פרויקט במבנה נתונים – סקירה:

אז הפרויקט מתאר את החברה "boom" שזה כמו זום תכלס.
לכל קורס יש מספר מזהה, ויש לו מספר מסוים של שיעורים(למשל 13 שיעורים) שניתן לצפות בהם,
וכל סטודנט בעצם יכול לבחור זמן מסוים לצפות בשיעור מסוים של קורס מסוים (ממש מערכת הקלטות שיעורים רגילה)
תיאור הפונקציות:

.1

```
StatusType AddCourse (void *DS, int courseID, int numOfClasses)
```

סיבוכיות זמן: $O(\log(n) + m)$ במקרה הגרוע, כאשר n הוא מספר הקורסים במערכת בזמן הפעולה ו-m הוא numOfClasses.

מוסיפים קורס עם coursedID שבו יש numOfClasses של שיעורים.

.2

```
StatusType RemoveCourse(void *DS, int courseID)
```

סיבוכיות זמן: $O(mlog(M))$ במקרה הגרוע, כאשר M הוא מספר ההרצאות במערכת בזמן הפעולה ו-m הוא מספר ההרצאות של הקורס שהסרנו (m הוא 1 אם אין קורס עם המזהה הנתון. ניתן להניח כי M גדול ממש ממספר הקורסים).

הסרת של קורס מהמערכת.

.3

```
StatusType WatchClass(void *DS, int courseID, int classID, int time)
```

`

סיבוכיות זמן: $O(\log(M) + t)$ במקרה הגרוע, כאשר M הוא מספר ההרצאות במערכת בזמן הפעולה ו-t הוא time.

הוספת צפייה של סטודנט כלשהו(לא ידוע מי זה) בקורס מספר coursed של סך הכל זמן time בשיעור מספר classID של אותו קורס

.4

```
StatusType TimeViewed(void *DS, int courseID, int classID, int *timeViewed)
```

<u>סיבוכיות זמן</u>: $O(\log(n))$ במקרה הגרוע, כאשר n הוא מספר הקורסים במערכת בזמן הפעולה.

חישוב סך הכל זמני הצפייה של שיעור מספר classID עבור קורס מספר courseID וכתיבת הערך לתוך המצביע timeViewed.

.5

```
StatusType GetMostViewedClasses(void *DS, int numOfClasses, int *courses, int *classes)
```

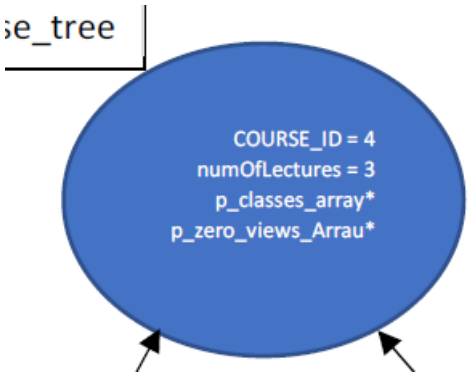
<u>סיבוכיות זמן</u>: $O(m)$ במקרה הגרוע, כאשר m הוא מספר השיעורים המבוקשים (numOfClasses).

שימו לב שהמערכים כבר מוקצים בגודל המתאים, יש רק למלא אותם.

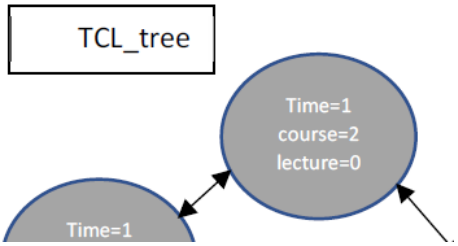
מחזירים סך הכל numOfClasses(כלומר יכול להיות 1 או 2 או יותר..) של שיעורים כך שהשיעורים האלה יש את סך זמן הצפיות הגדול ביותר (כלומר ייכול להיות שסך זמן הצפיות הגדול ביותר הוא למשל 100, ויש 3 שיעורים, כל אחד של קורס כלשהו, שלהם יש את הערך הזה). את ערך החזרה מחזירים בפוינטרים בהתאמה – כניסה ראשונה זה מספר קורס ומספר השיעור שלו וכן הלאה.

<u>פתרון יבש של הפרויקט:</u>

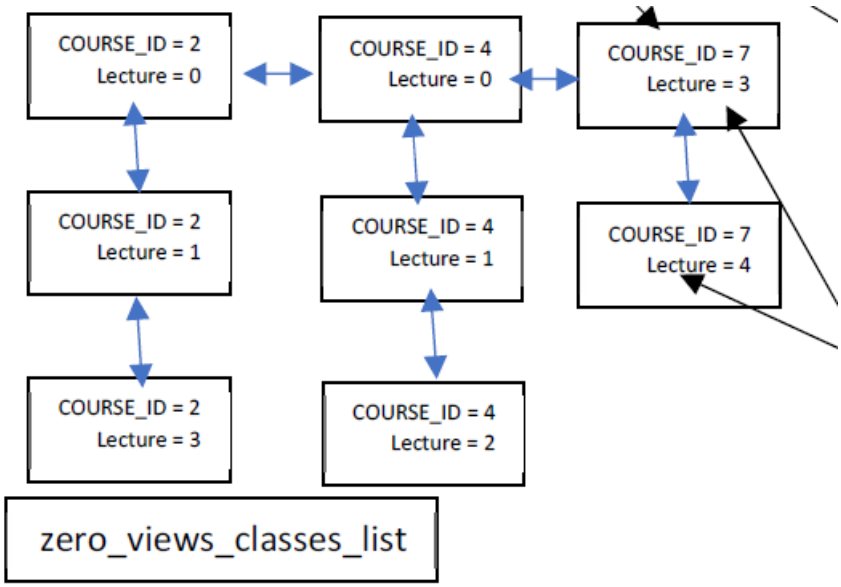
אז אנחנו החזקנו בעצם 2 עצים:
עץ ראשון זה עץ AVL שהמפתחות בו הם הcoursID.
צומת בעץ נראתה באופן הבא:



כאשר **numOfLectures** זה מספר השיעורים שיש לקורס,
p_classes_array זה מערך של אוביקטים מסוג TCL:



כאשר Time זה סך כל הזמן של השיעור lecture בקורס מספר course.
ולבסוף יש גם את p_zero_views_array שזה רשימה מקושרת – לכל קורס יש רשימה מקושרת של כל השיעורים שאין בהם בכלל צפיות:



זוהי רשימה מקושרת דו כיוונית, וכל איבר ברשימה המקושרת הוא ראש רשימה מקושרת דו כיוונית.

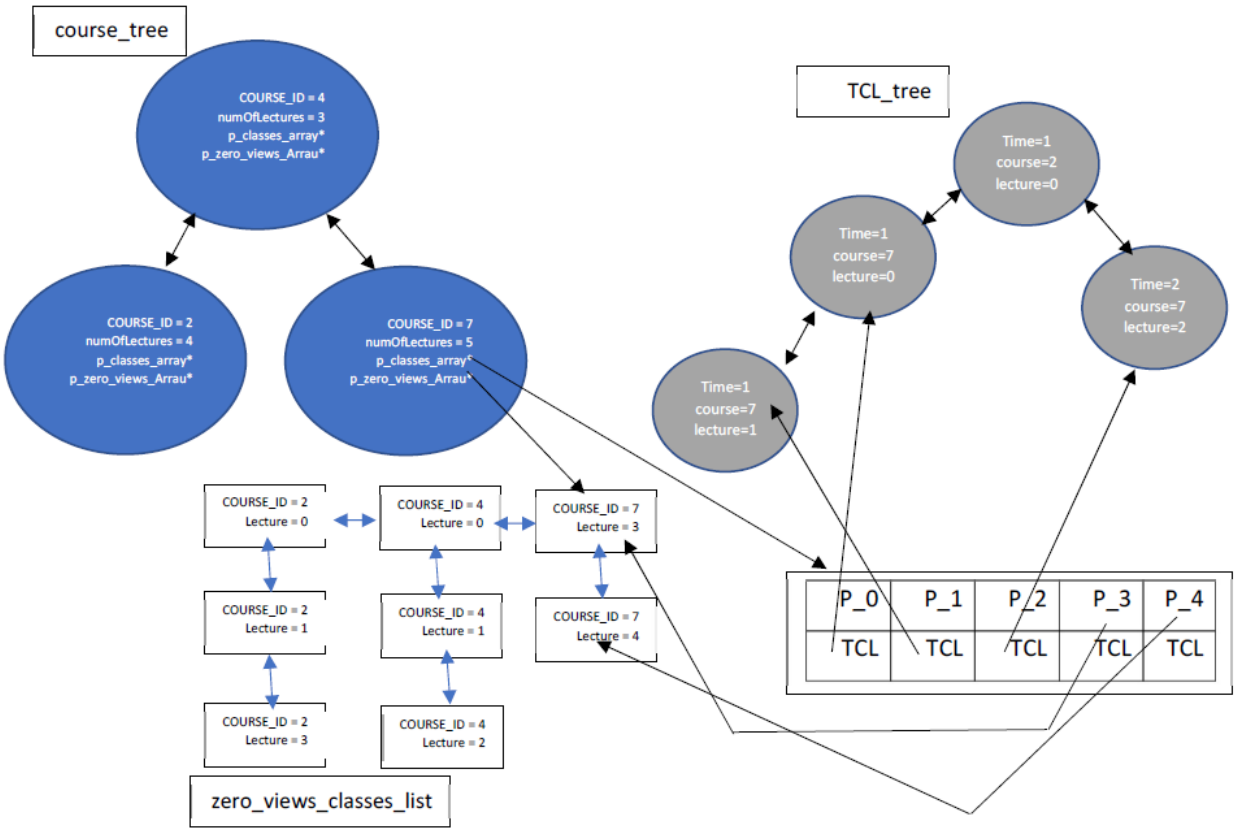
סך הכל ביחד זה נראה כך:

מבנה נתונים מכיל:

1.course_tree – עץ חיפוש שהמפתחות בו הם course_ID.
בכל צומת course נשמרים:
lectures – פוינטר למערך של ההרצאות של הקורס.
כך שלכל הרצאה יהיה אובייקט מסוג TCL ופוינטר כך שבהתחלה הוא יצביע לnode המתאים לlecture זה ברשימה המקושרת zero_views_lectures_list. לאחר שההרצאה תיצבור את הצפייה הראשונה, נמחק את הnode המתאים ברשימה של ההרצאות ללא צפיות ונעדכן את הזמן הצפייה בTCL של ההרצאה זאת.
p_zero_views_classes – פוינטר לראש הרשימה המקושרת של ההרצאות עם 0 זמן צפייה של course.
courseID – מספר הקורס
numOfLectures – מספר ההרצאות בקורס

2.tcl_tree(time,course,lecture) – עץ חיפוש שהמפתחות בו הם time,course,lecture, כך שהחשיבות למפתח היא לפי הסדר משמאל לימין(כלומר, קודם time אחכ course ואז lecture).
בכל צומת tcl נשמר אובייקט עם 3 ערכים מטיפוס int – שהם (time,course,lecture).
בנוסף העץ יכיל MaxNode שיצביע לnode עם הערך הגדול ביותר.

3.zero_views_classes_list - רשימה מקושרת דו כיוונית שכול node בה הוא ראש של רשימה מקושרת דו כיוונית,
כך שעל כל node מצביע p_zero_views_classes, כלומר, הפוינטר של הcourse המתאים.
מכל Node יוצאת רשימה מקושרת דו כיוונית כך שעל כל node בתוכה מצביע ההרצאה המתאימה, כלומר הפוינטר של הlecture המתאים.



מימוש הפונקציות:

.1

StatusType AddCourse (void *DS, int courseID, int numOfClasses)

סיבוכיות זמן: $O(\log(n)+m)$ במקרה הגרוע, כאשר n הוא מספר הקורסים במערכת בזמן הפעולה ו-m הוא numOfClasses.

יצירה של צומת חדש לעץ הcourseTree כולל כל השדות שלו, ובין היתר גם יצירה של רשימה מקושרת באורך numOfClasses שכל צומת מייצג שיעור של הקורס אשר אין בו צפיות, והוספה של הרשימה לzero_view_classes_list

.2

StatusType RemoveCourse(void *DS, int courseID)

סיבוכיות זמן: $O(mlog(M))$ במקרה הגרוע, כאשר M הוא מספר ההרצאות במערכת בזמן הפעולה ו-m הוא מספר ההרצאות של הקורס שהסרנו (m הוא 1 אם אין קורס עם המזהה הנתון. ניתן להניח כי M גדול ממש ממספר הקורסים).

מסירים מעץ הcourse_Tree את הרשומה המתאימה, ומסירים מעץ הTCL את כל השיעורים של הקורס. עבור כל שיעור שמסירים מהעץ הTCL עולה log(M)

.3

StatusType WatchClass(void *DS, int courseID, int classID, int time)

סיבוכיות זמן: $O(\log(M)+t)$ במקרה הגרוע, כאשר M הוא מספר ההרצאות במערכת בזמן הפעולה ו-t הוא time.

מוצאים את הקורס בעץ הcourseTree ואז נכנסים למערך השיעורים של הקורס.
אם לשיעור יש צפייה כלשהי, אז ניגש לצומת הTCL שלה ואז נגדיל את הtime של הצומת המתאים.
אחרת, צריך להוציא את הצומת מהרשימה המקושרת של הקורס, ולהוסיף צומת TCL חדש לעץ הTCL.

.4

StatusType TimeViewed(void *DS, int courseID, int classID, int *timeViewed)

סיבוכיות זמן: $O(\log(n))$ במקרה הגרוע, כאשר n הוא מספר הקורסים במערכת בזמן הפעולה.

מוצאים קודם את הצומת המתאים בcourse_Tree.
אחכ ניגשים למערך השיעורים שנמצא כשדה בצומת שמצאנו.
עבור כל כניסה במערך, או שהוא מצביע על צומת שנמצא ברשימה המקושרת של הצמתים של

שיעורים להם אין עוד צפיות, או שהוא מצביע לצומת שנמצאת בעץ הTCL ואז ניגש לצומת וניקח את הערך שיש שם.

.5

StatusType GetMostViewedClasses(void *DS, int numOfClasses, int *courses, int *classes)

סיבוכיות זמן: $O(m)$ במקרה הגרוע, כאשר m הוא מספר השיעורים המבוקשים (numOfClasses).

שימו לב שהמערכים כבר מוקצים בגודל המתאים, יש רק למלא אותם.

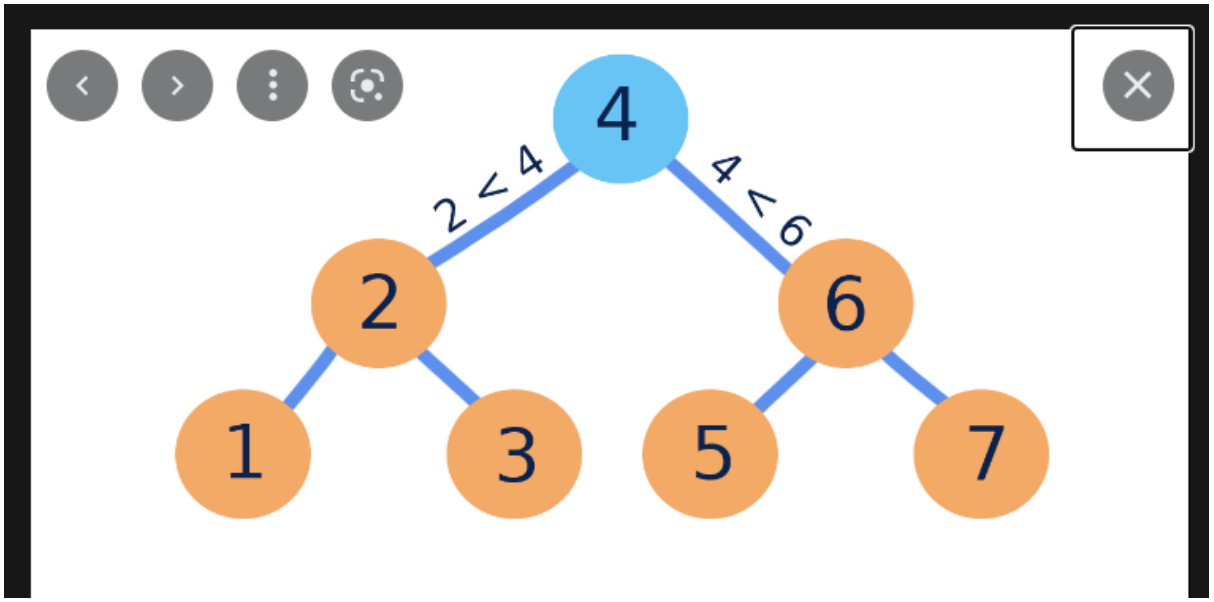
במהלך כל הפרויקט, שומרים מצביע שנקרא p_max_tcl אשר מחזיק בכל רגע נתון את הצומת הימני ביותר בעץ הTCL.
אז על מנת להחזיר את הnumOfClasses(זה מספר, למשל 10) השיעורים הנצפים ביותר, בעזרת המצביע אנחנו נתחיל ממנו סיור בעץ שהוא בעצם Inorder הפוך – הכוונה היא שקודם הולכים לבן הימני, אז לאבא, ואז לבן השמאלי.
מימוש:

```
def printInorder(root):
    if root:
        # First recur on left child
        printInorder(root.right)

        # then print the data of node
        print(root.val),

        # now recur on right child
        printInorder(root.left)
```

למשל עבור:



אז אם מחזיקים מצביע לצומת עם הערך 7, אז ריצה של inorder הפוך תניב:

7
6
5
4
3
2
1

הטריק פה הוא להחזיר את הצמתים בסדר יורד שהם עם הכי הרבה צפיות.
אז מתחילים מהמקסימילי.

```
//Returns the next smaller node in order
Node* getNextSmaller(){
        if(left != 0)
                return left->getMostRight();
        for(Node* node = this; node->parent != 0; node = node->parent){
                if(node == node->parent->right){
                        return node->parent;
                }
        }
        return 0;
}
```

ואז אם יש לו בן שמאלי, יורדים אליו, ואז לוקחים הכי הרבה ימינה שאפשר.
אם אין לו בן שמאלי, זה אומר שאין לו בכלל ילדים (כי התחלנו מהמקסימום) אז רוצים למצוא את האבא הראשון שלו, כך שהוא בן ימני של מישהו.