חלק יבש

קודם נציג את המחלקות הנכללות במבנה:

1- Vertex<KEY,VALUE> vetex (מחלקה גנרית):

אובייקט במחלקה מייצג צומת בעץ שמכילה מידע מסוג VALUE עם מזהה KEY .

שדות: KEY(key),VALUE(value),Vertex(left),Vertex(mother),Vertex(right) .double(Rank),int(BF),int(rightHeight),int(leftHeight),int(height)

2- AVLTree<KEY,VALUE>tree (מחלקה גנרית):

אובייקט במחלקה מייצג עץ דרגות מבוסס AVL שהוצג בכיתה.(נייצג את אחזקת הדרגות בפעולות ודרך החישוב בהמשך)

שדות: Vertex<KEY,VALUE> (max), int(size),Vertex<KEY,VALUE> (root)

3-Custmor:

אובייקט במחלקה מייצג לקוח בחברת התקליטים המזדהה בתעודת זהות ומספר טלפון. לקוח יכול יהיות חבר מעודון,אם כן אז יש לו סכום לתשלום בסוף כל חודש.

שדות: int(c_id),int(c_phoneNumber),bool(isMember),double(dept).

4-Record:

אובייקט במחלקה מייצג תקליט המזדהה בתעודת זהות, לכל תקליט יש מספר עותקים ומשתמש יכול לקנות אותו, נשמור לכל תקליט את מספר העותקים שנמכרו ממנו.

שדות: int(r_id),int(numCopies),int(sold),int(column).

5-HashTable:

אובייקט במבנה מייצג טבלת ערבול לאובייקטים מסוג לקוחות, המשתמש בעצי AVL כפי שהוצג בכיתה ותומך בפעולות חיפוש והכנסה גם כמו שהוצג בכיתה.

שדות : AVLTree<int,Custome*>* (array), int (capacity), int(customersNum).

6-Node:

אובייקט במחלקה מייצג צומת בעץ הפוך שמכיל מצביע לאובייקט מסוג תקליט ומידע נוסף על התקליט, בנוסף כל צומת מצביעה על ההורה שלה בעץ.

שדות: Customer*(customer),int(rank),int(hight),int(size), Node*(mother).

7- UnionFind:

אובייקט במחלקה הוא UnionFind של תקליטים התומך בפעולות חיפוס ואיחוד קבוצות של תקליטים כפי שהוצג בכיתה, במבנה כל קבוצה היא בעצם עץ הפוך ומייצגת עמודה של תקליטים, ומידע נוסף על גובה כל תקליט נשמר בצומת בעץ של התקליט(נראה בהמשך איך פועל המבנה).

שדות: int(size), Node**(recordsArray).

RecorsCompany:

אובייקט במחלקה שלנו מייצג חברת תקליטים, המידע בו נשמר בעזרת המחלקות שהצגנו למעלה באופן הבא:

1. AVLTree<int,Customer*> membersTree:

עץ דרגות מבוסס AVL שמכיל את הלקוחות שהם חברי מועדון, הדרגה בעץ מיועדת לשמירת ההטבות שמקבל חבר מעדון. העץ ממוין לפי תעודות זהות של חברי המעדון ובכל צומת יש מצביע ללקוח.

2. HashTable customersHash:

טבלת ערבול אשר נשמור בה את המידע על כל לקוח.הלקוחות ממוינים לפי תעודות השזות שלהם, פונקציית הערבול מתאימה לכל תעודת זהות את התא המתאים במערך ואז הלקוח נשמר בתוך העץ שנמצא בתא המתאים.

3. UnionFind* recordsUnion :

מצביע על איבר מסוגUnionFind ובUnionFind נשמור את המידע על כל התקליטים, בהתחלה כל תקליט יהיה קבוצה בעצמו, ואז אם נדרש נעשה פעולת איחוד לקבוצות ונייצר ערימות של תקליטים.

4. int numOfRecords:

נשמור במשתנה את מספר התקליטים בחברה.

*****כלומר לסיכום כל התקליטים נמצאים ב recordsUnion והלקוחות נמצאים בcustomersHash ואם הם חברי מעדון אז הם גם ב membersTree.

סיבוכיות מקום

נניח שבחברת התקליטים m הוא מספר התקליטים ו n הוא מספר הלקוחות.

אובייקט מסוג לקוח או תקליט תופס מקום O(1).

1. מספר חברי המעדון הוא לכל היותר כמו מספר הלקוחות הכללי ולכן גודל העץ membersTree יהיה לכל היותר כמו גודל מספר הלקוחות, ולכן העץ תופס מקום O(n) לפי מה שלמדנו בכיתה.

2. טבלת הערבול customersHash מכילה מערך מסדר גודל של מספר הלקוחות, בכל תא מצביע לעץ AVL, לכן המערך תופס מקום O(n), הלקויות מפוזרות בתוך העצים שבתוך המערך וכל לקוח נמצא רק פעם אחת בעץ אחדולכן בסכ"ה יש בהם צמתים כגודל מספר הלקוחות ולכן כל העצים ביחד תופסים מקום O(n).

נקבל שבסכ"ה הטבלה תופסת O(n)+O(n)= O(n) מקום.

3.ב recordsUnion יש מערך בגודל מספר התקליטים ובכל תא יש מצביע לצומת בעץ הפוך, ולכן המערך תופס מקום בגודל O(m), לכל תקליט יש צומת NODE שבה יש מצביע לתקליט, מספר הצמתים הוא כממספר התקליטים ולכן תופסים מקום O(m).

נקבל זבסכ"ה recordsUnion תופס מקום O(m)+O(m)=O(m).

ולכן המקום שדורש כל המבנה הוא:O(n)+O(m)+O(n)=O(n+m).

פעולות וסיבוכיות זמן:

RecordsCompany():

מאתחלים את כל האובייקטים המיוצגים למעלה כאוביקטים ריקים דרך הבנאיים החסרי פרמטרים של AVLTree, UnionFind, HashTable שהסיבוכיות שלהם היא O(1) לכן אתחול כל המבנה בסכ"ה יהיה ב O(1) כנדרש.

~RecordsCompany():

נרצה לשחרר את כל הזכרון שהקצאנו במבנה:

1- membersTree, מחיקת עץ דרגה מבוסס AVL למען זה היעזרנו בפונקציה destroyTree אשר מבצעת סיור inOrder על העץ ומוחקת כל צומת בה היא עוברת, סיור זה עולה סיבוכיות זמן של גודל העץ ולכן סיבוכיות הזמן שלה O(n) כגודל מספר חברי המעדון המקסימלי. את הערכים בתוך העץ שהם מצביעים ללקוחות לא נשחרר עכשיו, כי בהמשך נמחק את כל הלקוחות מהמערכת בעת מחיקת customersHash.

2-customersHash, יש לנו מערך של עצי AVL שמכילים מצביעים ללקוחות, נעבור על כל עץ, נשחרר את הזכרון של הלקוחות בו איבר איבר דרך סיור inOrder ואז ניעזר עוד פעם בפונקציה destroyTree כדי למחוק את העץ עצמו. הסכ"ה נעבור על כל לקוח פעם אחת בעץ שהוא נמתא ונמחק את הלקוח והצומת שהוא בה בעץ ולכן מחיקת כל העצים דורשת זמן מסדר גודל של מספר הלקוחות O(n). בסוף נמחק את המערך שמכיל את העצים, בסכ"ה מחיקת customersHash דורשת זמן O(n).

3- recordsUnion,יש לנו מערך בגודל מספר התקליטים, ובכל תא יש מצביע לצומת המכילה מצביע לתקליט,לאיפוס המבנה ניעזר בפונקציה DELETEUNION שבא נעבור איבר איבר על המערך, נשחרר את המצביע לתקליט, נשחרר את הצומת שהוא נמצא בה, ונעבור לתא הבא, בסכ"ה ביצענו פעולות בזמן O(1) בכל תא וישm תאים ולכן מחיקתrecordsUnion יעלה זמן O(m).

בסכ"ה שחרור כל הזכרון מהמבנה יעלה : O(n)+O(n)+O(m)=O(n+m)

StatusType newMonth(int *records_stocks, int number_of_records):

נבדוק את תקינות הקלט,אחר כך צריך לאפס את הנתונים הישנים ולהוסיף את המידע החדש:

נעדכן את numOfRecords לפי הערך הנתון, נאתחל מערך תקליטים בגודל מספר התקליטים החדש,נעבור על המערך איבר איבר, נאתחל תקליט חדש עם הנתונים החדשים ונוסיף אותו למערך התקליטים. בסיום נעביר את מערך זה לבנאי של UNIONFIND שמקבל מערך של תקליטים ובונה מהם מבנה UNION חדש שבו יש את התקליטים במערך ונשמור מצביע למבנה זה.

כעת נחיף את הUNION של המבנה בUNION החדש ונשתמש עוד פעם בפונקציה DELETEUNION כדי למחוק את הUNION הישן. כל פעולה שתיארנו דורשת מעבר על מערך בגודל מספר התקליטים מספר סופי של פעמים ולכן בסכ"ה עדכון recordsUnion יעלה זמן O(m).

כעת נותר לעבור על חברי המעדון וכל אחד מהם ישלם את החובות שלו ולאפס את המתנות שנותרו לכל אחד,

נעבור על עץ חברי המעדון בסיור INORDER, ולכל לקוח שנעבור בו נאפס את דרגת העץ, את החובות של הלקוח ואת הדרגה הפרטית שיש לו. סיור זה יעלה זמן כגודל העץ שהוא לכל היותר כמספר המשתמשים O(n).

בסכ"ה פעולה זו דורשת זמן O(n)+O(m)=O(n+m).

*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

StatusType addCostumer(int c_id, int phone):

הוספת לקוח חדש לחברה.

בהתחלה בודקים תקינות קלט. לאחר מכן מחפשים אם הלקוח קיים במערכת.

אם הוא לא נמצא נייצר מצביע לאובייקט חדש עם הנתונים בפונקציה ונכניס אותו למבנה שלנו (custmorsHash).

הסבר לסיבוכיות:

בדיקת תקינות קלט O(1)

חיפוש קיום לקוח- הפונקציה find מחפשת את הלקוח במבנה ,הפונקציה עובדת בסיבוכיות הפעולה היא O(1) בממוצע על הקלט.

הכנסת לקוח במידה ואינה נמצא- יש את הפונקציה insert שמכניסה לקוח למבנה ,הפונקציה עובדת בסיבוכיות הפעולה היא O(1) משוערך בממוצע על הקלט.

לכן בסכ"ה נקבל כי סיבוכיות הפונקציה היא היא O(1) משוערך בבמוצע על הקלט.

*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

(##)- הסבר לסיבוכיות הפונקציות find,insert במבנה customersHash:

נסמן num-מספר האיברים (הלקוחות) size-גודל המערך הדינמי

כפי שראינו בהרצאה אם נשמור על פקטור העומס שיהיה $O(1)$ אזי נקבל כי כל הפעולות יתבצעו בסיבוכיות זמן $O(1)$ בממוצע על הקלט. במבנה שלנו יש מערך דינמי של עצי AVL, בחרנו פונקצית ההערבול להיות מודלו גודל המערך שהיא מקיימת את הדרישה של פיזור אחיד. כיוון שמספר האיברים ישתנה ויגדל, המערך הדינמי עוזר לנו לאי פגיעה בסיבוכיות כך שאם הגענו למצב שבו $\frac{num}{size} \geqslant \frac{1}{2}$ אזי מבצעים את פעולת rehash בה מגדילים את המערך פי 2 ונקבל כי בתנאי הזה אנחנו דואגים לשמור $\Omega(num)$ תאים פנויים לכן פעולת rehash תתבצע פעם ב $\Omega(num)$ פעולות insert, לכן בכך שומרים על על פקטור העומס שיהיה $O(1)$ ונקבל כי סיבוכיות הזמן המשוערכת של כל הפעולות היא $O(1)$ בממוצע על הקלט כנדרש.

Output_t<int> getPhone(int c_id):

החזרת מספר הטלפון של לקוח.

בהתחלה בודקים תקינות קלט. לאחר מכן מחפשים אם הלקוח קיים במערכת.

אם הלקוח נמצא במערכת מהחיפוש מקבלים מצביע אליו לכן בעזרת המצביע נחזיר את מספר הטלפון שלו.

הסבר לסיבוכיות:

בדיקת תקינות קלט $O(1)$

חיפוש קיום לקוח- הפונקציה find מחפשת את הלקוח במבנה, הסבר לפונקציה היה ב (##) לעיל חיפוש בטבלת ערבול של עצים לכן סיבוכיות הפעולה היא $O(1)$ בבמוצע על הקלט.

החזרת מס הטלפון בעזרת המצביע - $O(1)$

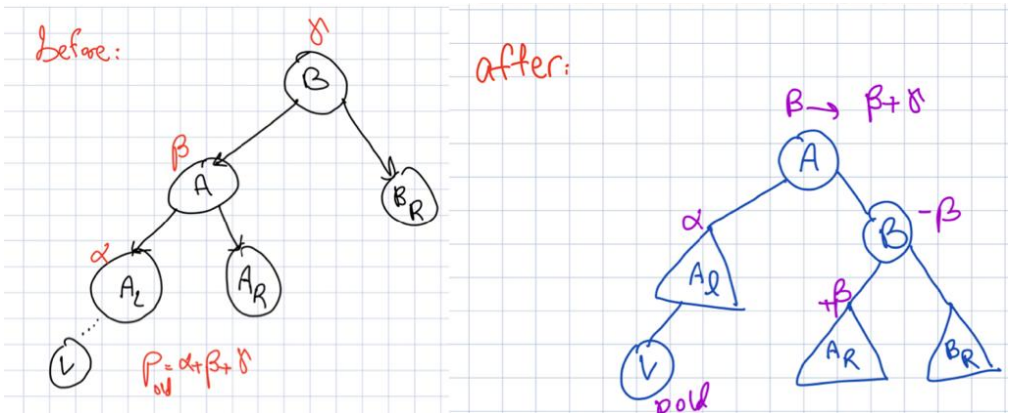
לכן בסכ"ה נקבל כי סיבוכיות הפונקציה היא היא $O(1)$ בבמוצע על הקלט.

*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

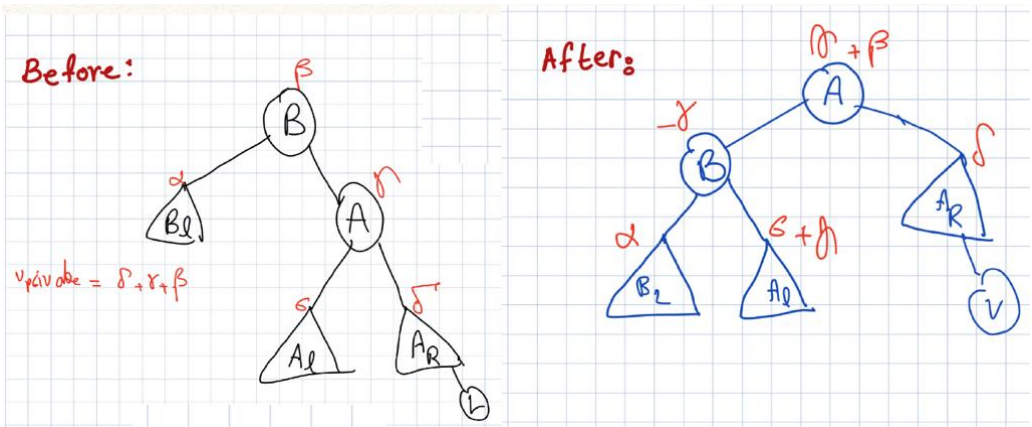
StatusType makeMember(int c_id):

קודם נבדוק תקינות הקלט, נבדוק אם לקוח זה קיים במערכת ע"י חיפוש בטבלת הערבול של הלקוחות (חיפוש בטבלת ערבול מבוססת עצי AVL יעלה במקרה הגרוע O(logn), אם הלקוח קיים, נבדוק אם הוא כבר חבר מעדון דרך השדה IsMember שיש ללקוח, אם לא אז נכניס אותו לעץ חברי המעדון ונסמן את השדה IsMember שלו ב True, כיוון שעץ שלנו הוא עץ דרגות מבוסס AVL אז צריך להבטיח את נכונות הדרגות בעת הכנסת איבר חדש ולכן לכל איבר שמכניסים לעץ פועלים באופן הבא:

- אם הופר איזון העץ והיינו צריכים גלגולים אז נבצע גלגול כרגיל, אך לצומת שביצענו עליה גלגול נשנה את הדרגות לפי הציורים הבאים לגלגול LL:



ולגלגול RR:



- שינוי הדרגות בכל גלגול דורש מספר סופי של פעולות בגודל O(1) ולכן זו אותה סיבוכיות זמן של גלגול רגיל ולכן הכנסת האיבר החדש לעץ היא כמו סיבוכיות הכנסה לעץ AVL רגיל כלומר O(logn).
- אחרי שהכנסנו את הלקוח לעץ חברי המעדון והתייצב המקום שלו, נשים לב שכעת הלקוח לא אמור לקבל הטבות ומתנות, אבל אם נחשב את הדרגה שלו נראה שהיא מושפעת מהדרגות שבמסלול החיפוש שלה, כי לפי הגדרתנו לדרגה של צומת, היא סכום הדרגות במסלול החיפוש של הצומת. כדי לפתור בעייה זו נעבור על מסלול החיפוש של הלקוח בשם privateRank שהכנסנו ונמצא את מה שהיה אמור להיות הדרגה שלו, נשמור סכום זה בשדה פרטי אצל הלקוח ובכל פעם שאנחנו בודקים את דרגתו, נחסיר סכום זה ממה שיצא לנו.
כיוון שבתרגיל זה אין פעולות הוצאה,ו פרס שהוענק יישאר עד סוף החודש ואיפוס הפרסים אז ערך זה יהיה תמיד רלוונטי עד תחילת חודש חדש שבו אנחנו מאפסים את הדרגות וגם את privateRank לכל לקוח.
- חישוב ה privateRank הוא כמו מעבר על מסלול החיפוש של צומת ושמירת הסכום ב O(1) לכן מתבצע בסיבוכיות זמן של חיפוש בעץ AVL שהוא O(logn).
- בסכ"ה ביצענו שלוש פעולות שדורשות זמן O(logn) ולכן סיבוכות הזמן של כל הפעולה היא O(logn)+O(logn)+O(logn)=O(logn)

*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

Output_t<bool> isMember(int c_id):

בדיקה אם הלקוח הוא חבר מועדון.

בהתחלה בודקים תקינות קלט. לאחר מכן מחפשים אם הלקוח קיים במערכת (חיפוש ב customerHash). אם הוא קיים נחפש אם הוא קיים בעץ membersTree ונחזיר את התוצאה בהתאם.

הסבר לסיבוכיות:

בדיקת תקינות קלט O(1)

חיפוש קיום לקוח- הפונקציה find מחפשת את הלקוח במבנה ,הסבר לפונקציה היה ב (##) לעיל. לכן סיבוכיות הפעולה היא O(1) בבמוצע על הקלט.

חיפוש בעץ membersTree כיוון שהעץ הוא עץ דרגות מבוסס AVL לכן כפי שלמדנו מתקיים כי סיבוכיות הפעולה היא O(1) בבמוצע על הקלט.

לכן בסכ"ה נקבל כי סיבוכיות הפונקציה היא היא O(1) בבמוצע על הקלט.

*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

StatusType buyRecord(int c_id, int r_id):

בהתחלה בודקים תקינות קלט. לאחר מכן מחפשים אם הלקוח קיים במערכת (חיפוש ב customerHash). אם הוא קיים נחפש את התקליט המתאים מהמבנה recordsUnion. במידה והלקוח הוא חבר מועדון (בדיקה בעזרת שדה IsMember) נעדכן את שדה ה dept שלו עם הערך המתאים כפי שנדרש. ולבסוף מוסיפים 1 לשדה sold של התקליט.

הסבר לסיבוכיות:

בדיקת תקינות קלט O(1)

חיפוש קיום לקוח- הפונקציה find מחפשת את הלקוח במבנה ,לכן סיבוכיות הפעולה היא O(logn) במקרה הגרוע במידה והיה מקרה שמספר הלקוחות מופו לאותו עץ הוא מסדר גודל של n .

חיפוש קיום תקליט - בעזרת הפונקציה findRecord מקבלים מצביע ל Node שמכיל מצביע ל Record ב O(1) בעזרת גישה למערך עם אינדקס נתון לכן סיבוכיות הפעולה היא O(1).

ולאחר מכן עדכון שדות של התקליט והלקוח שמתבצע ב O(1)

בסכ"ה ביצענו פעולה אחת שדורשת O(logn) ועוד פעולות שדורשות O(1) לכן סיבוכות הזמן של כל הפונקציה היא O(logn) כנדרש.

*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

StatusType addPrize(int c_id1, int c_id2, double amount):

הוספת מתנה לחברי מועדון בטווח מסיום.

חברי המועדון הם בעץ דרגות מבוסס AVL

בהתחלה בודקים תקינות קלט.

נעזרנו בפונקציה אחרת addPrizeHelper שהיא מוסיפה amount מטווח 1 עד ה id שהיא מקבלת אותו כלומר לכל הצמתים שהם קטנים ממש מהצומת id.

השתמשנו ב addPrizeHelper פעמיים בפעם הראשונה עד ל c_id2-1 עם ערך amount ופעם השנייה עם ערך amount- עד ל c_id1-1 כדי שהערכים האלו הם לא בטווח הנדרש.

הסבר לפונקציה addPrizeHelper: (הפונקציה ריקורסיבית)

בהינתן שמקבלים את המפתח של הצומת שרוצים להגיע אליו ואת הצומת שנמצאים בו כרגע

ניעזר במשתנה flag שימסן לנו את צריכים לעדכן את הדרגה או לא (המידע הנוסף) נסמן

curr: הצומת הנוכחי.

curr_id: המפתח של הצומת הנוכחי, c_id : הצומת שרוצים להגיע אליו

נחלק למקרים:

- curr_id == c_id:
  אם flag = false אזי ← curr.Rank+=amount
  אם קיים בן ימני אזי: curr.right.Rank-=amount
- curr_id < c_id:
  אם flag = false אז curr.Rank+=amount
  נקרא לפונקציה עם flag = true והבן הימני של הצומת הנוכחי.
- curr_id > c_id:
  אם flag = true אז curr.Rank-=amount
  נקרא לפונקציה עם flag = false והבן השמאלי של הצומת הנוכחי.

הסבר סיבוכיות:
בפונקציה אנו קוראים לפעולה addPrizeHelper פעמיים.
הפעולה addPrizeHelper היא בעצם סוג של "חיפוש" בעץ AVL אך עם עדכון שדה המידע הנוסף לכן כיוון שהעדכון מתבצע ב O(1) אזי הפעולה addPrizeHelper תיקח סיבוכיות זמן כמו חיפוש רגיל שזה O(logn).
בסכ"ה ביצענו שתי פעולות שדורשות זמן O(logn) ולכן סיבוכות הזמן של הפונקציה היא:
O(logn)+O(logn)=O(logn)
*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

Output_t<double> geptExpenses(int c_id):

החזרת את סך ההוצאות החודשיות ללקוח.
בהתחלה בודקים את תקינות בקלט לאחר מכן מחפשים את הלקוח בעץ membersTree כדי לבדוק אם הלקוח הוא חבר מועדון במידה וכן נחזיר את ה dept שלו שנמצא בנוסף לכך ניעזר בפונקציה findRank שסוכמת לאורך מסלול החיפוש את דרגות הצמתים שמבקרת בהם ואת הסכום - נסמנו Rank ונקבל כי הלקוח קיבל מתנות בסך של:
prices = Rank - PrivateRank
ונקבל כי סך ההוצאות הוא: dept - prices ונחזיר את הערך המתקבל.
הסבר סיבוכיות:
בדיקת תקינות קלט O(1)
חיפוש בעץ membersTree: חיפוש בעץ AVL כרגיל: O(logn)
פונקצית findRank: היא פונקציה שעושה "חיפוש" אך מחזירה את הערך שסכמה אותו לאורך מסלול החיפוש לכן הסיבוכיות שלה היא O(logn).
אירתמטיקה של מספרים וחישובים O(1).
בסכ"ה ביצענו שתי פעולות שדורשות זמן O(logn) ועוד פעולות עם O(1) ולכן סיבוכות הזמן של הפונקציה היא:
O(logn)+O(logn) + O(1) + O(1)=O(logn)
*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)
** הוסבר על ה PrivateRank בפונקצית makeMember.

StatusType putOnTop(int r_id1, int r_id2):

מעבירים את הערימה שנמצא בה r_id1 מעל לערימה שנמצא בה r_id2.
בודקים בהתחלה את תקינות הקלט וקיום התקליטים במערכת לאחר מכן בודקים אם הם נמצאים באותה עמודה אם כל הבדיקות האלה תקינות כלומר ניתן לשים את הערימה שנמצא בה r_id1 מעל לערימה שנמצא בה r_id2 אזי נקרה לפונקציה unite של המבנה unionFind:
הפונקציה unite פועלת באופן הבא:
בהתחלה היא מבצעת חיפוש לשורשי העצים ההפוכים שבהם נמצאים r_id1, r_id2 בתוך פעולת החיפוש מתבצע תהליך כיווץ מסלולים כפי שראינו בתרגול 9 - "שאלת הארגזים" ולאחר שקיבלנו את השורשים מבצעים איחוד לפי גודל שדורש עדכון מצביעים ועדכון שדות של מידע נוסף (O(1)) ולבסוף מעדכנים את העמודה של r_id1 להיות ההעמודה של r_id2
הסבר סיבוכיות:
נשים לב כי כל הפעולות הם O(1) והפעולה העיקרית היא פונקצית ה unite במימוש שלנו עשינו את ה find בתוך פעולת ה unite ושאר הפעולות הם עדכוני מצביעים ושדות לכן סיבוכיות הפעולה putOnTop היא כסיבוכיות הפעולה

find וכפי שלמדנו וראינו ב"שאלת הארגזים" מתקיים כי מספר קבוע של פעולות find דורש סיבוכיות זמן משוערכת של O(log*m) לכן בסכ"ה סיבוכות הזמן של הפונקציה היא: O(log*m)
*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)

StatusType getPlace(int r_id, int *column, int *hight):

החזרת גובה ועמודה של תקליט מסוים.
החזרת העמודה מתבצעת בסיבוכיות זמן של O(1) כי יש שדה column.
החזרת גובה של תקליט מסוים מתבצעת דרך גישה לצומת שבו נמצא התקליט ולאחר מכן סכימה את השדה Rank בכל צומת עד להגעה לשורש העץ ההפוך שהוא נמצא בו והסכום המתקבל יהיה הגובה של התקליט.
הסבר סיבוכיות:
התהליך התבצע בדומה לשאלת הארגזים וכפי שראינו בתרגול כיוון שפעולת ה find מבצעת כיווץ מסלולים לכן נקבל כי אורך המסלול לשורש העץ מתקצר ואז נקבל בגלל כיווץ המסלולים כי סיבוכיות זמן המשוערכת של הפונקציה היא O(log*m) כנדרש.
*בכל שלב מהשלבים לעיל מחזירים את ההערך המתאים לפי המתואר בפונקציה. (StatusType)