מבני נתונים 1
תרגיל רטוב 2-חלק יבש

מגישים:

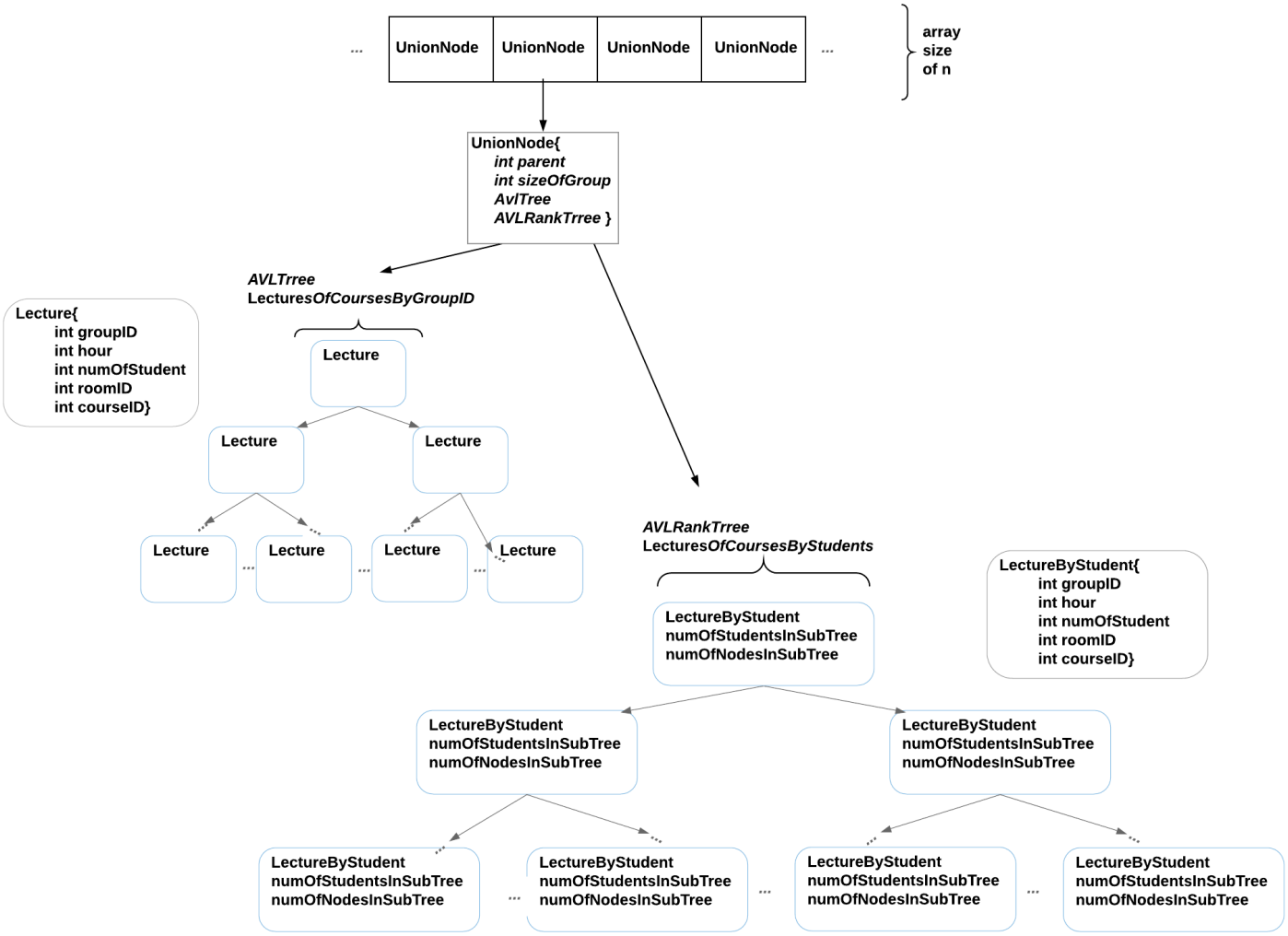
מאי פלסטר
נדב אביוב

מבנה נתונים שלנו כולל:

Courses – מבנה מסוג unionFind עם מספרי הקורסים {1…n} בתור מספרי הקבוצות. את המבנה מימשנו באמצעות מערך בגודל n כאשר כל קורס i מיוצג באינדקס הi-1 . המבנה מומש עם כיווץ מסלולים ואיחוד לפי גודל ולכן פעולות החיפוש והאיחוד ייקחו O(log*n) משוערך כפי שלמדנו בהרצאות. כל קבוצה תכיל, בנוסף לאיבריה את השדות הבאים:

- **LecturesOfCourseByGroups-** עץ AVL שמכיל את כל הרצאותיו של הקורס. העץ ממוין לפי מספר הקבוצה בתור מיון ראשוני. בתור מיון שני לפי השעה שהוא מתקיים (מפתח העץ הוא מבנה מסוג Lecture שהעמסנו על אופרטורי ההשוואה שלו).
- **LecturesOfCourseByStudents** – עץ דרגות שמכיל את כל הרצאותיו של הקורס. (מפתח העץ הוא מבנה מסוג LectureByStudent (שיורש מLecture) שהעמסנו על אופרטורי ההשוואה שלו. העץ ממוין לפי מספר הסטודנטים בכל הרצאה של הקורס בתור מיון ראשון. בתור מיון שני (כלומר אם מתקבל שוויון בין מספר הסטודנטים), ממוין לפי מספר הקבוצה ובתור מיון שלישי לפי השעה שהוא מתקיים. בנוסף כל צומת בעץ מכיל את מספר האיברים בתת העץ שלו (כולל הצומת עצמו) וגם את סכום מספר הסטודנטים תחתיו (כולל מספר הסטודנטים בצומת עצמו). בעת הסרה/הוספה של השדות הללו מתעדכנים.
- מספר ההרצאות של הקורס
- גודל הקבוצה

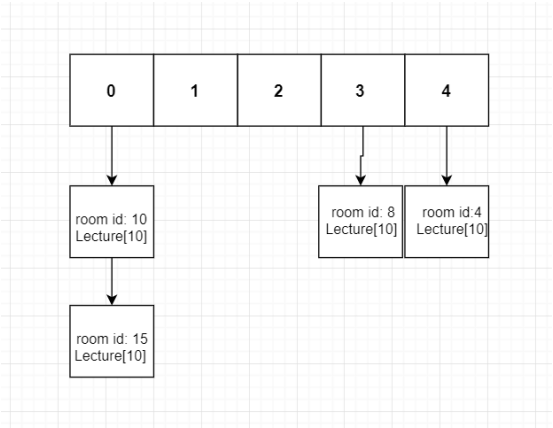
כל השדות ישמרו בתוך struct UnionNode אשר יוכל במערך.



סיבוכיות מקום: במערכת יש n קבוצות (קורסים) בunionFind וסך כל האיברים בכל העצים של הקורסים הוא k (k הוא סך כל הקורסים במערכת). לכן סיבוכיות המקום היא O(n+k).

Rooms- טבלת ערבול דינמית עם פונקציית ערבול רגילה שתכיל את מספרי החדרים בתור המפתחות. פונקציית הערבול שבחרנו היא roomId%size כאשר size הוא גודל הטבלה הנוכחי. הטבלה מומשה באמצעות מערך דינמי של רשימות (למניעת התנגשויות). כל חדר בטבלה מכיל מערך בגודל M של מבנה מסוג Lecture שמסמנות את ההרצאות בכל השעות האפשריות של המערכת.
סיבוכיות מקום : O(r*m)=O(r) כי m קבוע ו-O(m)=1.

לדוגמא:
גודל המערך הוא 5 ומספר האיברים הנוכחי הוא 4. כל איבר נמצא ברשימה המתאימה לפי החישוב roomId%5.
כל איבר ברשימה מכיל (מלבד מספר החדר) גם מערך של הרצאות(מבנה מסוג Lecture) בגודל M(10).



סיבוכיות מקום של כל המבנה: סיבוכיות המקום של כל המבנה היא הסיבוכיות של Courses וRooms (כאשר סיבוכיות המקום של העצים מוכלת בCourses). ולכן : O(r)+O(k+n)= O(n+k+r).

פירוט הפעולות:

1. **void * Init(int n)**

בפעולה זו אנו מאתחלים כל אחד מהשדות של מבנה הנתונים המפורטים למעלה.

- בודקים את תקינות הקלט- האם n קטן או שווה ל0.
- מאתחלים את Rooms –יוצרים טבלת ערבול ריקה בגודל קבוע מראש.
- מאתחלים את Courses - יוצרים מבנה union find עם n איברים. לכל קורס יוצרים 2 עצים ריקים(כמפורט למעלה), מגדירים את גודל הקבוצה כ-1, את מספר ההרצאות כ-0.

סיבוכיות הזמן של בדיקת התקינות ואתחול Rooms היא O(1) . סיבוכיות הזמן של אתחול Courses עם n קורסים היא O(n) מפני שאתחול כל האלמנטים של קורס(כולל בניית העצים הריקים) לוקחים O(1). לכן סך הכל סיבוכיות הזמן של הפעולה היא O(n).

2. **StatusType addRoom(void *DS, int roomID)**

בפעולה זו מוסיפים למערכת חדר ללא הרצאות ולכן יש לעדכן רק את המערך Rooms.

- בודקים את תקינות הקלט – O(1).
- בודקים אם roomId כבר קיים בRooms ומחזירים שגיאה אם הוא כבר קיים – O(1) בממוצע על הקלט לפעולת החיפוש.
- אחרת, מוסיפים את מספר החדר לרשימה המתאימה (כלומר לרשימה שנמצאת במיקום המתאים במערך הדינמי לפי פונקציית הערבול). בעת ההוספה של החדר לטבלת הערבול, נוצר עבורו מערך בגודל M של מבנה מסוג Lecture. מערך זה מסמל את ההרצאות שמתקיימות בחדר בכל שעה במערכת. בשלב הוספת החדר אין עוד הרצאות שמתקיימות בו ולכן כל ההרצאות במערך מאותחלות לערך ברירת מחדל AVAILABLE. מספר השעות קבוע ולכן אתחול המערך לוקח O(1). סך הכל לפעולת ההכנסה לטבלת הערבול O(1) משוערך בממוצע על הקלט.

סך הכל סיבוכיות הזמן של פעולה זה היא O(1) בממוצע על הקלט באופן משוערך.

3. StatusType deleteRoom(void *DS, int roomID)

בפעולה זו אנו רוצים למחוק מהמערכת את החדר בעל מזהה roomId ולכן בדומה לפעולת ההוספה, יש לעדכן רק את המבנה Rooms.

- בודקים את תקינות הקלט – O(1).
- בודקים אם roomsId קיים בטבלת הערבול Rooms ומחזירים שגיאה אם הוא לא קיים- O(1) בממוצע על הקלט לפעולת החיפוש.
- אם הוא קיים, בודקים אם לא קיימות לא הרצאות. לצורך כך, עוברים על כל מערך ההרצאות שלו בלולאה בגודל M, ובודקים עבור כל הרצאה במערך האם היא מכילה את ערך ברירת המחדל AVAILABLE שמסמן כי החדר פנוי בשעה זו. אם מקבלים ערך אחר באחת ההרצאות במערך, אנו יודעים שמתקיימת לפחות הרצאה אחת בחדר זה ולכן לא ניתן למחוק אותו -לכן מחזירים שגיאה. בדיקה זו לוקחת O(1).
- אם לא מתקיימת אף הרצאה בחדר הזה, מוחקים אותו מטבלת הערבול Rooms- פעולת ההסרה לוקחת O(1) משוערך בממוצע על הקלט.

סך הכל סיבוכיות הזמן של פעולה זה היא O(1) בממוצע על הקלט באופן משוערך.

4. StatusType addLecture(void *DS, int courseID, int groupID, int roomID, int hour, int numStudents)

בפעולה זו אנו רוצים להוסיף למערכת הרצאה לפי הפרטים שהתקבלו. לצורך כך יש לעדכן את שני המבנים Rooms וCourses. נבצע זאת לפי סדר הפעולות הבא:

- בודקים את תקינות הקלט – O(1).
- מחפשים בRooms האם החדר בעל המזהה roomID קיים. אם הוא לא קיים, לא ניתן להוסיף את ההרצאה ולכן מחזירים שגיאה. פעולת החיפוש לוקחת O(1) בממוצע על הקלט.
- אם החדר קיים במערכת, נחפש בRooms את ההרצאה שמתקיימת בו בשעה שקיבלנו (כלומר ההרצאה שנמצאת במיקום המתאים במערך ההרצאות של החדר). בודקים האם מתקיימת הרצאה (כלומר אם ההרצאה לא מכילה את הערך הדיפולטי AVAILABLE). אם מקבלים ערך שונה מAVAILABLE , החדר לא פנוי בשעה הנתונה ולכן לא ניתן להוסיף את ההרצאה ולכן נחזיר שגיאה. בחלק זה ביצענו חיפוש נוסף ב Rooms ולכן שוב ביצענו פעולה ב O(1) בממוצע על הקלט.
- מוצאים את הקורס המתאים בCourses. חיפוש זה לוקח O(log*n) משוערך.
- יוצרים אובייקט מסוג Lecture לפי הנתונים שהתקבלו.
- מחפשים בעץ ההרצאות לפי קבוצות של הקורס את ההרצאה. כאמור, העץ ממוין לפי מספרי הקבוצות והשעות של ההרצאות ולכן אם קיימת הרצאה של קבוצה זו בשעה זו נוכל לזהות זאת בבדיקה הזו. נחזיר שגיאה במקרה שנמצאה הרצאה של אותה הקבוצה באותה השעה.
- מוסיפים לשני העצים של הקורס את ההרצאה החדשה. לעץ שממוין לפי הקבוצות, נוסיף את האובייקט מסוג Lecture שיצרנו. לעץ שממוין לפי מספר הסטודנטים, נמיר את האובייקט לאובייקט מסוג LectureBySteudent(שיורש מLecture) ונכניס אותו. הכנסה לכל עץ תיקח O(logK) ולכן שתי פעולות כאלו יקחו עדיין O(logK) ,כאשר K הוא מספר ההרצאות בקורס(גודל העץ).
- מעדכנים את שדה מספר ההרצאות של הקורס בCourses (מגדילים ב 1).

סך הכל סיבוכיות הזמן של פעולה זו היא O(log*n+logK) בממוצע על הקלט משוערך – O(log*n) משוערך לחיפוש בCourses וO(logK) להכנסת איבר לעצים. שאר הפעולות לוקחות O(1) במקרה הגרוע/בממוצע על הקלט ולכן מוכלות בסיבוכיות.

5. StatusType deleteLecture(void *DS, int hour, int roomID)

בפעולה זו אנו רוצים להוריד מהמערכת הרצאה שמתקיימת בחדר והשעה שהתקבלו. לצורך כך יש לעדכן את שני המבנים Rooms וCourses. נבצע זאת לפי סדר הפעולות הבא:

- בודקים את תקינות הקלט- O(1).
- מחפשים בRooms האם החדר בעל המזהה roomID קיים. אם הוא לא קיים, לא ניתן להוריד את ההרצאה ולכן מחזירים שגיאה. פעולת החיפוש לוקחת O(1) בממוצע על הקלט.
- אם החדר קיים במערכת, מחפשים בRooms את ההרצאה שמתקיימת בו בשעה שהתקבלה (כלומר ההרצאה שנמצאת במיקום המתאים במערך ההרצאות של החדר). בודקים האם מתקיימת הרצאה (כלומר אם ההרצאה לא מכילה את הערך הדיפולטי AVAILABLE). אם מקבלים את הערך AVAILABLE , אין הרצאה שמתקיימת בחדר בשעה הנתונה ולכן מחזירים שגיאה. אם מקבלים ערך אחר, נשמור את Lecthere (הרצאה) שקיבלנו במשתנה זמני, ומעדכנים את המיקום המתאים במערך בחזרה לערך AVAILABLE. בחלק זה מבצעים חיפוש נוסף ב Rooms ולכן שוב מבצעים פעולה ב O(1) בממוצע על הקלט.
- אם מוצאים שמתקיימת הרצאה, נחפש אותה (לפי מספר הקורס שמצוי בLecture ששמרנו במשתנה זמני) ב Courses בO(log*n).
- מורידים משני העצים של הקורס את ההרצאה שמצאנו. לעץ שממוין לפי הקבוצות, נוריד את האובייקט מסוג Lecture שמצאנו. לעץ שממוין לפי מספר הסטודנטים, נמיר את האובייקט לאובייקט מסוג LectureBySteudent ונוריד אותו. הסרה מכל עץ תיקח O(logK) ולכן שתי פעולות כאלו יקחו עדיין O(logK) , כאשר K הוא מספר ההרצאות בקורס(גודל העץ).
- מעדכנים את שדה מספר ההרצאות של הקורס בCourses (מקטינים ב 1).

סך הכל סיבוכיות הזמן של פעולה זו היא O(log*n+logK) בממוצע על הקלט משוערך – O(log*n) משוערך לחיפוש בCourses וO(logK) להסרת איבר מהעצים. שאר הפעולות לוקחות O(1) במקרה הגרוע/בממוצע על הקלט ולכן מוכלות בסיבוכיות.

6. StatusType mergeCourses(void *DS, int courseID1, int courseID2)

בפונקציה זו ממזגים שני קורסים בעלי המזהים שהתקבלו. בקורס 1 k1 הרצאות ובקורס 2 k2 הרצאות. מבצעים את הפעולות הבאות:

- בודקים את תקינות הקלט – O(1).
- תחילה בודקים אם ניתן לאחד בין הקורסים:
  - בודקים אם courseID1 וcourseID2 מתייחסים לאותו קורס – לצורך כך מחפשים את שניהם ב Courses ומשווים בין הקבוצות. אם מקבלים שווין, מחזירים שגיאה. חיפוש זה לוקח O(log*n).
  - בודקים אם בשני הקורסים יש הרצאה של קבוצה בעלת אותו מזהה שמתקיימות באותה שעה: מעבירים את איברי שני העצים, LecturesOfCourseByGroups, למערך בסיור inorder. סיור inorder מדפיס את האיברים למערך בצורה ממוינת ולכן ידוע לנו בשלב זה ששני המערכים ממוינים לפי Lecture (לפי מספר הקבוצה בתור מיון ראשוני, ולפי שעה בתור מיון שני). נבדוק אם יש איבר שמופיע פעמיים בשני המערכים: נשווה בין האיבר הראשון של שני המערכים, ונקדם את האינדקס של הקטן מביניהם עד שנגיע לסוף המערכים או לשוויון בין שני איברים. לכל היותר נבצע k1+k2 בדיקות כאלו. גם הדפסת העצים למערכים בסיור inorder לוקחת O(k1+k1) ולכן בדיקה זו לוקחת סך הכל O(k1+k1), כאשר k1 וk2 הם מספר ההרצאות בכל אחד מהקורסים המזוהיים.
- מבצעים איחוד בCoursres בין שני הקורסים:
  - בודקים באיזו קבוצה יש יותר איברים. חיפוש זה לוקח O(log*n). מאחדים את הקבוצה הקטנה לקבוצה הגדולה.
  - מבצעים מיזוג בין העצים של שני הקורסים בין שני עצי הLecturesOfCourseByGroups, ושני עצי LecturesOfCourseByStudent (לפי אלגוריתם מיזוג העצים שלמדנו בתרגול. בשונה מעץ רגיל, לאחר האיחוד מבצעים סיור postorder על עץ הדרגות המאוחד כדי לעדכן את השדות הנוספים שלו) ומכניסים במקום העץ של שורש הקבוצה אליה איחדנו את הקורסים. שורש עץ הדרגות לאחר המיזוג LecturesOfCourseByStudents מכיל מידע על מספר הסטודנטים הכולל בשני הקורסים. פעולת האיחוד והסיור הנוסף לוקחות O(k1+k1).

- מעדכנים את השדות של שורש הקבוצה הגדולה :גודל הקבוצה(גודלה הישן+גודל הקבוצה הקטנה) ומספר ההרצאות סך הכל (המספר הישן+מספר ההרצאות בקבוצה הקטנה). עדכונים אלו לוקחים O(1).

סך הכל סיבוכיות הזמן של פעולה זו היא O(log*n+k1+k1) משוערך – O(log*n) משוערך לחיפוש שני הקורסים בCourses ו(O(k1+k1 לחיפוש בשני העצים הרצאה שחוזרת פעמיים ולאיחוד העצים. שאר הפעולות לוקחות O(1) במקרה הגרוע/משוערך ולכן מוכלות בסיבוכיות.

7. StatusType competition(void *DS, int courseID1, int courseID2, int numGroups, int * winner)

- בודקים את תקינות הקלט- O(1).
- נבדוק אם courseID1 ו courseID2 מתייחסים לאותו מזהה קורס ב**Courses** Union
- נחפש בCourses (UnionFind) את השורשים של courseID1 ו courseID2, בlog*n . במידה והשורשים זהים מחזירים שגיאה, אחרת ממשיכים.
- נחפש בCourses את העץ LecturesOfCourseByStudents שמשויך למזהה של courseID1 , בlog*n ונבצע את הבדיקות הבאות :
  1. נגדיר את המשתנים הבאים :
  v=root(LecturesOfCourseByStudents) , numGroupsLeft =numGroups ,sum=0 .
  2. אם v=null (עץ ריק) נצא ונחזיר את sum אחרת נמשיך.
  3. נבדוק האם **מספר האברים בתת עץ(v)** =<**numGroupsLeft** במידה וכן נחזיר את sum אחרת נמשיך.
  4. כעת נבדוק אם **מספר האיברים בתת עץ(v->right)** =<**numGroupsLeft** במידה וכן נגדיר : v=v->right ונחזור לסעיף 3 ,אחרת נמשיך.
  5. נגדיר : **מספר סטודנטים בצומת(v)**+ **מספר סטודנטים בתת עץ(v->right)** =+sum , **(מספר האיברים בתת עץ(v->right)** + **1)** = **numGroupsLeft-** , v=v->left ונחזור לסעיף 3.
- כך בעצם קיבלנו את סכום numGroup ההרצאות בעלות מספר הסטודנטים הכי גדול בקורס **courseID1**.
  נשמור את התוצאה שקיבלנו בsum1 ונחזור על החישוב עבור העץ LecturesOfCourseByStudents שמשויך למזהה של courseID2, ונשמור את התוצאה שקיבלנו בsum2 .
  בכל חישוב sum בעץ סה"כ ירדנו לכל היותר לעלה פעם אחת ובדרך ביצענו מספר סופי של פעולות, ולכן הסיבוכיות הזמן היא logk לחישוב כל אחד מהסכומים, כאשר k הוא מספר ההרצאות המקסימלי בכל אחד מהקורסים (גודל העץ המקסימלי).

- אחרי שקיבלנו את Sum מכל עץ נחזיר את הקורס ששיך לSum המקסימלי דרך המצביע, במידה והם שווים נחזיר את הקורס בעל הID הגדול יותר.

סה"כ סיבוכיות זמן O (log*n+logk) באופן משוערך – O(log*n) משוערך לחיפוש בCourses, ו(O (logK לחישוב הסכומים. שאר הפעולות לוקחות O(1) ולכן מוכלות בסיבוכיות.

8. StatusType getAverageStudentsInCourse(void *DS, int hour, int roomID, float * average)

- בודקים את תקינות הקלט- O(1).
- בודקים אם roomsId קיים בטבלת הערבול Rooms ומחזירים שגיאה אם הוא לא קיים- O(1) בממוצע על הקלט לפעולת החיפוש.
- אם החדר קיים במערכת, מחפשים בRooms את ההרצאה שמתקיימת בו בשעה שהתקבלה (כלומר ההרצאה שנמצאת במיקום המתאים במערך ההרצאות של החדר). בודקים האם מתקיימת הרצאה (כלומר אם ההרצאה לא מכילה את הערך הדיפולטי AVAILABLE). אם מקבלים את הערך AVAILABLE , אין הרצאה שמתקיימת בחדר בשעה הנתונה ולכן מחזירים שגיאה. בחלק זה מבצעים חיפוש נוסף ב Rooms ולכן שוב מבצעים פעולה ב O(1) בממוצע על הקלט.
- אם מוצאים שמתקיימת הרצאה, נחפש אותה (לפי מספר הקורס בLecture) ב Courses בO(log*n).

- על מנת לחשב את הערך הממוצע של סטודנטים להרצאה, צריך למצוא את מספר הסטודנטים סך הכל בקורס ומספר ההרצאות של הקורס. לוקחים את שדה מספר ההרצאות שאנו מחזיקים בCourses. את מספר הסטודנטים הכולל של הקורס, ניקח משורש העץ LecturesOfCourseByStudents של הקורס (כאמור, כל צומת בעץ מכיל את מספר הסטודנטים בהרצאה עצמה וכל הצמתים שתחתיו ולכן שורש העץ מכיל את כמות הסטודנטיים הכוללת של הקורס). השדות האלו שמורים ומתוחזקים במבנה בכל הפעולות ולכן גישה אליהם תיקח O(1).
- נחשב את הערך ($\frac{\text{סך הסטודטנטים}}{\text{מספר ההרצאות}}$) שמצאנו ונחזיר במצביע average ב$O(1)$.

סך הכל סיבוכיות הזמן של פעולה זו היא $O(\log^* n)$ בממוצע על הקלט משוערך – $O(\log^* n)$ משוערך לחיפוש בCourses. שאר הפעולות לוקחות $O(1)$ במקרה הגרוע/בממוצע על הקלט ולכן מוכלות בסיבוכיות.

9. void Quit(void **DS)

בפעולה זו שחררנו את כל הזיכרון שהקצינו בשימוש במערכת. בפעולה זו אנו מוחקים את המבנה this ולכן קוראים להורס של המבנים Rooms וCourses. נראה כי היא עומדת בדרישת סיבוכיות הזמן(כאשר n הוא מספר הקורסים,k מספר הקורסים, ו r מספר החדרים):

- מחיקת Rooms: Rooms הוא מערך דינמי של רשימות. סך כל האיברים בכל הרשימות שווה למספר החדרים r במערכת. ב במחיקת המבנה נשחרר את כל איברי המערך (וכל איברי הרשימות בתוכם). כפי שפירטנו קודם, כל איבר כזה כולל את מספר החדר ומערך בגודל M של הרצאות. כל איבר כזה הוא בגודל קבוע לכן סיבוכיות הזמן של מחיקת כל איבר היא $O(1)$. לכן פעולה זו תיקח בסך הכל $O(r)$.
- מחיקת Courses : Courses הוא מבנה מסוג UnionFind שמומש על ידי מערך בגודל n. כל איבר במערך מכיל 2 עצים (ועוד מספר משתנים מסוג int). סך כל האיברים של כל העצים במערך הוא 2k (כל הקורסים במערכת שמפוזרים בין כל קורסי המערכת). לכן סך הכל יש למחוק 2n עצים שמספר הצמתים **הכולל** בהם הוא 2k ולכן סיבוכיות הזמן של המחיקה היא $O(2n+2k)=O(n+k)$.

לכן בסך הכל הסיבוכיות של פעולה זו היא $O(r+k+r)$.