**מבני נתונים 1 – 234218 – תרגיל רטוב 1**

**תיאור מבנה הנתונים – עץ AVL**

העץ מורכב ממצביע לשורש, כאשר השורש מטיפוס AVLNode.

הטיפוס Node מורכב ממצביע למידע של הטיפוס, גובהו, וגורם האיזון (Balance Factor) צומת בעץ מכיל את השדות הבאים: תוכן (data), גובה (height), בן שמאלי (right son), בן ימני (left son) וגורם האיזון (balance factor).

**תיאור מחלקות אב המשמשות להשוואה בין טיפוסים**

|  |  |
| --- | --- |
| IsLess | משווה שני טיפוסים מאותו הסוג. |
| IsLessThanKey | משווה בין טיפוס ובין ערך כלשהו מאותו טיפוס. |

**תיאור הפונקציות שמומשו עבור class AVLNode**

|  |  |
| --- | --- |
| getData | מחזיר את הערך של הטיפוס שהחוליה מחזיקה – בסיבוכיות זמן O(1) במקרה הגרוע. |
| setData | מעדכן את הערך של הטיפוס שהחוליה מחזיקה - מעדכן ערך בערך קיים ולכן מתבצע בסיבוכיות זמן O(1). |
| getHeight | מחזיר את גובה החוליה – מחזיר ערך קיים ולכן מתבצע בסיבוכיות זמן O(1). |
| getBalanceFactor | מחזיר את גורמי האיזון של החוליה - מחזיר ערך קיים ולכן מתבצע בסיבוכיות זמן O(1). |
| HeightUpdate | מעדכן את גובה החוליה באמצעות שימוש בתת העץ השמאלי והימני של החוליה – מתבצע לכל היותר בסיבוכיות זמן O(1) שכן משתמש בערכים קיימים. |
| BfUpdate | מעדכן את גורמי האיזון של החוליה – באופן דומה , משתמש בערכים קיימים ולכן מתבצע בסיבוכיות זמן O(1). |

**תיאור הפונקציות שמומשו עבור class AVLTree**

|  |  |
| --- | --- |
| isEmpty | בודק האם העץ ריק, אם כן מחזיר true, אחרת false. הפעולה בודקת רק את ערכו של שורש העץ, ולכן מתבצעת בסיבוכיות זמן O(1). |
| addNode | מוסיף צומת לעץ בצורה שתשמור על תנאי ה – BF של העץ, והסדר הנדרש של עצי חיפוש. במקרה הגרוע, נגיע למקום המתאים ב O(log(n)) פעולות, ונכניס את הצומת החדשה ב O(1) פעולות. לאחר מכן, נקרא לפעולות update ו – rotate, שיבצעו את האיזון בעץ, וימומשו סה"כ במקרה הגרוע בO(log(n)) פעולות. לכן, סה"כ, הפעולה תמומש בסיבוכיות זמן O(log(n)). |
| Insert | פעולה זו תשתמש ב addNode כדי להוסיף צומת חדשה בעל התוכן שנשלח לפעולה זו כארגומנט. הפעולה תמומש ב O(log(n)) פעולות באותו אופן. |
| removeNode | מסיר צומת מהעץ בצורה שתשמור על תנאי ה – BF של העץ, והסדר הנדרש של עצי חיפוש. במקרה הגרוע, נחפש את הצומת ב O(log(n)) פעולות. אם אין לו ילדים, נסיר את הצומת ב O(1) פעולות. אחרת, אם יש לו ילד אחד, נסדר את המצביעים, ונסיר את הצומת ב O(1) פעולות. אם יש לו 2 ילדים, נזמן שוב הפעולה באופן רקורסיבי – יתבצע לכל היותר ב O(log(n)) פעולות. לאחר מכן, נקרא לפעולות update ו – rotate, שיבצעו את האיזון בעץ, וימומשו סה"כ במקרה הגרוע בO(log(n)) פעולות. לכן, סה"כ, הפעולה תמומש בסיבוכיות זמן O(log(n)). |
| Remove | נסיר פעולה זו תשתמש ב removeNode כדי להסיר צומת קיימת בעל התוכן שנשלח לפעולה זו כארגומנט. הפעולה תמומש ב O(log(n)) פעולות באותו אופן. |
| removeAllData | מסיר את כל המידע מהעץ באופן רקורסיבי בשיטת סיור postOrder, כפי שנלמד בכיתה יתבצע בסיבוכיות זמן O(log(n)). |
| searchElement | הפונקציה תשתמש ב searchElementRec למצוא את התוכן של הטיפוס אותו אנחנו רוצים למצוא, לכן תתבצע גם היא בסיבוכיות זמן O(log(n)) במקרה הגרוע. |
| searchElementRec | הפונקציה תחפש את הצומת בשיטת סיור inorder בעץ הממוין – יתבצע בO(log(n)) פעולות לכל היותר, ותחזיר את המצביע לטיפוס שבתוך הצומת. |
| min | הפונקציה תחפש את הצומת המינימלי במסלול, יידרשו O(log(n)) פעולות. |
| max | הפונקציה תחפש את הצומת המקסימלי במסלול, יידרשו O(log(n)) פעולות. |
| update | מעדכן את האיזון שהתבצע בעץ, לשם כך נעשה שימוש בפונקציוצ updateHeight ו -updateBF, ולכן הפעולה מתבצעת לכל היותר בסיבוכיות זמן O(1) במקרה הגרוע. |
| rotate | מבצע את הגלגול בעץ – כלומר החלפת המצביעים. מתבצע לכל היותר בסיבוכיות זמן O(1) במגקרה הגרוע. |
| next | מחזיר מצביע לצומת הבאה בעץ כך שאם אנחנו בתחילת המסלול, כלומר בשורש העץ, הפעולה תתבצע בסיבוכיות זמן O(1), אך עבור צומת פנימי זה יתבצע בסיבוכיות זמן O(log(n)). |
| InorderArray | מחזיר מערך של מצביעים בשיטת סיור inodrer, מתבצע בסיבוכיות זמן O(log(n). |
| ArraySize | מחזיר את אורכו של המערך שאורכו כמספר הצמתים בעץ. הפונקציה סופרת את מספר הצמתים באופן רקורסיבי, מתבצע בסיבוכיות זמן O(log(n)). |
| AssembleArray  AVLtree constructor | הפונקציה תחזיר מערך של מצביעים לצמתים השונים בעץ לאחר סיור inorder מתבצע בסיבוכיות זמן .O(log(n))  מקבל מערך ממוין מהקטן לגדול של מצביעים ל Data בגודל n ויוצר עץ חדש מאוזן ב O(n) ע"י לקיחת אמצע בכל רגע נתון, הפיכתו לשורש מקומי וקביעת הבנים שלו כאמצע של החלק הימני במערך והשמאלי במערך וכך הלאה ברקורסיה. עוברים על כל תא במערך פעם אחת ולכן הסיבוכיות היא O(n) , והגובה הוא O(log n) מכיוון שבכל רמה מחלקים את כמות האיברים שנשארו להוסיף ב 2 מכיוון שמוסיפים את האמצע בכל פעם וכל קריאה רקורסיבית לפונקציה מקבלת חצי מהמערך שהיה, ולכן מקבלים עץ מאוזן עם |BF|≤1 בכל צומת. |

**פירוט מבנה הנתונים**

**DataStructure**

|  |  |
| --- | --- |
| Companies | עץ חברות ממוין לפי מזהה companyID. |
| WorkersByID | עץ עובדים ממוין לפי מזהה workerID. |
| WorkersByRank | עץ עובדים ממוין לפי rank. |
| bestWorkerID | מחזיק את מספר המזהה של העובד הטוב ביותר מבין כל העובדים הקיימים במבנה. |
| bestWorkerRank | מחזיק את דירוגו של העובד הטוב ביותר מבין כל העובדים הקיימים במבנה. |

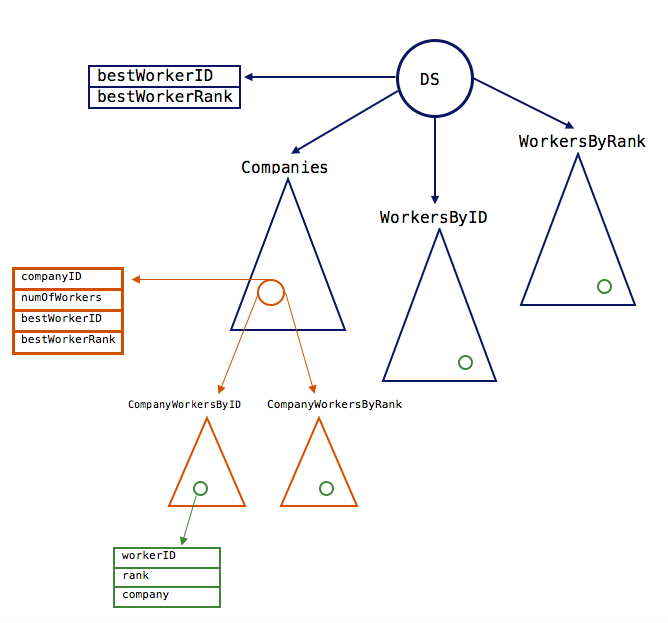
**Class Company**

|  |  |
| --- | --- |
| companyID | מספר מזהה של החברה. |
| numOfWorkers | מספר העובדים בחברה. |
| CompanyWorkersByID | עץ עובדי החברה ממוין לפי מזהה workerID. |
| CompanyWorkersByRank | עץ עובדי החברה ממוין לפי rank. |
| bestWorkerID | מחזיק את מספר המזהה של העובד הטוב ביותר מבין כל העובדים בחברה. |
| bestWorkerRank | מחזיק את דירוגו של העובד הטוב ביותר מבין כל העובדים בחברה. |

**Class Worker**

|  |  |
| --- | --- |
| workerID | מספר מזהה של עובד. |
| rank | דירוג העובד. |
| company | מצביע לחברה אליה שייך העובד. |

**תרשים אבסטרקטי של מבנה הנתונים**

****

**סיבוכיות מקום מבנה הנתונים**

|  |  |
| --- | --- |
| companies | O(k) כאשר k מספר החברות. |
| WorkersByID | O(n) כאשר n מספר העובדים. |
| WorkersByRank | O(n) כאשר n מספר העובדים. |

**תיאור מימוש הפעולות**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **void\* init ()** | | | | | | | |
| *תיאור הפונקציה* | | אתחול מבנה נתונים ריק. | | | | | |
| *דרישת סיבוכיות* | | O(1) | | | | | |
| *ניתוח המימוש* | | מצביע למבנה נתונים ריק או Null במקרה של כישלון. | | | | | |
| *סיבוכיות ממומשת* | | O(1) | | | | | |
| **StatusType addWorker (void \*DS, int workerID, int rank)** | | | | | |  | |
| *תיאור הפונקציה* | | יצירת עובד חדש בעל מזהה workerID ובעל דירוג התחלתי rank, כך שבעת הוספה העובד לא שייך לאף חברה. | | | | | |
| *דרישת סיבוכיות* | | O(log(n)) במקרה הגרוע כאשר n מספר העובדים. | | | | | |
| *ניתוח המימוש* | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * בדיקה האם קיים עובד בעל מזהה workerID, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(n)). במידה וקיים עובד בעל אותו מזהה, נחזיר FAILUR. * יצירת עובד חדש בעל מזהה workerID ודירוג rank. בעת האתחול העובד לא שייך לאף חברה, ולכן שדה company יצביע לNull. פעולה זו מתבצעת בסיבוכיות זמן O(1). * במידה ויצירת העובד נכשלה נחזיר ALLOCATION\_ERROR. * הוספת העובד לעץ workersByID בסיבוכיות זמן O(log(n)). * הוספת העובד לעץ workersByRank בסיבוכיות זמן O(log(n)). * במקרה של הצלחה נחזיר SUCCESS. | | | | | |
| *סיבוכיות ממומשת* | | O(log(n)) במקרה הגרוע כאשר n מספר העובדים. | | | | | |
| **StatusType addCompany (void \*DS, int companyID)** | | | | | | |  |
| *תיאור הפונקציה* | | | | הוספת חברת סטרטאפ חדשה עם מזהה companyID. | | | |
| *דרישת סיבוכיות* | | | | O(log(k)) במקרה הגרוע כאשר k מספר החברות. | | | |
| *ניתוח המימוש* | | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * בדיקה האם קיימת חברה בעלת מזהה companyID, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(k)). במידה וקיימת חברה בעלת אותו מזהה, נחזיר FAILUR. * יצירת חברה חדשה בעלת מזהה companyID. פעולה זו מתבצעת בסיבוכיות זמן O(1). * במידה ויצירת החברה נכשלה נחזיר ALLOCATION\_ERROR. * הוספת העובד לעץ לCompanies בסיבוכיות זמן O(log(k)). * במקרה של הצלחה נחזיר SUCCESS. | | | |
| *סיבוכיות ממומשת* | | | | O(log(k)). | | | |
| **StatusType addworkerToCompany (void \*DS, int workerID, int companyID)** | | | | | | |  |
| *תיאור הפונקציה* | | | | הוספת עובד בעל workerID לחברה companyID. במידה והעובד עבר נמצא בחברה כלשהי, יועבר לחברה החדשה ויימחק מהקודמת. | | | |
| *דרישת סיבוכיות* | | | | O(log(k)+log(n)) במקרה הגרוע כאשר k מספר החברות, וn הוא מספר העובדים. | | | |
| *ניתוח המימוש* | | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * בדיקה האם קיימת חברה בעלת מזהה companyID, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(k)). במידה ולא קיימת חברה בעלתcompanyID , נחזיר FAILUR. * בדיקה האם קיימם עובד בעל מזהה workerID, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(n)). במידה ולא קיים עובד workerID , נחזיר FAILUR. * אם העובד בעל מזהה workerID לא שייך לאף חברה, נשייך אותו לחברה בעלת מזהה companyID על ידי הוספתו לעץ companyWorkersByID בסיבוכיות זמן O(log(n)). כמו כן, נוסיף את העובד גם לעץ companyWorkersByRank בסיבוכיות זמן O(log(n)). * נגדיל את מספר העובדים בחברה באחד, בסיבוכיות זמן O(1). * אחרת, במידה והעובד בעל מזהה workerID שייך לחברה אחרת, נסיר אותו הן מעץ companyWorkersByID והן מעץ companyWorkersByRank של החברה הקודמת בסיבוכיות זמן O(log(n)). ונוסיף את העובד לחברה בעלת מזהה companyID על ידי הוספתו הן לעץ companyWorkersByID והן לעץ companyWorkersByRank בסיבוכיות זמן O(log(n)). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור החברה בסיבוכיות זמן O(log n). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור המבנה נתונים בסיבוכיות זמן O(log n). * במקרה של בעיה בהקצאת זיכרון נחזיר ALLOCATION\_ERROR. * במקרה של הצלחה נחזיר SUCCESS. | | | |
| *סיבוכיות ממומשת* | | | | O(log(k)+log(n)). | | | |
| **StatusType removeWorker (void \*DS, int workerID)** | | | | | |  | |
| *תיאור הפונקציה* | | | | | יש להסיר את העובד workerID מהמערכת. | | |
| *דרישת סיבוכיות* | | | | | O(log(n)) במקרה הגרוע כאשר n מספר העובדים. | | |
| *ניתוח המימוש* | | | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * בדיקה האם קיים עובד בעל מזהה workerID, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(n)). במידה ואינו קיים עובד בעל אותו מזהה, נחזיר FAILUR. * במידה והעובד משויך לחברה כלשהי, בעזרת מצביע company נגיע לחברה אליה שייך, ונסיר אותו הן מעץ companyWorkersByID והן מעץ companyWorkersByRank בסיבוכיות זמן O(log(n)). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור החברה בסיבוכיות זמן O(log n). * בין אם העובד משויך לחברה כלשהי ובין אם לא נסיר אותו הן מעץ workersByID והן מעץ workersByRank בסיבוכיות זמן O(log(n)). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור המבנה נתונים בסיבוכיות זמן O(log n). * במקרה של הצלחה נחזיר SUCCESS. | | |
| *סיבוכיות ממומשת* | | | | | O(log(n)). | | |
| **StatusType mergeCompanies (void \*DS, int companyID1, int companyID2, int minimalRank)** | | | | | | | | |  |
| *תיאור הפונקציה* | | | | | החברות companyID1 וcompanyID2 התאחדו לחברה אחת. מזהה החברה המאוחדת יהיה מזהה של החברה שהכילה יותר עובדים. אם מספר העובדים היה זהה, יילקח המזהה הנמוך מבין השניים. על עובד שדירוגו נמוך מערך הסף minimal Rank לא ימשיך לחברה המאוחדת ויישאר ללא חברה. | | |
| *דרישת סיבוכיות* | | | | | O(n1+n2+log(k)) כאשר n1 מספר העובדים ב companyID1, n2 מספר העובדים בcompanyID2 k מספר החברות. | | |
| *ניתוח המימוש* | | | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * בדיקה האם קיימת חברה בעלת מזהה companyID1, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(k)). במידה ולא קיימת חברה בעלתcomapnyID1 , נחזיר FAILUR. * בדיקה האם קיימת חברה בעלת מזהה companyID2, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(k)). במידה ולא קיימת חברה בעלתcomapnyID2 , נחזיר FAILUR. * נבצע סיור inorder על עצי העובדים של החברה בעלת מזהה companyID1 ונקבל שני מערכים ממוינים, אחד לפי ID ואחד לפי Rank , הפעולה תתבצע בסיבוכיות זמן O(n1). * נבצע סיור inorder על עצי העובדים של החברה בעלת מזהה companyID2 ונקבל שני מערכים ממוינים, אחד לפי ID ואחד לפי Rank , הפעולה תתבצע בסיבוכיות זמן O(n2). * בעזרת הפונקציה mergeWorkerArr נבצע איחוד בין המערכים הממוינים של company1 ו company2 , נקבל מערך ממוין לפי ה ID בגודל size≤n1+n2 ואחד ממוין לפי ה Rank בגודל size , כאשר הפעולה מאחדת בסיבוכיות O(n1+n2) כמו שראינו בהרצאות, וגם לא מוסיפה את העובדים שה rank שלהם קטן מ minimalRank ומשנה את מצביע החברה שלהם ל NULL . * מבדוק לפי מספר העובדים בכל חברה ולפי ה ID כמו שנדרש בתרגיל מי החברה שנשארת ונמחק את הנותרת מ Companies בסיבוכיות O(log k) . * נעדכן את כל העובדים שנשארו במערכים הממוינים להצביע על החברה שנשארת, סיבוכיות O(n1+n2) . * ניצור עץ ממוין לפי rank ואחד לפי ID מהמערכים המאוחדים בעזרת ה constructor של העץ בסיבוכיות O(n1+n2) . * נעדכן את מספר העובדים בחברה ל size ואת עצי העובדים לעצים שעכשיו בנינו בסיבוכיות O(1) . * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור החברה ל max של העץ לפי דרגות שלנו בסיבוכיות זמן O(log (n1+n2))=O(n1+n2). * במקרה של הצלחה נחזיר SUCCESS. | | |
| *סיבוכיות ממומשת* | | | | | O(n1+n2+log(k)). | | |
| **StatusType changeRank (void \*DS, int workerID, int newRank)** | | | | | |  | |
| *תיאור הפונקציה* | | | דירוגו של העובד בעל מזהה workerID השתנתה וכעת היא newRank. | | | | |
| *דרישת סיבוכיות* | | | O(log(n)) במקרה הגרוע כאשר n מספר העובדים. | | | | |
| *ניתוח המימוש* | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * בדיקה האם קיים עובד בעל מזהה workerID, חיפוש בעץ מתבצע בסיבוכיות זמן O(log(n)). במידה ואינו קיים עובד בעל אותו מזהה, נחזיר FAILUR. * נמצא את החברה אליה משויך העובד בעזרת שדה company, בסיבוכיות זמן O(1). * במידה והעובד משויך לחברה כלשהי, בעזרת מצביע company נגיע לחברה אליה שייך, ונסיר אותו הן מעץ companyWorkersByID והן מעץ companyWorkersByRank בסיבוכיות זמן O(log(n)). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור החברה בסיבוכיות זמן O(log n). * בין אם העובד משויך לחברה כלשהי ובין אם לא נסיר אותו הן מעץ workersByID והן מעץ workersByRank בסיבוכיות זמן O(log(n)). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור המבנה נתונים בסיבוכיות זמן O(log n). * נעדכן את שדה הrank עבור העובד בעל מזהה workerID וכעת עלינו להוסיפו בחזרה לכל העצים מהם הסרנו את העובד לפני עדכון הדרגה. * בין אם העובד משויך לחברה כלשהי ובין אם לא נוסיף אותו הן לעץ workersByID והן לעץ workersByRank בסיבוכיות זמן O(log(n)). * במידה והעובד משויך לחברה כלשהי, בעזרת מצביע company נגיע לחברה אליה שייך, ונוסיף אותו הן לעץ companyWorkersByID והן לעץ companyWorkersByRank בסיבוכיות זמן O(log(n)). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור החברה בסיבוכיות זמן O(log n). * נעדכן את ערכי bestWorkerID וbestWorkerRank בעבור המבנה נתונים בסיבוכיות זמן O(log n). * במקרה של הצלחה נחזיר SUCCESS. | | | | |
| *סיבוכיות ממומשת* | | | O(log(n)). | | | | |
| **StatusType getBestWorker (void \*DS, int companyID, int \*workerID)** | | | | | | | | |  |
| *תיאור הפונקציה* | | | | | יש להחזיר את המזהה של העובד בעל הדירוג הכי גבוהה בחברה companyID.  אם companyID<0 יש להחזיר את העובד בעל הדירוג הגבוה ביותר בכל המערכת.  אם לשני עובדים יש דירוג זהה, נחזיר את העובד בעל מזהה workerID הקטן יותר.  אם אין עובדים לחברה בעלת מזהה companyID (או במערכת כולה) נחזיר 1- בworkerID . | | |
| *דרישת סיבוכיות* | | | | | O(log(k)) במקרה הגרוע כאשר k מספר החברות , או O(1) כאשר companyID<0. | | |
| *ניתוח המימוש* | | | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * אם companyID<0 נחזיר את העובד בעל הדירוג הגבוה מבין כל העובדים במערכת, כלומר נחזיר את bestWorkerID השמור במחלקה DataStructure. בסיבוכיות זמן O(1). * אם לחברה בעלת מזהה companyID אין עובדים נחזיר 1- בworkerID , בסיבוכיות זמן O(1). * אחרת, נמצא את החברה בעלת מזהה companyID בעץ החברות companies, חיפוש מתבצע בסיבוכיות זמן O(log(k)). * כלומר נחזיר את bestWorkerID השמור בחברה בעלת מזהה companyID . בסיבוכיות זמן O(1). * במקרה של הצלחה נחזיר SUCCESS. | | |
| *סיבוכיות ממומשת* | | | | | O(log(k)) או O(1) כאשר companyID<0. | | |
| **StatusType getCompanyWorkersByRank (void \*DS, int companyID, int \*\*workers, int \*numOfWorkers)** | | | | | | | | |  |
| *תיאור הפונקציה* | | | | | יש להחזיר את כל העובדים בחברה בעלת מזהה companyID ממוינים על סמך הדירוג שלהם.  כאשר, אם companyID<0 יש להחזיר את העובדים בכל המערכת ממוינים על סמך הדירוג שלהם.  העובדים יוחזרו ממוינים על סמך הדירוג שלהם בסדר יורד, אם לשני עובדים דירוג זהה נמיין אותם בסדר עולה לפי מספר מזהה workerID.  אם אין עובדים השייכים לחברה בעלת מזהה CompanyID (או במערכת כולה) יש להחזיר Null בworkers, ואפס בnumOfWorkers. | | |
| *דרישת סיבוכיות* | | | | | אם companyID<0 אז בO(n) במקרה הגרוע. כאשר n הוא מספר העובדים במערכת.  אחרת, כאשר k הוא מספר החברות ו מספר העובדים נמצאים בחברה בעלת מזהה companyID. | | |
| *ניתוח המימוש* | | | | | * בדיקת תקינות הקלט בסיבוכיות זמן O(1). במידה והקלט אינו תקין נחזיר INVALID\_INPUT. * אם אין עובדים השייכים לחברה בעלת מזהה CompanyID (או במערכת כולה) יש להחזיר Null בworkers, ואפס בnumOfWorkers. נבדוק את עץ העובדים workersByRank, אם ערכו המקסימלי שווה לאפס נחזיר Null בworkers, ואפס בnumOfWorkers. הפעולה מתבצעת בסיבוכיות זמן O(1) * אם companyID<0 נחזיר את העובדים בכל המערכת ממוינים על סמך הדירוג שלהם. העובדים יוחזרו ממוינים על סמך הדירוג שלהם בסדר יורד, אם לשני עובדים דירוג זהה נמיין אותם בסדר עולה לפי מספר מזהה workerID. נבצע סיור inorder בעץ workersByRank, ונשמור את המידע במערך זמני. סיור inorder מתבצע בסיבוכיות זמן O(n).   נקצה מערך, במקרה של כישלון בהקצאה נחזיר את השגיאה ALLOCATION\_ERROR.  נכניס את העובדים הממוינים למערך workersבסיבוכיות זמן O(n).  נסיר את המערך הזמני בסיבוכיות זמן O(1).   * אחרת, נחפש בעץ החברות Companies, את החברה בעלת מזהה companyID. החיפוש מתבצע בסיבוכיות זמן O(log(k)). במידה והחברה בעלת מזהה companyID לא נמצא נחזיר שגיאה FAILURE. ניגש לעץ העובדים של החברה companyWorkersByRank ונבצע סיור inorder, בסיבוכיות זמן . נשמור את המידע במערך זמני. * נקצה מערך, במקרה של כישלון בהקצאה נחזיר שגיאה ALLOCATION\_ERROR. * נכניס את עובדי החברה הממוינים למערך workers בסיבוכיות זמן . נסיר את המערך הזמני בסיבוכיות זמן O(1). * במקרה של הצלחה נחזיר SUCCESS. | | |
| *סיבוכיות ממומשת* | | | | | או O(n) כאשר companyID<0. | | |
| **void quit (void\*\* DS)** | | | | | |  | |
| *תיאור הפונקציה* | שחרור מבנה הנתונים. יש להציב את הערך Null בDS, אף פעולה לא תיקרא לאחר מכן. | | | | | | |
| *דרישת סיבוכיות* | O(n+k) כאשר n הוא מספר העובדים וk הוא מספר החברות. | | | | | | |
| *ניתוח המימוש* | * שחרור עץ Companies בסיבוכיות זמן O(k), כך שלכל חברה נשחרר את companyWorkersByRank בסיבוכיות זמן O(n). * שחרור עץ workersByID בסיבוכיות זמן O(n). * שחרור עץ workersByRank בסיבוכיות זמן O(n). | | | | | | |
| *סיבוכיות ממומשת* | O(n+k). | | | | | | |