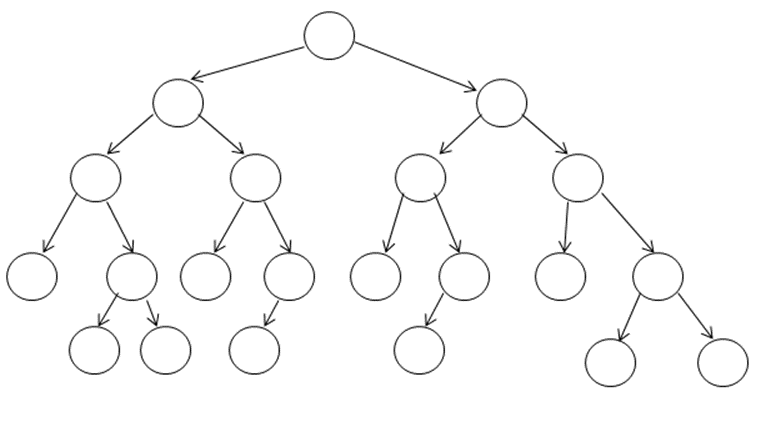
חלק יבש:

תחילה נציג סכמה כללית של התרגיל, כך שיהיה קל להבין את הסיבות למימוש דברים מסוימים בדרך שנעשו. בתרגיל זה הוגדרו כמה מחלקות עיקריות:

**AVLTree**:

מבנה הנתונים שעליו מושטת התרגיל הוא עץ AVL. מבנה נתונים זה עוזר לנו לעמוד בדרישות סיבוכיות זמן הדרושות בתרגיל. בעץ AVL ניתן למצוא איבר, להכניס איבר ולהוציא אותו ב O(logn). כמו שנלמד בכיתה נצטרך להתמודד עם הפרות גורמי האיזון בעץ לאחר הוצאה והכנסה. במימוש המחלקה של העץ ישנם מתודות אשר תפקידם הוא לתפעל את העץ ולבצע עליו פעולות בצורה נוחה. כמו כן מימשנו את העץ בצורה גנרית כך שניתן להשתמש בעץ לכל טיפוס/מחלקה ובכל סוג מפתח לא רק מפתח מטיפוס ערך שלם למשל. בנוסף מימשנו מחלקה אשר תפקידה לאחד בין עצי AVL. ברגע שהוגדר הטיפוס עץ זה יהיו מאותו טיפוס למהלך הריצה. לפניכם דוגמא של תרשים .AVL

**מתודות נבחרות:**

החזרת גובה תת העץ:

**int** getHeight(AVLNode<T, K> \*node) **const**;

החזרת BF(v) (פירושו נלמד בכיתה)

**int** balanceFactor(AVLNode<T, K> \*root) **const**;

הכנסת צומת חדשה לעץ עם מפתח ומידע:

**bool** insert(**const** T &data, **const** K &key);

מחיקת הצומת בעלת המפתח key:

**Void** deleteBYKey(**const** K &key);

ביצוע גלגול ימינה לשם איזון העץ:

**void** rotateRight(AVLNode<T, K> \*root);

ביצוע גלגול שמאלה לשם איזון העץ:

**void** rotateLeft(AVLNode<T, K> \*root);

חיפוש והחזרת הצומת בעלת המפתח key:

AVLNode<T, K> \*findBYKey(**const** K &key);

קבלת הצומת הבאה בסדר inorder של המפתח key:

AVLNode<T, K> \*nextINOrder(AVLNode<T, K> \*root,**const** K &key);

הדפסת מפתחות העץ בסדר inorder:

**void** printInOrder() **const**;

הכנסת עץ למערך בסדר inorder:

**void** inOrderToArrayKeys(K \*\* arr);

**Lecture:**

המחלקה שיעור מוגדרת לשם שמירת השיעורים בכיתה. שיעור מוגדר על פי הכיתה שבו הוא נערך והשעה שבו הוא מתקיים. כמו כן כל שיעור שקיים במערכת שייך לקורס כלשהו יחיד ולכן גם נוכל לגשת לקורס דרך השיעור שלו.

**מימוש השוואת מפתחות של שיעור:**

**class** compareLectures{  
**public**:  
 compareLectures(){};  
  
 **bool operator**()(**const** Lecture\_Key &k1,**const** Lecture\_Key &k2) { // p1 is first  
 **if** (k1.getHour() > k2.getHour())  
 **return false**;  
 **else if** (k2.getHour() == k1.getHour())  
 **return** k1.getRoom() > k2.getRoom();  
 **else  
 return true**;  
 }  
  
 ~compareLectures(){};  
};

**מתודות נבחרות:**

החזרת הקורס אליו שייך השיעור:

Course \*getCourse();

שינוי הקרוס אליו שייך השיעור:

**void** setCourse(Course \*course);

החזרת החדר בו נמצא השיעור:

**int** getRoom() **const**;

**Course:**

המחלקה קורס מוגדרת לשם שמירת הקורסים במערכת. לכל קורס במערכת יש מספר ייחודי המזהה אותו. כמו כן בכל קורס נחזיק את כל השיעורים של אותו הקורס בעץ AVL. נשים לב כי הכנסה לעץ נעשית על ידי השוואת המפתחות של המחלקה שיעור שיפורט בהמשך.

**מתודות נבחרות:**

הוספת שיעור לקורס:

**void** addLectureToCourse(Lecture \*lec);

מחיקת שיעור מקורס:

**void** removeLectureFromCourse(**const** Lecture\_Key &k1);

שינוי מספר מזהה של קורס:

**void** setId(**int** id\_)

**System**

המחלקה מערכת מוגדרת לשם הגדרת המערכת. כלומר, מחלקה זו תהיה אחראית למימוש הפעולות הדרושות בתרגיל, במחלקה זאת נחזיק עץ של קורסים, מטריצה של שיעורים, מערך של השעות והמונים ומערך של השעות ועצי החדרים.

**הסבר ופירוט המתודות:**

StatusType addLecture(**int** hour,**int** room,**int** courseId);

הוספת שיעור חדש למערכת בכיתה והשעה הנתונים:

סיבוכיות מבוקשת: O(log n+log m)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* אם השיעור נמצא כבר(חיפוש ב-O(log n) כפי שנלמד בעץ הקורסים ואז O(log m) חיפוש של השיעור) נחזיר שגיאה = O(log n+log m)
* אחרת השיעור תקין להכנסה , אם הקורס לא קיים נוסיף אותו לעץ הקורסים O(log n) ואז נוסיף את השיעור לעץ השיעורים לפי המפתח המשולב של שיעור. מיון ראשוני לפי שעה ומיון שניוני לפי חדר. בנוסף נציב במקום המתאים במטריצה את השיעור החדש ונעדכן את מערך העצים כך שבשעה הנתונה החדר כבר לא פנוי. = O(log n+log m)
* StatusType getCourseId(**int** hour,**int** room,**int** \*courseId);

החזרת מספר הקורס של השיעור שמתקיים בחדר ובשעה התנתונים:

סיבוכיות מבוקשת: O(1)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* נבדוק במטריצה, אם לא קיים שיעור בשעה והחדר הנתונים נחזיר שגיאה = O(1)
* אחרת השיעור קיים, ניגש לתא במטריצה ונחזיר את הקורס של השיעור = O(log n)
* StatusType deleteLecture(**int** hour,**int** room);

מחיקת שיעור מהמערכת:

סיבוכיות מבוקשת: O(log n+log m)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* נבדוק במטריצה, אם אין שיעור כזה במערכת נחזיר שגיאה = O(1)
* אחרת, נמחק את השיעור מעץ השיעורים במחלקה קורס שלה השיעור שייך. נמחק את התא הרלוונטי במטריצה. נוסיף במערך השעות בשעה הנתונה את החדר שהתפנה לעץ החדרים. נעדכן את מערך השיעורים במערכת = O(log n + log m)
* StatusType changeCourseId(**int** old, **int** new\_);

שינוי מספר הקורס של קורס במערכת:

סיבוכיות מבוקשת: O(log n+m)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* אם מספר הקורס החדש אינו קיים, נעדכן את המספר הישן בחדש = O(log n)
* אם מספר הקורס החדש קיים, ניצור מחלקה חדשה עם מספר הקורס החדש שעץ השיעורים שלה יהיה האיחוד של שני עצי השיעורים של המחלקות הישנות. את איחוד העצים נבצע בעזרת מערכי עזר וסיור inorder על העץ = O(log n+m)
* אם מספר הקורס הישן לא קיים נחזיר שגיאה = O(log n)

StatusType CalculateScheduleEfficiency(**float** \*efficiency);

חישוב יעילות המערכת: קיים מבנה במערכת בשם HourAndCounters אשר מחזיק מערך ומספר מונים אשר בעת הכנסה והוצאה של הרצאות מהמערכת מעדכן את המונה של כל שעה, וקיים מונה אשר מתפקד כסכימה של המונים במערך, בעת הפעלת הפונקציה המבנה מבצע פעולה אחת ולכן עומד בסיבוכיות הנדרשת.

סיבוכיות מבוקשת: O(1)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* אם לא קיימים שיעורים במערכת נחזיר שגיאה = O(1)
* אחרת נעזרת במחלקה hourandcounter מספר השעות במערכת בהם מתקיים לפחות שיעור אחד ומספר השיעורים הכולל במערכת. O(1)

StatusType getAllRoomsByHour(**int** hour,**int** \*\*rooms, **int** \*numOfRooms);

קבלת על החדרים הפנויים לפי שעה:בעזרת המבני הנתונים HoursAndTrees נחזיק מערך של עצים שלכל שעה אפשרית נחזיק את מספר הכיתות הפנויות בעץ של השעה המתאימה

בעת הוספת הרצאה בשעה כלשהי נסיר מהעץ את הכיתה הרלוונטית וכאשר נבטל הרצאה נוסיף בחזרה את הכיתה לעץ בשעה הרלוונטית

כאשר נקרא לפונקציה getAllRoomsByHour ניגש בO(1) לתא המתאים לשעה

ונעבור על העץ בIndOrder כפי שנלמד בכיתה בO(r) ונטען את הכיתות הפנויות למערך עזר אותו נקבל מהמערכת לטובת הפלט לכן סהכ סיבוכיות 1+2r לכן O(r) כפי שהתבקש.

סיבוכיות מבוקשת: O(r)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* אם לא קיימים חדרים פנויים נחזיר שגיאה = O(1)
* אחרת, ניגש למערך השעות והעצים, נעבור על עץ החדרים הפנויים סיור inorder, בכל צומת בעץ, נעתיק את השעה והחדר למערך = O(r)

StatusType GetAllLecturesByCourse(**int** courseId,**int** \*\*hours,**int** \*\*rooms,**int** \*numOfLectures);

החזרת כל השיעורים של קורס:

סיבוכיות מבוקשת: O(m+log n)

הסבר:

* אם הפרמטרים אינם עומדים בדרישות התרגיל נחזיר שגיאה = O(1)
* אם הקרוס לא קיים נחזיר שגיאה = O(log n)
* אחרת נמצא את הקורס בעץ הקורסים, נבצע סיור בעץ השיעורים של הקורס סיור inorder ונכניס את השעה והחדר אשר נמצאים בכל צומת לשני מערכים שונים בהתאמה.

**void** Quit(**void** \*\*DS);

מחיקת המערכת ושחרור הזכרון

סיבוכיות מבוקשת: O(n+kr)

הסבר:

שחרור המערכת מתבצע לפי השלבים הבאים:

* מעבר על כל עץ הקורסים והכנסת כל מצביע לCourse מכל צומת בעץ לתוך מערך של מצביעים לCourse – O(n), מעבר על המערך ושחרור כל מצביע של Course – O(n).
  + בעת קריאה להורס של כל Course מתבצעת קריאה להורס של עץ השיעורים ששייך לאותו קורס אשר עובר על העץ בPostOrder ומשחרר כל צומת בעץ – O(kr) (גודל העץ הוא לכל היותר kr ).

סה"כ סיבוכיות עבור שלב זה : O(kr+ n)

* שחרור מטריצת השיעורים על ידי מעבר על כל תא בה – O(kr), שחרור מערך השעות ועצי השיעורים – O(kr), שחרור מערך השעות והמונים – O(k)

סה"כ סיבוכיות עבור שלב זה: O(kr)

* קריאה להורס העצים של המערכת אשר עובר על כל עץ ב-PostOrder ומשחרר כל צומת בעץ –O(n) עבור עץ השבטים.

קיבלנו סה"כ סיבוכיות של O(n + kr) עבור הפעולה הזאת, כנדרש.

סיבוכיות המקום של המערכת:

* אנו מקצים מספר קבוע של מערכים שגודלם הוא לכל היותר n או kr.
* אנו מקצים עץ קורסים שגודלו הוא n ועץ שיעורים שגודלו הוא kr.
* אנו מקצים מטריצה שגודלה הוא kr.

סה"כ סיבוכיות מקום: O(n+kr).