Init() .1*** *– מאתחל את מבנה הנתונים שלנו עם הערכים הנחוצים לנו במבנה.*

*-* ***AddCourse(void\* DS, int courseID, int numOfClasses) .2*** *– מוסיף את הקורס המבוקש לעץ הקורסים הראשי שלנו ויוצר מערך בו כל אינדקס מייצג את השיעורים הקיימים בקורס. כמו כן, דואג לעדכן כי כרגע דקות הצפייה בכלל השיעורים של הקורס הנוכחי הם 0.* ***(שימו לב, תא 0 תמיד ישאר ברשימה המקושרת, הוא אינו נמחק!).***

***- RemoveCourse(void\* DS, int courseID) .3*** *– מוחק תחילה את כל השיעורים של הקורס, כולל הקורס עצמו מהמבנה שמחזיק את דקות הצפייה, תוך מחיקת תאים מהרשימה המקושרת שאין בהם צורך יותר (כדי לשמור על סיבוכיות המקום). לאחר מכן דואג למחוק את הקורס מעץ הקורסים הראשי.*

***- WatchClass(void\* DS, int courseID, int classID, int time) .4*** *– דואג למחוק את השיעור מהערך הקודם של זמן הצפייה שלו, ובמקרה הצורך גם מחיקת קורס ותא מהרשימה המקושרת אם צריך. לאחר מכן דואג להוסיף (ובמקרה הצורך ליצור) לתא עם הערך המתאים את השיעור. בסיום תהליך עדכון מבנה הזמנים, מעדכן גם את התא במערך המתאים שבעץ הקורסים הראשי להצביע לערך הזמן הנכון.*

***- TimeViewed(void\* DS, int courseID, int classID, int \*timeViewed) .5*** *– דואג לעדכן את המצביע שהכניסו לנו לזמן המתאים לשיעור שבקורס המבוקש.*

***- GetMostViewedClasses(void\* DS, int numOfClasses, int \*courses, int \*classes) .6*** *– מעדכן את המערכים הנתונים לנו בשיעורים הנצפים ביותר, הדבר נעשה ע"י גישה למצביע ששמרנו לערך הצפייה המקסימאלי במערכת, ותוך כדי גישה מיידית לשיעור הקטן ביותר שבקורס הקטן ביותר שבתא עם ערך הצפייה הגדול ביותר, וביצוע סיור inorder החל ממנו. הפונקציה מבצעת עדכון של המערכים לפי סדר לקסיקוגרפי כאשר סדר עדכון המערכים יהיה לפי:*

*1. מספר הקורס מהנמוך לגבוה.*

*2. מספר השיעור מהנמוך לגבוה.*

*ועוברת לזמנים קטנים יותר (מהגדול לקטן) לפי הצורך. כמו כן, הפונקציה דואגת לעצור כאשר הגענו למספר הקורסים שהיוזר ביקש לעדכן במערכים (או בשגיאה כלשהי כמובן..).*

***- Quit(void\*\* DS) .7***

*הפיכת מצביע לNULL גורמת ליציאה מהסקופ כך שהדיסטרקטורים של מבני הנתונים שלנו נקראים.*

***סיבוכיות:***

***נסמן: n – מספר הקורסים במערכת, m – מספר שיעורים בקורס מסוים, M – מספר השיעורים בכלל המערכת.***

*1 .זמן של* ***Init()*** *–*

1. *אתחול מבנה נתונים שכולל: עץ ריק – O(1).*
2. *רשימה מקושרת עם איבר בודד – O(1).*
3. *מצביע – O(1).*

*סה"כ:* ***O(1)****.*

*2 .זמן של* ***AddCourse(void\* DS, int courseID, int numOfClasses)***  *-*

1. *הכנסת צומת חדשה לעץ AVL עם לכל היותר n צמתים (קורה פעמיים) – O(2log(n)).*
2. *בניית עץ AVL עם מספרים מ0 עד m ועדכון ערכיו – O(m).*

*סה"כ: O(2log(n)+m) =* ***O(log(n)+m)****.*

*3 .זמן של* ***RemoveCourse(void\* DS, int courseID)*** *–*

*נסתכל על המקרה הגרוע ביותר:*

*יש במערכת n קורסים, כאשר לקורס שמעוניינים למחוק יש m שיעורים, ובמערכת יש סה"כ M שיעורים.*

1. *מציאת צומת בעץ AVL עם n צמתים: O(log(n)).*
2. *סריקת מערך עם m תאים, כמספר השיעורים בקורס, כאשר לכל שיעור בקורס, במקרה הגרוע ביותר, יש זמן צפייה שונה – O(m).*

*ב.1. ניגשים לתא במצביע שבמערך – O(1).*

*ב.2. מחפשים את צומת הקורס בעץ AVL עם לכל היותר n צמתים: O(log(n)).*

*ב.3. מבצעים מחיקה של השיעור מעץ AVL + שינוי המצביע במערך – O(1). (אם כל השיעורים תחת אותו זמן, מדובר בO(m), אך הסריקה log(n) תתבצע פעם אחת בלבד).*

*סה"כ:* ***O(mlog(n)).*** *מתקיים (ונתון) בהכרח כי , לכן בהכרח מדובר ב: O(mlog(M)).*

* *בכל מקרה אחר, סיבוכיות הזמן דווקא קטנה יותר: בזמן צפייה זהה לכל השיעורים בקורס יתקבל O(m) פעולות – m פעולות מחיקה + m-1 פעולות בסדר גודל O(1).*
* *במקרה שבו כל השיעורים תחת קורס אחד, הסריקה תתבצע בסיבוכיות log(M) כיוון שישנו רק קורס אחד במערכת ולמצוא אותו לוקח O(1).*

*4.זמן של* ***WatchClass(void\* DS, int courseID, int classID, int time)***  *–*

*נתייחס למקרה הגרוע ביותר: מעלים את ערכו של t כך שכל הערכים מערך התא הנוכחי ועד לערך התא הנוכחי + t קיימים ברשימה המקושרת עבור קורסים שונים. בנוסף, כמעט כל השיעורים במערכת שייכים לקורס הנתון, ונמצאים בעץ יחד עם השיעור המבוקש (בכל מצב אחר, סעיף ב להלן מקבל ערך קטן אף יותר).*

*א. מציאת צומת בעץ AVL עם n צמתים (קורה לכל היותר 3 פעמים) – O(3log(n)).*

*ב. מציאת שיעור בעץ AVL עם לכל היותר m שיעורים – O(log(m)).*

*ג. מחיקת/יצירת צומת/עץ ריק/תא ברשימה מקושרת – O(1).*

*ד. מעבר על t תאים עד למציאת המקום המתאים לשיעור ברשימה המקושרת – O(t).*

*סה"כ: O(3log(n)+log(m)+t), מתקיים כי M>m וגם M>n ולכן:*

*5.זמן של****TimeViewed(void\* DS, int courseID, int classID, int \*timeViewed)***  *–*

*א. מציאת צומת בעץ AVL עם n צמתים – O(log(n)).*

*ב. גישה לאיבר במערך + עדכון מצביע – O(1).*

*סה"כ:* ***O(log(n)).***

*6.זמן של*

***GetMostViewedClasses(void\* DS, int numOfClasses, int \*courses, int \*classes) –***

*\*\* כאן הערך m = numOfClasses. \*\**

*א. גישה למצביע + יצירת משתנים – O(1).*

*ב. ביצוע לולאה כאשר עוצרים לאחר m פעולות:*

*ב.1. גישה למצביע (פעמיים) וגישה לתא ברשימה מקושרת – O(1).*

*ב.2. עידכון מערכי הנתונים courses וclasses – O(1).*

*ב.3. ממשיכים בסיור inorder, סדר גודל של כמות האיברים שהדפסנו. – O(1).*

*סה"כ:* ***O(m)****.*

*7. זמן של* ***Quit(void\*\* DS) -***

*כפי שצויין במימוש הפונק' – עלינו למחוק במקרה הגרוע ביותר 3M איברים מהרשימה המקושרת + M+n איברים מהעץ הראשי – סך הכל* ***O(4M+n) = O(M+n).***

*8 .מקום -*

*במקרה הגרוע ביותר, מתקיים שזמני הצפייה בכל השיעורים במערכת שונים, לכן, אנחנו משתמשים ב:*

*א. עץ AVL\_v2 עם n צמתים – O(n).*

*ב. בכל צומת בעץ יש מערך בגודל של מס' השיעורים בקורס (כלל השיעורים מהווים M) - O(M).*

*ג. רשימה מקושרת עם לכל היותר M תאים שונים – O(M).*

*ג.1. במקרה הגרוע ביותר– יש לכל היותר קורס אחד שבו שיעור אחד – O(2M).*

*הסבר ג.1.: בכל מצב, יש במבנה הנתונים של זמני הצפייה לכל היותר M שיעורים (כי אין כפילויות) ולהם יכולים להתאים לכל היותר M קורסים שונים.*

*סה"כ: O(4M+n) = O(M+n).*