<u>תיאור מבנה הנתונים:</u>

- למבנה שלנו נקרא $UGschedule$.
- כל ההרצאות מתקיימות באותו יום בשבוע, וקיימות $m = 10$ שעות שבהן יכולות להתקיים הרצאות.
- מבנה הנתונים מנהל nקורסים, כאשר הפרמטר n ניתן בזמן האתחול של מבנה הנתונים. לכל אורך ריצת התוכנית, הID- של הקורסים יהיה מספר בתחום$1,2,\ldots n$.
- כל שיעור המיוצג במערכת מוגדר על ידי קבוצה, שעה ומספר הסטודנטים בו.

מבנה הנתונים שלנו $UGschedule$ בנוי באופן הבא:

- n - <u>משתנה עבור מספר הקורסים</u> במערכת לתמיכה בכלל הפעולות של המבנה.
- $Rooms$ - <u>טבלת ערבול עבור החדרים</u> הקיימים בטאוב, בתחילה אין חדרים ובהמשך יתווספו. נשתמש במערך דינמי (נממש כמו שנלמד בקורס) כדי לממש את הטבלה, פונקציית הערבול תהיה מודלו של גודל המערך ברגע נתון. זוהי טבלת ערבול מסוג chain hashing, כל תא במערך מחזיק מצביע לרשימה מקושרת של חדרים.
- $Courses$: משתנה מסוג <u>UnionFind(כפי שנלמד בקורס) אשר מחזיק את כל הקורסים</u> שבמערכת. בעזרת courses נוכל לנהל את כל הקורסים שבמערכת בסיבכויות נדרשת $O(log * n)$. זהו <u>UnionFind</u> שמנהל אוביקטים של מחלקת Course המתוארת באופן הבא :
- -<u>לכל קורס יהיו שני עצים</u> עבור ההרצאות שלו :
  - lectures <u>: עץ AVL רגיל</u> (אותו מימשנו בתרגיל הרטוב הקודם) בו נשמור את כל ההרצאות של הקורס, ממויינות לפי שני מפתחות : קבוצה ושעה.
- lectures_ranking <u>: עץ דרגות הפוך(מהגדול לקטן) שבו ההרצאות ימויינו לפי מספר הסטודנטים</u> בהן, עבור הפונקצייה competition. את העץ נממש פשוט על ידי הוספת שני שדות חדשים לכל צומת : מספר הצמתים (מספר ההרצאות) וסכום הסטודנטים בתת העץ שהצומת הוא שורשו. שדות אלו יסייעו לנו לשמור מידע נוסף על תת-העץ של כל צומת ולקיים את ההגדרה : עץ דרגות הוא עץ בו בכל צומת v נשמר מספר הצמתים בתת-העץ ששורשו v.

<u>בתוך עץ -</u>lectures_ranking <u>:</u>

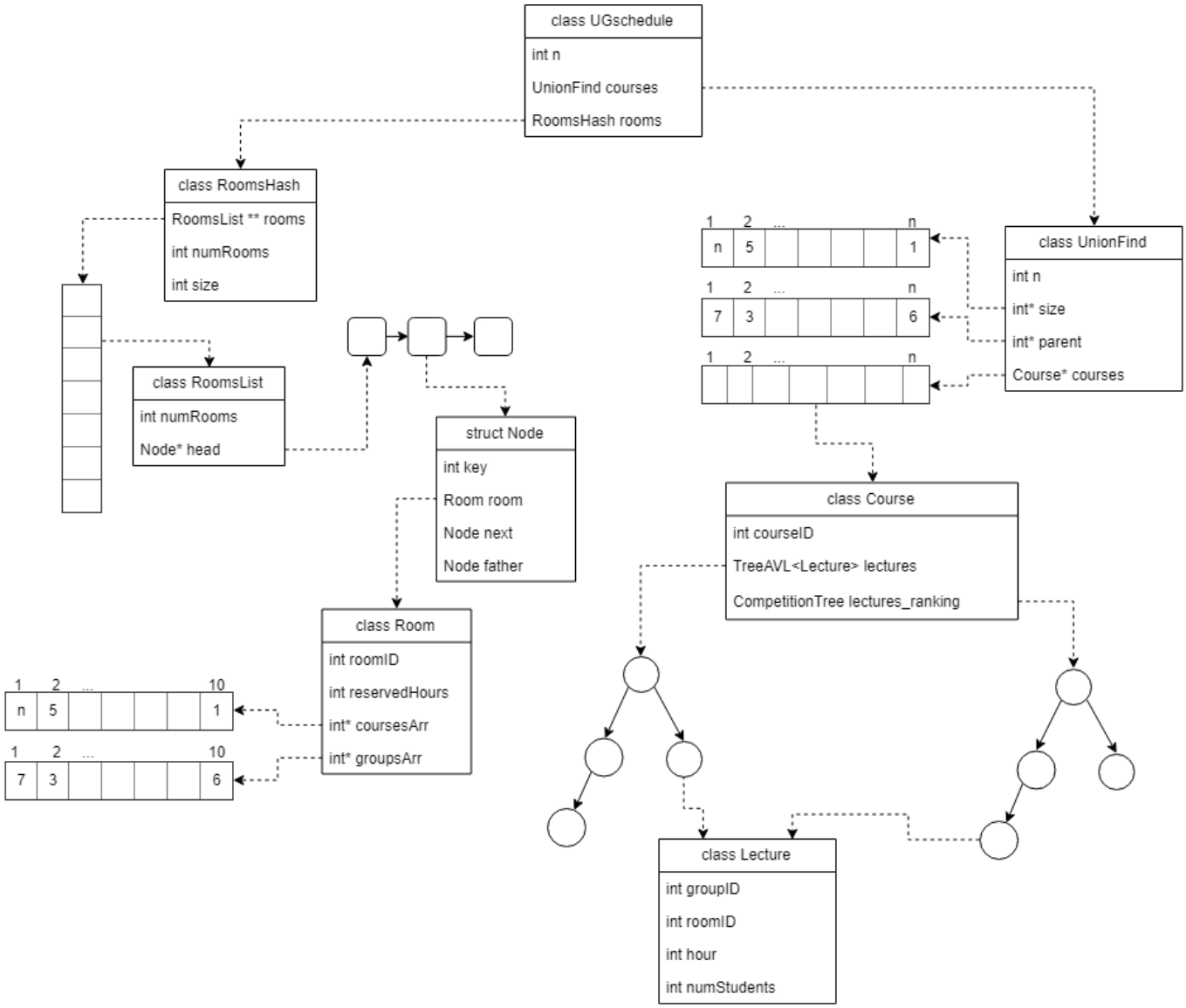
- נתחזק מצביע $Root$ לשורש של <u>עץ הקורסים</u>.
- נתחזק צמתים $CompetitionNode$ <u>עבור</u> ההרצאות. כל צומת מכיל את השדות הבאים :
  - מצביע ל $lecture$ - מצביע לאובייקט מחלקה של הרצאה.
  - גובה העץ $height$ - כלומר גובה תת-העץ שהצומת הוא השורש שלו.
  - מצביע לבן ימני $right\_son$.
  - מצביע לבן שמאלי $left\_son$.
  - מספר הצמתים בתת העץ של הצומת $nodes\_in\_subtree$.
  - סה"כ הסטודנטים בהרצאות בתת העץ השל הצומת $students\_sum\_in\_subtree$.

- המיון בעץ הדרגות יתבצע לפי מספר הסטודנטים בהרצאה, ובמקרה שיש שתי הרצאות עם אותו מספר סטודנטים הוא יתבצע באותו אופן כמו בעץ AVL הרגיל (לפי מספר קבוצה ולאחר מכן שעה).

שני עצי ההרצאות מחזיקים מצביעים לאוביקטים מסוג מחלקת Lecture המתוארת באופן הבא :

- groupID : משתנה עבור מספר קבוצת התרגול.
- roomID : מספר החדר בו מתקיימת ההרצאה.
- hour : השעה בא מתקיימת ההרצאה.
- numStudents : מספר הסטודנטים בהרצאה .

שרטוט של מבנה הנתונים:



דגשים להמשך:

- בכל הפונקציות תחילה נבדוק את תקינות הקלט, בסיבוכיות זמן $O(1)$.
- בכל הפונקציות נבצע בדיקת תקינות לאחר הקצאת זיכרון, בדיקה מתבצעת בסיבוכיות זמן $O(1)$.

סיבוכיות המקום של מבנה הנתונים

- באתחול של מבנה הנתונים סיבוכיות המקום היא $O(n)$ מאחר ועוד אין הרצאות וחדרים במערכת, כך שn מייצג את מספר הקורסים שבמערכת
- בהמשך יתווספו חדרים למערכת, מספר החדרים מסומן ב-r, ולכן סיבוכיות המקום היא $O(r+n)$.
- בהמשך יתווספו הרצאות למערכת, מספר ההרצאות הכולל מסומן ב-k, ולכן סיבוכיות המקום היא $O(n+k+r)$.
- סה"כ סיבכויות המקום של מבנה הנתונים היא $O(n+k+r)$.

תיאור הפעולות:

- **void * Init(int n):**

הפעולה מקצה מבנה נתונים ריק מסוג UGschedule עם n קורסים שה-ID שלהם רץ מ1 עד n באופן הבא:

הפעולה שומרת את מספר הקורסים, מאתחלת טבלת ערבול ריקה rooms של חדרים ומאתחלת UnionFind בשם courses של n קורסים, כך שלכל קורס אין הרצאות.

סיבוכיות זמן $O(n)$ וסיבוכיות מקום $O(n)$.

בהתחלה עוד לא נוספו חדרים ולכן סיבוכיות המקום היא $O(1)$, בהמשך יתווספו חדרים והסיבוכית מקום תגדל ל- $O(r)$.

<u>סיבוכיות זמן</u>: אתחול טבלת הערבול היא ב-$O(1)$, מאחר ואנחנו מאתחלים מערך עם 3 תאים. כאשר אנחנו מאתחלים את courses אנחנו יוצרים 3 מערכים באורך n ולכן הסיבוכיות היא $O(n)$. לכן, אתחול המערכת מתבצע ב-$O(n)$ כנדרש.

נגדיר עץ הרצאות $CompetitionTree$ עבור ההרצאות של כל קורס:

- נגדיר מצביע $Root$ לשורש של העץ מאותחל ל-$NULL$, סיבוכיות זמן $O(1)$ וסיבוכיות מקום $O(1)$.
- בהתחלה עוד אין הרצאות במערכת ולכן אין צמתים. בהמשך יהיה מספר דינמי של צמתים לפי מספר ההרצאות הכולל במערכת k, וסיבוכיות המקום תהיה $O(k)$.

לכן, בסה"כ נקבל שסיבוכיות הזמן של הפעולה היא $o(n)$ במקרה הגרוע, כנדרש.

- **StatusType addRoom(void *DS, int roomID):**

הפעולה מוסיפה כיתה חדשה עם מזהה *roomID*, כך שלכיתה שהתווספה אין עדיין הרצאות:

1. נבדוק את תקינות הקלטים. במקרה ואחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.
2. נבדוק אם לא קיים כבר במערכת חדר עם המזהה *roomID*. אם כן, תיזרק הודעת שגיאה *FAILURE*.
3. נוסיף לטבלת הערבול *rooms* כיתה חדשה עם המזהה *roomID*. במקרה של בעיה בהקצאת הזיכרון תיזרק שגיאה ALLOCATION_ERROR.

<u>סיבוכיות זמן</u>: כפי שלמדנו בתרגול, הוספת איבר לטבלת ערבול מתבצע בסיבוכיות זמן $O(1)$ בממוצע על הקלט באופן משוערך, כנדרש.

- **StatusType deleteRoom(void *DS, int roomID):**

הפעולה מוחקת כיתה עם מזהה *roomID* בתנאי שבכיתה אין הרצאות:

1. נבדוק את תקינות הקלטים. במקרה ואחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.
2. נבדוק אם קיימת כיתה עם מזהה *roomID* ונבדוק שאין בה הרצאות. אם לא קיימת כיתה עם מזהה *roomID* או שבכיתה יש הרצאות תיזרק הודעת שגיאה *FAILURE*.
3. נמחוק את החדר מטבלת הערבול *rooms*.

סיבוכיות זמן: כפי שלמדנו בתרגול, הסרת איבר לטבלת ערבול מתבצעת בסיבוכיות זמן $O(1)$ בממוצע על הקלט באופן משוערך, כנדרש.

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

הפעולה מוסיפה הרצאה חדשה:

1. נבדוק את תקינות הקלטים. במקרה ואחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.
2. נבדוק אם החדר עם המזהה *roomID* קיים ע"י חיפוש בטבלת הערבול *rooms*. אם הוא לא קיים, תיזרק הודעת שגיאה *FAILURE*. אם הוא קיים, נבדוק אם החדר פנוי בשעה *hour*, אם לא, תיזרק הודעת שגיאה *FAILURE*.
3. נחפש את הקורס עם המזהה **courseID** ב- *courses* (*UnionFind* של הקורסים).
4. נבדוק שלא קיימת לקורס הרצאה של קבוצה עם מזהה **groupID** בשעה הנ"ל. אם כן, תיזרק הודעת שגיאה *FAILURE*.
5. נוסיף את ההרצאה לעץ ההרצאות של הקורס. במקרה של בעיה בהקצאת הזיכרון תיזרק שגיאה ALLOCATION_ERROR.
6. נעדכן שהחדר עם המזהה *roomID* תפוס ע"י קבוצת התרגול מספר **groupID** של קורס עם מזהה **courseID**.

<u>סיבוכיות זמן</u>: חיפוש החדר עם מזהה *roomID* בטבלת הערבול מתבצע בסיבוכיות זמן $O(1)$ בממוצע על הקלט משוערך. חיפוש הקורס עם מזהה *courseID* מתבצע בסיבוכיות זמן $O(\log^* n)$ משוערך כאשר n מייצג את מספר הקורסים במערכת, כפי שלמדנו בהרצאה על חיפוש במבנה *Union Find*. נבחין כי עצי ההרצאות הם עצי AVL לכן, הוספת הרצאה לקורס עם המזהה **courseID** מתבצעת בסיבוכיות $O(k)$, כאשר k מייצג את מספר הרצאות שיש לקורס. נקבל שבסה"כ סיבוכיות הזמן של הפעולה היא $O(\log^* n + k)$ בממוצע על הקלט משוערך, כנדרש.

- **StatusType deleteLecture(void *DS, int hour, int roomID)**

הפעולה מוחקת את ההראצה שנמצאת בשעה *hour* בחדר *roomID*:

1. נבדוק את תקינות הקלטים. במקרה ואחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.
2. נבדוק אם החדר עם המזהה *roomID* קיים ע"י חיפוש בטבלת הערבול *rooms*. אם הוא לא קיים, תיזרק הודעת שגיאה *FAILURE*. אם הוא קיים, נבדוק אם החדר פנוי בשעה *hour*. אם כן, תיזרק הודעת שגיאה *FAILURE*. אחרת, נשמור את מס' הקורס ומספר קבוצת התרגול של ההרצאה שמתקיימת בחדר בשעה *hour*.
3. נעדכן את החדר להיות פנוי בשעה *hour*.
4. נחפש את הקורס שיש לו הרצאה בחדר *roomID* בשעה *hour* ונמחק את ההרצאה מהקורס.

<u>סיבוכיות זמן</u>: חיפוש החדר עם מזהה *roomID* בטבלת הערבול מתבצע בסיבוכיות זמן $O(1)$ בממוצע על הקלט משוערך ועדכונו להיות פנוי מתבצע בסיבוכיות זמן $O(1)$. חיפוש הקורס עם מזהה *courseID* מתבצע בסיבוכיות זמן $O(\log^* n)$ משוערך כאשר n מייצג את מספר הקורסים במערכת, כפי שלמדנו בהרצאה על חיפוש במבנה *Union Find*. נבחין כי עצי ההרצאות הם עצי AVL לכן, מחיקת הרצאה של קורס עם המזהה **courseID** מתבצעת בסיבוכיות $O(k)$, כאשר k מייצג את מספר הרצאות שיש לקורס. בסה"כ נקבל שסיבוכיות הזמן של הפעולה היא $O(\log^* n + k)$ בממוצע על הקלט משוערך, כנדרש.

- **StatusType mergeCourses(void *DS, int courseID1, int courseID2)**

הפעולה מאחדת בין שני קורסים, כך שהמזהים courseID1, courseID2 יתייחסו לאותו הקורס וקבוצות ההרצאות של שני הקורסים יהיו שייכות לאותו הקורס:

1. נבדוק את תקינות הקלטים. אם אחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.
2. נבדוק אם שני המזהים מתייחסים לאותו הקורס. אם כן, תיזרק הודעת שגיאה *FAILURE*.
3. נבדוק אם בשני הקורסים courseID1, courseID2 יש הרצאות של קבוצות עם אותו מזהה המתקיימות באותה השעה. אם כן, תיזרק הודעת שגיאה *FAILURE*.
4. נאחד את עצי ההרצאות של שני הקורסים עם המזהים courseID1, courseID2.

במקרה של בעיה בהקצאת הזיכרון בעת איחוד עצי ההרצאות, תיזרק שגיאה ALLOCATION_ERROR.

<u>סיבוכיות זמן:</u> חיפוש הקורסים עם המזהים courseID1, courseID2 מתבצע בסיבוכיות זמן $O(\log^* n)$ משוערך כאשר n מייצג את מספר הקורסים במערכת. בנוסף, איחוד עצי ההרצאות של שני הקורסים מתבצע בסיבוכיות זמן $O(k_1 + k_2)$ כך ש-k_1, k_2 הם מספר ההרצאות בכל אחד מהקורסים המזוהים, כפי שלמדנו בתרגול. קיבלנו שבסה"כ סיבוכיות הזמן של הפעולה היא $O(\log^* n + k_1 + k_2)$ משוערך.

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

הפעולה מבצעת תחרות בין שני קורסים. כל קורס בוחר את numGroups ההרצאות שרשומים אליהם הכי הרבה סטודנטים. הקורס המנצח הוא הקורס שיש לו יותר סטודנטים בכל numGroups ההרצאות הללו. במקרה בו לאחד הקורסים יש פחות מ-numGroups הרצאות, אז כל ההרצאות שלו ישתתפו בתחרות.

1. נבדוק את תקינות הקלטים. אם אחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.
2. נחפש את שני הקורסים עם המזהים *courseID1, courseID2*. לאחר מכן נבדוק שהם לא מתייחסים לאותו הקורס. אם כן, תיזרק הודעת שגיאה *FAILURE*.
3. נחשב עבור כל אחד מהקורסים, נחשב את numGroups ההרצאות שרשומים אליהן הכי הרבה סטודנטים ונשווה בניהם. אם יש תיקו, המנצח הוא הקורס שה-*ID* שלו הוא הגדול מבין השניים. נחזיר את מספר הקורס המנצח במשתנה *winner*.

<u>סיבוכיות זמן:</u> חיפוש שני הקורסים מתבצע ב*courses* שהוא מסוג *UnionFind*. לכן, כפי שלמדנו בהרצאה סיבוכיות הזמן של החיפוש מתבצעת בסיבוכיות זמן $O(\log^* n)$ משוערך, כך ש-n מייצג את מספר הקורסים במערכת. החישוב של מספר הסטודנטים בnumGroups ההרצאות שרשומים אליהם הכי הרבה סטודנטים, מתבצע בעזרת עץ דרגות הפוך מסוג AVL. עבור כל אחד מהקורסים, נחפש את הצומת שדרגתו שווה ל-*numGroups*. בהרצאה למדנו שחיפוש צומת בעץ AVL מתבצע בסיבוכיות זמן $O(logm)$ כך ש-m מייצג את מספר ההרצאות שיש לקורס. לכן נקבל שפעולת החיפוש עבור שני הקורסים מתבצעת בסיבוכיות זמן $O(logk)$ כך ש-k מייצג את מספר ההרצאות המקסימלי בכל אחד מהקורסים. לסיכום, קיבלנו שבסך הכל סיבוכיות הזמן של הפעולה היא $O(\log^* n + logk)$ משוערך, כנדרש.

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

הפונקציה תחזיר את ממוצע מספר הסטודנטים להרצאה בקורס שיש בו הרצאה בחדר roomID בשעה hour:

1. נבדוק את תקינות הקלטים. אם אחד הקלטים לא תקין תיזרק הודעת שגיאה INVALID_INPUT.

2. נבדוק אם החדר עם המזהה *roomID* קיים ע"י חיפוש בטבלת הערבול *rooms*. אם הוא לא קיים, תיזרק הודעת שגיאה *FAILURE*. אם הוא קיים, נבדוק לאיזה קורס מתקיימת הרצאה בשעה *hour*. אם החדר פנוי בשעה הנ"ל אז תיזרק הודעת שגיאה *FAILURE*.
3. נחפש את הקורס ב-Union Find של הקורסים courses.
4. נחשב את ממוצע מס' הסטודנטים להרצאה בקורס.

<u>סיבוכיות זמן:</u> חיפוש החדר עם מזהה *roomID* בטבלת הערבול מתבצע בסיבוכיות זמן $O(1)$ בממוצע על הקלט משוערך. חיפוש הקורס עם מזהה *courseID* מתבצע בסיבוכיות זמן $O(\log^* n)$ משוערך כאשר n מייצג את מספר הקורסים במערכת, כפי שלמדנו בהרצאה על חיפוש במבנה *Union Find*. חישוב הממוצע מתבצע בסיבוכיות זמן $O(1)$, מאחר ואנחנו סוכמים את מס' ההרצאות של הקורס, ואת הסכום הכולל של הסטודנטים בהרצאות בקורס בזמן הפעולות של הוספת ומחיקת הרצאה לקורס ומיזוג קורסים. נקבל שבסך הכל סיבוכיות הזמן של הפעולה היא $O(\log^* n)$ משוערך בממוצע על הקלט, כנדרש.

- **void Quit(void **DS)**

הפעולה משחררת את המבנה.
נשחרר את הפוינטר DS ולאחר מכן נציב בו ערך NULL.

<u>סיבכוכיות זמן:</u> שחרור טבלת הערבול rooms מתבצע בסיבוכיות זמן $O(r)$ במקרה הגרוע כך ש-r הוא מספר החדרים במערכת. זאת מכיוון שהגודל הכולל של רשימות שיש בטבלת הערבול הוא כמספר החדרים במערכת ולכן מחיקת כל הרשימות יתבצע בסיבוכיות זמן $O(r)$. בנוסף, גודל המערך של הרשימות חסום ע"י מספר החדרים, כך שבמקרה הגרוע גודלו הוא פי 2 ממספר החדרים במערכת. לכל קורס יש 2 עצי הרצאות, לכן כפי שלמדנו בתרגול, מחיקת כל עצי ההרצאות מתבצעת בסיבוכיות זמן $O(k)$, כאשר k מייצג את מספר ההרצאות בכל המערכת. מחיקת הקורסים מהמערכת מתבצע בסיבוכיות זמן $O(n)$ כאשר n הוא מספר הקורסים במערכת, מאחר והקורסים מיוצגים ב*Union Find*, שמחזיק מצביעים למחלקת *Course*. לכן, עלינו לשחרר n קורסים.
בסך הכל קיבלנו שסיבוכיות הזמן הכללית של הפעולה היא $O(n+k+r)$ במקרה הגרוע, כנדרש.