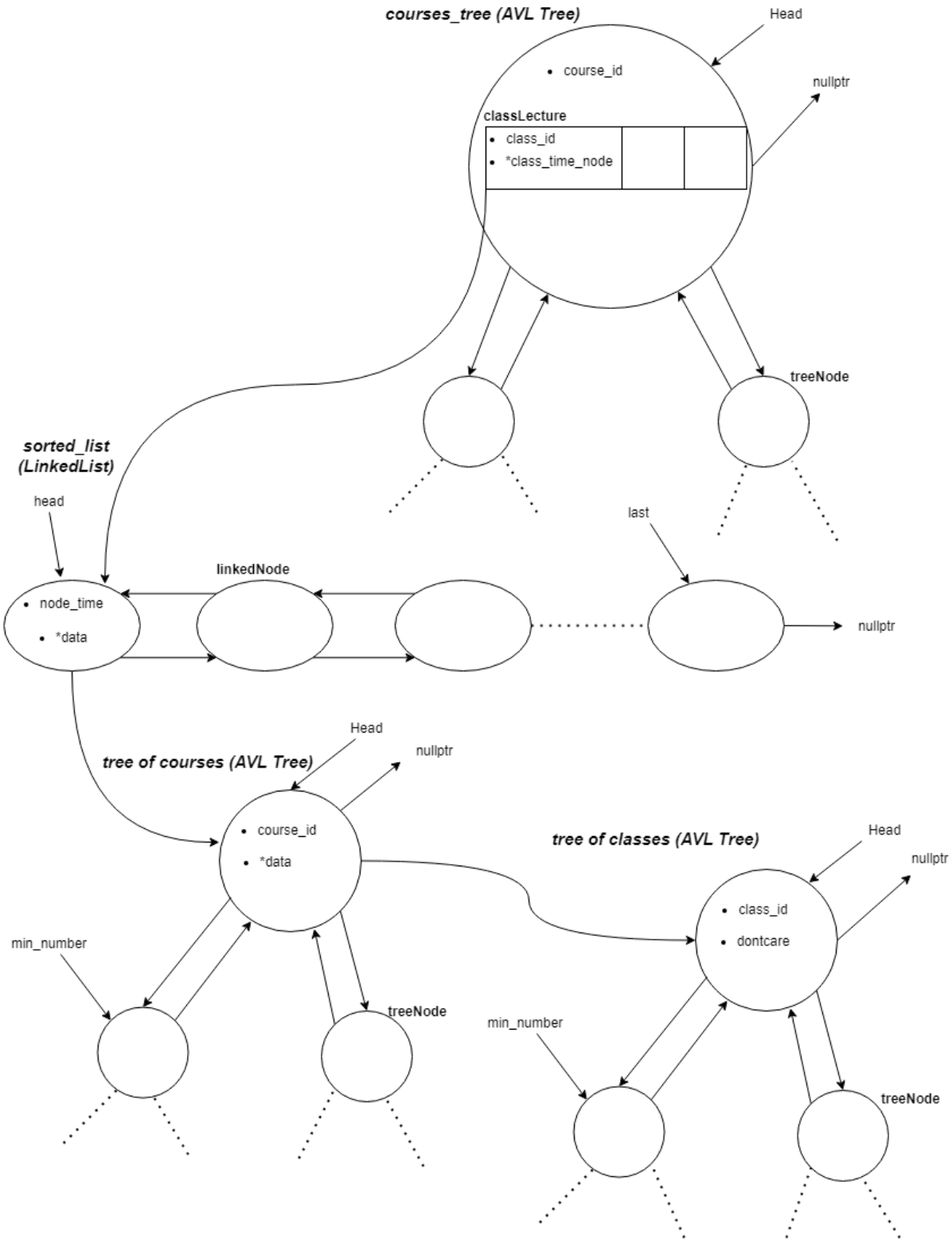
תרשים למבנה הנתונים בו השתמשנו:



חלק יבש:

תיאור המבנים למימוש CoursesManager:

- עץ AVL גינרי בשם courses_tree שמתאר את הקורסים במערכת.
- מבנה נתונים בשם classLecture השתמשנו בו כData לעץ (יפורט בהמשך).
- רשימה מקושרת דו-כיוונית בשם sorted_list שהשתמשנו בה לשמירת זמני הצפייה עבור כל ההרצאות בכל הקורסים ומצביעה לעץ AVL שמייצג קורסים שהוא בעצמו מצביע לעץ AVL שמייצג הרצאות/שיעורים של קורס זה (גם יפורט בהמשך).

מבנה עץ AVL כללי:

עץ ה AVL מכיל שני משתנים:

1) nodesAmount המכיל את מספר הtreeNodes בעץ, כלומר גודל העץ.
2) Head, שהוא פוינטר לשורש העץ, כך ש Head הוא מטיפוס *treeNode שמכיל בתוכו מצביע לבן השמאלי, מצביע לבן הימני, מצביע להורה. בנוסף בכל treeNode יש מצביע ל- Key ול- Data של הצומת, בנוסף למשתנים פנימיים לצורך איזון העץ, כמו BF(balance factor) וHeight שהוא הגובה.
3) min_number שמייצג את המפתח המינימלי בעץ (רק בעץ הראשי courses_tree אנחנו לא נדאג לעדכן אותו).

העץ תומך בבניית עץ ריק בסיבוכיות זמן O(1), ופעולת הכנסה והוצאה של איברים בסיבוכיות זמן O(logn) כאשר n הוא מספר האיברים בעץ.

סיבוכיות המקום של העץ היא לכל היותר O(n) כאשר n הוא מספר הצמתים (ברור).

מבנה הנתונים courses_tree:

עץ AVL גנרי שבו שמרנו את המידע עבור כל הקורסים במערכת, הוא מכיל:

1) המספר המזהה של הקורס ששמור כ Key של צומת בעץ מסוג int.
2) מערך של מצביעים כך שכל איבר במערך מצביע לחולייה ברשימה המקושרת הדו-כיוונית sorted_list והוא שמור כ Data של צומת בעץ, והמערך מסוג classLecture.
3) רשימה מקושרת דו-כיוונית sorted_list, שמכילה את זמני הצפיות של כל ההרצאות במערכת באופן ממוין כמפתחות ומצביע לעץ קורסים שמכיל בתוכו עץ הרצאות כמידע (יפורט בהמשך).

אתחול מבנה הנתונים מתבצעת בסיבוכיות זמן O(1) כאשר נחזיר מצביע לעץ ריק או NULL במקרה של כשלון. הריסתו של מבנה הנתונים מתבצעת בסיבוכיות זמן O(M) במקרה הגרוע, כאשר M הוא מספר ההרצאות הכולל של כל הקורסים במערכת (יפורט למטה ובהמשך).

סיבוכיות המקום של המבנה היא לכל היותר O(n+M) כאשר n הוא מספר הקורסים הכולל ו- M הוא מספר ההרצאות הכולל של כל הקורסים במערכת, כי:

- העץ courses_tree הוא בגודל n כאשר n הוא מספר הקורסים הכולל.
- הגודל של מערך המצביעים (שקיים לכל קורס) שמסוג classLecture הוא m, והסיבוכיות מקום של classLecture הוא O(1) (יפורט בהמשך) ולכן סיבוכיות מקום של מערך זה הוא O(m).
- סיבוכיות המקום של רשימה דו-כיוונית הוא O(M) במקרה הכי גרוע (כי יכול להיות שלכל הרצאה יש זמן צפייה נפרד משלה ולכן גודל הרשימה הוא גדול מאוד ושווה ל M).
- כל חולייה ברשימה מצביעה לעץ קורסים שהוא יכול להיות בגודל n לכל היותר וכל צומת בעץ מצביעה על עוד עץ של הרצאות ששייכות לקורס זה ולכן כל עץ כזה יכול להיות בגודל m כאשר m הוא מספר ההרצאות של קורס זה.

לכן סיבוכיות מקום של כל המבנה היא O(M) כאשר הייתה הנחה נסתרת ש n*m = M כלומר כי מספר הקורסים הכולל (n) כפול מספר ההרצאות של כל קורס בממוצע (m) הוא מספר כל הקורסים במערכת בממוצע (M).

מבנה הנתונים classLecture:

מבנה שבו שמרנו את מידע עבור הרצאה מסוימת, הוא מכיל:

1) class_id, המספר המזהה של ההרצאה.
2) constant_number_of_class, מספר כל ההרצאות ששייכות לאותו קורס שהרצאה זו שייכת.
3) class_time_node, מצביע לחולייה ברשימה המקושרת הדו-כיוונית sorted_list.

אתחול והריסת מבנה הנתונים מתבצעות בסיבוכיות זמן O(1) , סיבוכיות המקום היא גם O(1).

מבנה הנתונים LinkedList (רשימה מקושרת דו-כיוונית גנרית):

בנינו רשימה מקושרת דו-כיוונית שמכילה:

1) מצביע לNode שקראנו לו head שמצביע לתחילת הרשימה.
2) מצביע לNode שקראנו לו last שמצביע לסוף הרשימה.
3) size_of_list שמכיל את מספר הNodeים ברשימה.

כל איבר ברשימה הוא Class בשם linkedNode שמכיל:

1) מצביע לאיבר השמאלי.
2) מצביע לאיבר הימני.
3) data של הlinkedNode שהוא מטיפוס גנרי.

סיבוכיות הזמן של הרשימה:

- פעולת אתחול רשימה ריקה O(1)
- הכנסת איבר לרשימה לוקחת O(M) כאשר M הוא מספר ההרצאות הכולל במערכת(גודל הרשימה במצב הגרוע ביותר).
- פעולת הוצאה לוקחת O(1) כאשר הפונקציה מקבלת מצביע לNode שצריך למחוק. (אנו מניחים שהאיבר אכן קיים ברשימה.

סיבוכיות המקום של הרשימה הוא O(M) כאשר M הוא מספר האיברים ברשימה(מספר ההרצאות הכולל בכל הקורסים).

מבנה הנתונים sorted_list:

היא רשימה מקושרת דו-כיוונית שהשתמשנו בה לצורך שמירת זמני הצפייה של כל הרצאה באופן ממוין.

1) node_time, מזהה של linkedNode והוא זמן צפייה של כל ההרצאות שמצביעות על חולייה זו.
2) data*, הוא מצביע לעץ AVL גנרי שמכיל את כל הקורסים שיש בהם קורס אחד לפחות עם זמן צפייה זה. ה Key של עץ זה הוא מספר הקורס וה Data* הוא מצביע לעץ AVL גנרי שמכיל את כל ההרצאות עם זמן צפייה זה. ה Key בעץ זה הוא מספר ההרצאה ו Data הוא לא חשוב.

על סיבוכיות הזמן ומקום של הרשימה/עץ AVL ופעולותיו מפורט לעיל.

```
class CoursesManager{
    public:

    AVLTree<int, classLecture<linkedNode<AVLTree<int, AVLTree<int, int>>>>> *courses_tree;
    LinkedList<AVLTree<int, AVLTree<int, int>>> *sorted_list;

    CoursesManager();
    ~CoursesManager();
```

הסבר על מבנה הנתונים CoursesManager:

הClass בשם CoursesManager מכיל כל מיני שדות לצורך מימוש הפעולות הנדרשות למבנה הנתונים:

1) עץ AVL בשם courses_tree שהשתמשנו בו כדי לשמור את כל המידע עבור הקורסים שנמצאים במערכת, כל treeNode בו מכיל Key מסוג אינטג'ר שמכיל את מזהה הקורס, Data שהיא מערך מצביעים מסוג classLecture, שמכיל את כל המידע עבור קורס (והרצאותיו) שנמצא במערכת כפי שמפורט לעיל.
2) רשימה מקושרת דו-כיוונית בשם sorted_list שהשתמשנו בה כדי לשמור את הסדר בין ההרצאות שנמצאים במערכת לפי זמן הצפייה שלהם (יפורט בהמשך).

אתחול הCoursesManager מתבצע בסיבוכיות זמן O(1), כי הבנאי הדיפולטיבי קורא לבנאי הדיפולטיבי של עץ הAVL, סיבוכיות הזמן של בנאי זה היא O(1) כפי שמפורט למעלה.

סיבוכיות המקום של מבנה הנתונים היא O(n+m) כאשר n הוא מספר הקורסים הנמצאים בו וm הוא מספר ההרצאות הכולל שנמצאות בו.

נסביר נכונות: סיבוכיות המקום של עץ היא O(n) וסיבוכיות המקום של כל מערכי המצביעים O(m) וסיבוכיות המקום של הרשימה המקושרת הדו-כיוונית היא גם O(m) וסיבוכיות במקום של עץ העצים היא לכל היותר O(n+m) ולכן:

$$n + 2 * O(m) + O(n + m) = O(n + m)$$

הסבר על הפעולות הנדרשות:

void * Init()

מאתחל מבנה נתונים ריק.

אתחלנו דרך הקצאה דינמית למבנה ריק, לכן נקבל שסיבוכיות הזמן הנדרשת של הפונקציה זהה לסיבוכיות הזמן לאתחול CoursesManager ולכן זה שווה ל O(1) (סה"כ מקצים זיכרון לעץ ריק ורשימה מקושרת ריקה).

סיבוכיות מקום: O(1), כי מקצים בסה"כ O(1) מצביעים, כל מצביע דורש O(1) מקום.

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

בדקנו ערכי שגיאה, ותקינות פרמטרים. בסיבוכיות O(1).

בדקנו האם קיים כבר קורס עם אותו מספר מזהה, בסיבוכיות זמן O(logn) (עבור החיפוש בעץ AVL).

אחרי זה הקצאנו מצביע לצומת שמתאר קורס עם מזהה courseID, ומערך בגודל מספר ההרצאות numOfClasses. בפעולה זו אנו יוצרים צומת לעץ הAVL , מערך ולכל היותר חולייה לרשימה המקושרת הדו-כיוונית, שמתבצעת בסיבוכיות זמן ומקום O(numOfClasses) כלומר O(m).

אחרי שהקצאנו את הקורס עם כל המידע שצריך עבורו, עכשיו אנו נוסיף אותו לעץ הAVL ששמו courses_tree שנמצא בCoursesManager כפי שמפורט לעיל, בעץ זה הסידור מתבצע לפי מזהה מספרי הקורסים השונים, כלומר courseID. פעולה זו מתבצעת בסיבוכיות זמן O(logn) (הוספת treeNode לעץ AVL) וסיבוכיות מקום O(1).

לאחר מכן, בודקים אם החולייה שמחזיקה את הזמן צפייה אפס (זמן צפייה הוא נחשב כ Key ברשימה) נמצאת כבר ברשימה sorted_list. אם כן, נאתחל את מצביעי איברי המערך שבגודל numOfClasses שנמצא בתוך צומת הקורס הזה כ Data, שיצביעו על החולייה עם זמן צפייה אפס שנמצאת ברשימה המקושרת וזה מתבצע בסיבוכיות זמן O(numOfClasses) שזה שווה ל O(m) בהנחה ש numOfClasses = m.

אחרי זאת נבנה עץ AVL שכל צומת בו המפתח שלו הוא מספר ההרצאה classID והמידע הוא מצביע לעוד עץ של הרצאות ששייכות לקורס זה עם אותו זמן צפייה. אם כבר קיים עץ זה אז נחפש מקום להוסיף אותו בסיבוכיות זמן O(logn) כאשר n הוא גודל עץ הAVL ההוא בעת ההוספה. מדובר בזמן צפייה אפס כי רק עכשיו הוספנו את הקורס. ולכן בהינתן מערך ההרצאות/המצביעים נבנה עץ AVL מגודל numOfClasses(כי כל ההרצאות טרם נצפו) בסיבוכיות זמן O(m) = O(numOfClasses) ע"י האלגוריתם הבא:

1) נשיג את אמצע המערך ונהפוך אותו לשורש העץ.

2) רקורסיבית נעשה את זה עבור החצי השמאלי והחצי הימיני של המערך.

א) נשיג את אמצע החצי השמאלי ונהפוך אותו לבן השמאלי של השורש שנוצר בשלב 1.

ב) נשיג את אמצע החצי הימני ונהפוך אותו לבן הימיני של השורש שנוצר בשלב 1.

חשוב לציין כי אלגוריתם זה מתאפשר לעבוד בסיבוכיות זמן זו כיוון שמערך ההרצאות ממויין לפי מספר הקורס class_id.

כל הפעולות בפונקציה זו מתבצעות בסיבוכיות זמן O(logn+m) ((O(logn) עבור הכנסת איברים לעץ, O(m) עבור אתחול מערך המצביעים שמסוג classLecture, O(m) עבור בניית עץ ההרצאות ע"י האלגוריתם שהצגנו לעיל), וסיבוכיות מקום O(m) (הקצאת מערך ההרצאות בגודל m והקצאת מקום עבור כלm צמתי העץ).

נראה נכונות סיבוכיות זמנו של האלגוריתם שהצגנו(שמיוצג בפעולה sortedArrayToBST_aux (start, end)):

הנוסחה הרקורסיבית עבור אותה פונקציה היא:

$$T(m) = 2T(m/2) + C$$

$$T(m) \; --> \; Time\ taken\ for\ an\ array\ of\ size\ m$$

$$C \; --> \; Constant\ (Finding\ middle\ of\ array\ and\ linking\ root\ to\ left\ and\ right\ subtrees\ take\ constant\ time)$$

לפי שיטת המאסטר, $f(m) = C$, a = b = 2. ולכן עבור $\epsilon = 1$ מתקיים:

$$C = f(m) = O\left(m^{\log_b a - \epsilon}\right) = O\left(m^{\log_2 2 - \epsilon}\right) = O(m^{1-1}) = O(1)$$

ואכן זה נכון כי פונקציה קבועה היא $O(1)$. ולכן זה מתאים למקרה 1 בשיטת המאסטר ולכן סיבוכיות זמן פעולה/אלגוריתם זה הוא:$T(m) = O\left(m^{\log_b a}\right) = O\left(m^{\log_2 2}\right) = O(m)$.

<u>StatusType RemoveCourse(void *DS, int courseID)</u>

בדקנו ערכי שגיאה, ותקינות פרמטרים בסיבוכיות זמן O(1).

אחרי זה חיפשנו את הקורס בעץ courses_tree לפי מזהה מספר הקורס הנתון (courseID). פעולה זו מתבצעת בסיבוכיות זמן O(logn). אם הקורס אכן קיים במערכת, נסרוק את המערך ההרצאות שלו, ועבור כל

איבר במערך ניגש לחולייה ברשימה המקושרת sorted_list אשר הוא מצביע עליה. אחר כך ניגש לעץ הקורסים שחולייה זו מצביעה עליו ונחפש את הקורס בסיבוכיות זמן O(logn) (מובטח כי נמצא אותו אחרת לא היינו מוצאים אותו בעץ הראשי courses_tree). אחר כך ניגש לעץ ההרצאות שצומת קורס זה מצביעה עליו ונחפש את מספר ההרצאה עם המזהה class_id המתאים למספר האיטרציה של הלולאה שנסרק בה המערך (מובטח כי נמצא אותו כי אנחנו לא מוציאים הרצאה בודדת). בעץ זה של ההרצאות יש לכל היותר m צמתים, שהם בעצם כל ההרצאות בקורס, ולכן המחיקה תתבצע במצב הגרוע ביותר בסיבוכיות זמן O(logm). אם אחרי מחיקתו נקבל שצומת הקורס בעץ שמעליו מצביע על עץ ריק אז נמחק צומת זה ב O(1), ואם מחקנו את צומת של קורס זה וקיבלנו שגם החולייה שברשימה המקושרת מצביעה על עץ ריק אז גם אותה נמחק ב O(1). אחרי שעברנו על כל איברי המערך נסיר את צומת הקורס מהעץ הראשי courses_tree. כיוון שבפעולת delete אנחנו עושים שחרור לtreeNode שהקצנו, תתבצע קריאה להורס של צומת בעץ AVL, שהיא קוראת להורס של classLecture כי זהו טיפוס כל איבר במערך. ראינו קודם כי סיבוכיות זמן ההורס של classLecture היא O(1) ולכן סיבוכיות הזמן הכוללת של הריסת המערך כולו היא O(numOfClasses) כלומר O(m).

חשוב לציין כי אחרי הוצאת צומת הקורס מהעץ שהרשימה המקושרת sorted_list מצביעה עליו אנחנו נעדכן את min_number למפתח המינימלי ביותר. פעולה זו תתבצע ב O של גובה העץ כלומר O(log(m)) עבור עץ ההרצאות ו O(logn) עבור עץ הקורסים.

סיבוכיות של פונקציה זו היא אכן O(mlogm) כי ביצענו O(1) פעולות שמתבצעות בסיבוכיות זמן O(logn) ו-O(logm) (חיפוש ומחיקה מעץ AVL) תחת לולאה שעוברת על m איברים. ו O(1) פעולות שחרור שמתבצעות בסיבוכיות זמן O(numOfClasses) כלומר O(m).

ולכן סיבוכיות הזמן היא: (ברור כי מספר הקורסים n ≥ מספר ההרצאות m, וגם כי לפי אותה הנחה נסתרת מקודם: מספר ההרצות הממוצע בקורס m כפול סך כל הקורסים n שווה לסך כל ההרצאות במערכת M).

$$\log(n) + m(\log(n) + \log(m)) + m + \log(n) + \log(m)$$

$$\underset{\substack{n\le m \\ \text{חוקי log}}}{=} O\big(mlog(n*m)\big) \underset{n*m=M}{=} O(mlog(M))$$

לגבי סיבוכיות מקום אין הקצאות זיכרון או קריאות רקורסיביות, ולכן סיבוכיות המקום היא O(1).

<u>StatusType WatchClass(void *DS, int courseID, int classID, int time)</u>

בדקנו ערכי שגיאה, ותקינות פרמטרים ב O(1).

לאחר מכן חיפשנו את הcourseID בעץ courses_tree (O(logn)) אם לא מצאנו אז החזרנו שגיאה. אחרת:

ניגשנו להרצאה המתאימה במערך עם מזהה classID ועדכננו את זמן הצפייה בהתאם. אחר כך ניגשנו לחולייה בתוך הרשימה המקושרת sorted_list שאנו מצביעים עליה. אחר כך ניגש לעץ הקורסים שחולייה זו מצביעה עליו ונחפש את הקורס בסיבוכיות זמן O(logn) (מובטח כי נמצא אותו אחרת היינו קודם לפני חוזרים מהפונקציה עם ערך שגיאה). אחר כך ניגש לעץ ההרצאות שצומת קורס זה מצביעה עליו ונחפש את מספר ההרצאה עם המזהה classID המתאים. בעץ זה של ההרצאות יש לכל היותר m צמתים, שהם בעצם כל ההרצאות בקורס, ולכן המחיקה תתבצע במצב הגרוע ביותר בסיבוכיות זמן O(logm). אם אחרי מחיקתו נקבל שצומת הקורס בעץ שמעליו מצביע על עץ ריק אז נמחק צומת זה ב O(1), ואם מחקנו את צומת של קורס זה וקיבלנו שגם החולייה שברשימה המקושרת מצביעה על עץ ריק אז גם אותה נמחק ב O(1).

כעת נבדוק אם החולייה עם הזמן החדש כבר נמצאת בתוך הרשימה המקושרת sorted_list. אם כן, נבדוק אם כבר חולייה זו מצביעה על עץ לא ריק. אם כן נוסיף את הקורס עם המזהה courseID בסיבוכיות זמן O(logn). אחרת, נקצה מקום לעץ AVL חדש של הקורסים ב O(1) ונוסיף את הקורס ב O(1) כי הוא נמצא לבד בעץ. ואז נקצה מקום לעוד עץ AVL חדש של ההרצאות ב O(1) ונוסיף את ההרצאה עם המזהה classID לעץ ב O(1). אם החולייה עם הזמן החדש לא נמצאת בתוך הרשימה, אז ניצור חולייה חדשה עם הזמן החדש ונוסיף

אותה לרשימה ע"י הפעולה addNewNode(newNode) שפועלת בסיבוכיות זמן O(t) כאשר t הוא time שיכול להיות הזמן המקסימלי של הרשימה המקושרת sorted_list כך שבמקרה הגרוע סך כל ההרצאות האחרות יש להן את זמני הצפייה מ 0 עד t-1 ולכן נצטרך לעבור על כל הרשימה ולהוסיף את החולייה עם הזמן החדש.
אחר כך נקצה מקום לעץ AVL חדש של הקורסים ב O(1) ונוסיף לו את הקורס עם המזהה courseID בסיבוכיות זמן O(1) כי הוא לבד נמצא בעץ. ואז נקצה מקום לעוד עץ AVL חדש של ההרצאות ב O(1) ונוסיף את ההרצאה עם המזהה classID לעץ ב O(1). בהוספת כל צומת שהיא מתארת קורס לעץ המתאים, אנחנו תמיד מקפידים על עדכון מפתח הקורס המינימלי לערך המתאים שזה לוקח בפעולה זו במקרה הגרוע סיבוכיות זמן כגובה העץ, כלומר O(logm) + O(logn) כאשר n הוא מספר הקורסים הכולל במערכת, ו m הוא מספר ההרצאות עבור קורס courseID).

בסה"כ בצענו O(t)+ O(logm) + O(logn) בנוסף ל O(1) פעולות, ולכן סיבוכיות הזמן היא:

$$\log(n) + \log(m) + O(t) \underset{\text{חוקי log}}{=} O(log(n * m) + t) \underset{n*m=M}{=} O(log(M) + t)$$

השתמשנו באותה הנחה נסתרת מקודם: מספר ההרצות הממוצע בקורס m כפול סך כל הקורסים n שווה לסך כל ההרצאות במערכת M).

בפעולה זו ביצענו O(1) הקצאות, כך שכל הקצאה דורשת O(1) מקום, ולכן סיבוכיות המקום היא O(1).

<u>StatusType TimeViewed(void *DS, int courseID, int classID, int *timeViewed)</u>

קודם כל בדקנו ערכי שגיאה, ותקינות פרמטרים בסיבוכיות זמן O(1).

לאחר מכן חיפשנו את הcourseID בעץ courses_tree ((O(logn) אם לא מצאנו אז החזרנו שגיאה. אחרת:

ניגש למערך ההרצאות של קורס זה ונשיג את החולייה שאיבר המערך במקום classID מצביע אליה. בחולייה זאת מאוחסן משך הצפייה הכולל של שיעור classID של קורס courseID בשדה node_time.

פונקציה זו מתבצעת בסיבוכיות זמן O(logn). וסיבוכיות מקום O(1).

<u>StatusType GetMostViewedClasses(void *DS, int numOfClasses, int *courses, int *classes)</u>

בדקנו ערכי שגיאה, ותקינות פרמטרים בסיבוכיות זמן O(1).

כפי שפורט קודם, אנו שומרים את המפתח המינימלי בעץ (min_number).

כשנקראת פונקציה זו אנו ניגשים לכל חוליות הרשימה המקושרת sorted_list מהסוף להתחלה. כלומר מתחילים מהמצביע לסוף הרשימה (last) וניגשים data שהוא עץ קורסים ומתחילים לסרוק את העץ ממספר הקורס המינימלי ששמרנו קודם בסיור inorder, כלומר קודם כל מטפלים בבן השמאלי -> שורש -> בן ימיני. לפי סיור זה אנו מקבלים את מספרי הקורסים מסודרים מקטן לגדול. אחרי כך, אנו ניגשים לשדה data של אותו צומת/קורס ומקבלים בכך את עץ ההרצאות של קורס זה עם הזמן המתאים לזמן שמופיע בחולייה שבסוף הרשימה ומבצעים באופן דומה סיור inorder ומקבלים את מספרי ההרצאות מהקטן לגדול. עבור כל שיעור/הרצאה נשמור את המספר המזהה של הקורס שלו במערך courses והמספר המזהה של השיעור במערך classes כך שהמקומות בשני המערכים תואמים זה לזה. אם לא נמצא את כמות השיעורים המבוקשת נצא מעץ ההרצאות ונמשיך בסיור ה inorder בעץ הקורסים ונעשה את אותו דבר עבור הקורס בעל במספר הגדול יותר העוקב. ובכך נקבל את כמות השיעורים הכי נצפים שהוא יכול להיות לכל היותר numOfClasses, עבור כל קורס במערכת מסודרים מהכמות הגדולה לקטנה כנדרש.

פעולה זו מתבצעת בסיבוכיות זמן O(m) כאשר m הוא מספר ההרצאות במערכת, כי כל טיפול באיבר הנוכחי מתבצע בסיבוכיות O(1) ויש m_i איברים/שיעורים בכל עץ שיעורים עבור כל קורס i במערכת, ועל כל איבר

מהם עוברים בדיוק פעם אחת, ולכל היותר נעבור על כל השיעורים בכל הקורסים. ולכן סיבוכיות הזמן היא O(m), כי $\sum_i(m_i) = m$.

סיבוכיות המקום של הפונקציה הוא O(logm) כגובה העץ כלומר העומק של הרקורסיה (בסיור inorder שממומש תוך שימוש ברקורסיה ובהנחה ש $n \leq m$).

void Quit(void **DS)

נמיר את DS מ void*- ל CoursesManager* ע"י static_cast וזה גם קורה בזמן O(1), לאחר מכן מבצעים delete ואז:

נקראים הDestructors של האובייקטים בתוך CoursesManager שהם:

1) העץ courses_tree שמכיל n איברים, כדי לשחרר אותו, צריכים סיבוכיות זמן O(n+m) כאשר n הוא מספר הקורסים ו m הוא מספר ההרצאות הכולל במערכת. (הDATA עבור כל treeNode בעץ היא מסוג classLecture. סיבוכיות ההורס של classLecture היא O(1) הKey עבור הtreeNode הוא מטיפוס int ולכן לא צריך שחרור, בהנחה שבמערכת יש n הרצאות, וכיוון שצריך לעבור על כל הרצאה בדיוק פעם אחת, נקבל שהסיבוכיות הכוללת היא $O(n + \sum_{i=1}^{n} k_i)$ כאשר k_i הוא מספר ההרצאות בקורס ה-i, ולפי הנתון ש $\sum_{i=1}^{n} k_i = m$ נקבל כי סיבוכיות הזמן הכוללת היא O(n+m).
2) שחרור הרשימה המקושרת sorted_list מתבצע בזמן O(m) במקרה הגרוע ביותר. כי נצטרך לעבור עליה איבר איבר ולקרוא להורס שהוא תלוי בגודל ה data עליו מצביעה חוליה זו. אמרנו קודם כי היא מצביעה על עץ קורסים ועץ הקורסים מצביע על עץ הרצאות. במקרה הגרוע ביותר כל הרצאה בכל קורס תקבל זמן משלה. לכן גודל הרשימה sorted_list יהיה בגודל m שהוא מספר ההרצאות הכולל במערכת. במקרה זה החולייה תצביע על צומת בודד בעץ הקורסים וצומת זה יצביע גם הוא על צומת בודד בעץ ההרצאות. לכן יש O(1) צמתים לכל איבר ברשימה, לכן נקבל כי סיבוכיות הזמן הכוללת היא O(m).

לאחר מכן, נעדכן את הערך של DS לפי הדרישה לnullptr.

סיבוכיות זמן ומקום:

עמדנו בסיבוכיות זמן של כל הפונקציות כפי שנדרש.

לגבי סיבוכיות מקום: כפי שפורט קודם (כמו בQuit), במבנה הנתונים שבנינו קיימים רק 3 עצים, ננתח את סיבוכיות המקום של כל אחת מהם:

1) עבור העץ courses_tree, כל treeNode בו מכיל Key שהוא מסוג int, וData מטיפוס מערך Class classLecture. נסמן ב k_i את מספר ההרצאות בקורס ה- i. כפי שפורט קודם, המקום הנדרש עבור הקורס הi הוא c*(1+k_i) כאשר c קבוע חיובי. סה"כ יש n treeNodes (n- מספר הקורסים), וכל treeNode דורש c*(1+k_i) מקום), ומתקיים כי $\sum_{i=1}^{n} c_i * (1 + k_i) \leq c * (n + m)$ (כי $\sum_{i=1}^{n} k_i = m$ לפי נתון, וכאשר $c = \max\{c_1, c_2, \dots, c_n\}$), ולכן מתקיים כי סיבוכיות המקום של courses_tree היא O(n+m).
2) רשימה מקושרת sorted_list שהיא במצב הגרוע ביותר בגודל m כאשר m הוא מספר ההרצאות הכולל במערכת((כלומר לכל הרצאה יש זמן צפייה משלה).
3) ראינו לעיל כי עבור אותו מקרה שמתואר ב 2 מתקיים כי גודל עץ העצים שמצביעה עליו הרשימה המקושרת הוא מגודל קבוע כלומר O(1). אבל גם אם הם מגודל גדול יותר, בסה"כ הוא מגודל O(n+m).

ולכן בסה"כ קיבלנו שסיבוכיות המקום היא O(n+m) כפי שנדרש.