**ממ"ן 18**

עבודה זו כוללת את המסמכים והקבצים הבאים:

1. (מסמך זה) קובץ הסבר למימוש ב-JAVA.
2. קבצי JAVA של הפרוייקט (מן הסתם, ניתן גם לקריאה במעבד תמלילים).
3. קובץ JAR להרצת המחלקה.
4. קובץ word ובו הדפסת פלטים של ריצות נסיון של המימוש.

**הסבר כללי**

בעבודה זו בחרתי *לייצג את בעיית הבנק הנתונה באמצעות 3 מבני נתונים:*

*עא"ש ראשי - מטיפוס RBClientTree - שמחזיק את כל הלקוחות בבנק. העץ מסודר לפי מפתח של מספר לקוח – מפתח ייחודי ובכל צומת יש מצביעים לצמתים המקבילים במבני הנתונים הבאים.*

*עא"ש משני - מטיפוס RBMaxTree- שמחזיק את כל הלקוחות ומסודר לפי מפתח של יתרות של לקוחות.*

*רשימה דו- מקושרת מטיפוס LinkedList ובה רשימת כל הלקוחות שנמצאים ביתרת חוב.*

*מטרת העבודה היא ליישם תכנית לניהול בנק באופן יעיל מבחינת סיבוכיות זמן הריצה של התכנית. המשתמש מזין למערכת שורה של נתונים לגבי לקוחות הבנק וכן בקשות שונות בקשר לנתונים אלו.*

*הנחת העבודה היא כי אין הגבלה על גדלי הקלטים, כלומר על היקף הלקוחות בבנק ולכן - בחרתי את המתודות ומבני הנתונים לפיסיבוכיות זמן הריצה שלהם באופן אסימפטוטי. ייתכן, כי על קלטים בהיקף מצומצם ניתן היה למצוא מתודות אחרות – יעילות יותר עבור קלטים מוגבלים.*

***הסבר אודות המחלקות השונות והמתודות שבהן –***

**Cell.java –**

*מחלקה המיועדת לייצוג תא ברשימה דו-מקושרת. המחלקה מעוצבת במיוחד לקבל מידע מטיפוס* KeyClient, *הוא הטיפוס של מפתחות הצמתים בעא"ש הראשי.*

*כל מתודות הזנת נתונים לתוך תא וגם המתודה של הפיכת התא למחרוזת כוללות מספר קבוע של פעולות ולכן מתבצעות בסיבוכיות של Θ(1).*

**LinkedList.java –**

רשימה דו-מקושרת (עם משתנה המעיד על גודלה של הרשימה). הדיון בבחירת מבנה נתונים הזה איננו מורכב. מבנה הנתונים מיועד כדי שבמענה על שאילתא מסויימת – נדפיס את כל איברי הרשימה ברצף. מטבע ההגדרה הזו, יש לעבור על כל k הלקוחות שיתרתם שלילית ולכן סיבוכיות בזמן היא Ω(k).

עוד משתמע מההגדרה האמורה, כי יש צורך במבנה נתונים דינאמי, שכן מספר הלקוחות המחזיקים ביתרה שלילית עשוי להשתנות מעת לעת ואין חסם עליון או תחתון על מספר הלקוחות הללו.

מבנה נתונים שעונה על שתי הדרישות הללו הוא רשימה דו מקושרת. מעבר על כל איברי הרשימה נעשה בסיבוכיות של Θ(k) והרשימה הינה, מן הסתם, דינאמית.

בחרתי ברשימה דו מקושרת בשל יתרונה במחיקת תא מהרשימה, בסיבוכיות זמן קבועה Θ(1).

במחלקה המתודות הבאות:

בנאי, מתודת עזר לבדיקת גודל הרשימה, ו-gettrs – כל אלה מבצעים מספר פעולות קבועות ובלתי תלויות בגודל הרשימה ונעשות בסיבוכיות זמן קבועה Θ(1).

בנוסף, ישנן גם המתודות הבאות:

**public** **void** **InsertAtEnd**(KeyClient Key) {

מתודת הכנסה בסוף הרשימה. שומרת על סדר כרונולוגי של הכנסת האיברים ומתבצעת בסיבוכיות זמן קבועה Θ(1).

**public** **void** **Delete**(Cell cell) {

מתודת מחיקת תא מהרשימה – המתודה מקבלת תא ומוחקת אותו. בזכות העובדה כי הרשימה היא דו-מקושרת – המחיקה נעשית בסיבוכיות זמן קבועה Θ(1).

**public** String **toString**() {

המתודה העיקרית. ממירה את הרשימה למחרוזת לצורך הדפסה. לשם כך, עוברת המתודה על כל איברי הרשימה וממירה אותם. סיבוכיות זמן Θ(k) (כזכור, k הוא מספר הלקוחות שיתרתם שלילית).

נדגיש, כי במקרה הגרוע, הרי שכל לקוחות הבנק הינם בעלי יתרה שלילית ולפיכך במקרה זה תפעל המתודה בסיבוכיות זמן של Θ(n).

RedBlackNode<T extends Comparable<T>>

זוהי מחלקת הבסיס לייצוג צומת בעא"ש. המחלקה יכולה לקבל כל טיפוס גנרי שניתן לבצע עליו פעולת השוואה (אחרת, אם לא ניתן להשוות בין עצמים, כיצד ניתן להכניס אותם לעח"ב?(

כל צומת כולל את השדות הבאים:

צבע – הצבעים "שחור" ו"אדום" מיוצגים על ידי הספרות "0" או "1" בהתאמה.

מפתח – מיועד להחזיק את המידע שבצומת הכולל גם את השדה המבחין בין צמתים .

ושדות המפנים לצמתים אחרים – left, right ו-parent – כמקובל בעח"ב.

במחלקה מספר מתודות פשוטות – בנאי, getters ו-setters והמרה למחרוזת, כולן פועלות בסיבוכיות זמן קבועה Θ(1).

RedBlackTree.java –

מחלקה זו הינה מימוש בקוד Java של האלגוריתמים הרגילים של עץ אדום שחור – ממומשים באופן זהה לאלגוריתמים שבספר – הכנסה; תיקון צביעה בהכנסה; רוטציה ימינה ושמאלה; אלגוריתם למציאת צומת עוקב / קודם לצומת הנתון; אלגוריתם למחיקת צומת ואלגוריתם לתיקון צביעה; אלגוריתמים למציאת מינימום ומקסימום בעץ וכמובן – אלגוריתם חיפוש בעץ. סיבוכיות הזמן של מתודות אלו זהה לסיבוכיות של האלגוריתמים בספר הלימוד.

נשים לב לשני עניינים – האחד, הכנסת צומת לעץ נעשית על ידי הכנסת מפתח מצד הלקוח; לעומת זאת – מחיקת צומת נעשית על ידי מחיקת הפניה לצומת ספציפי מצד הלקוח. זאת, לנוכח המימוש של המחלקה העוסקת בעיבוד הקלט.

השני, על מנת לפשט את מתודות התיקון והצביעה מחדש, השתמשתי המצביע כללי NIL בתור הצומת השחור המהווה את העלים הסופיים בעץ וגם אביו שורש העץ.

בשולי הדברים מצויות במימוש מתודה isNIL בה אני משתמש באופן תדיר כדי לדעת האם לצומת יש בנים / הורה ומתודה InorderWalk בה השתמשתי באופן פנימי לבדיקת העץ.

KeyClient.java -

מחלקה המהווה מפתח לעא"ש הראשי בו יאופסנו כל הלקוחות בסידור לפי השדה של מספר לקוח ייחודי. הסיבה שהעא"ש האמור מוגדר כראשי נעוצה בכך שמכל צומת שבו, ניתן להגיע בסיבוכיות זמן קבועה ליתר מבני הנתונים. כל צומת מחזיק מצביע לכל אחד מהמבנים הללו כפי שיוסבר להלן.

למחלקה השדות הבאים:

שם הלקוח, מספר חשבון בנק ייחודי, תעודת זהות, יתרה בבנק וכן שני מצביעים. מצביע אחד לעא"ש המסודר לפי יתרות ומצביע לרשימת הלקוחות שביתרת חוב.

כל המתודות במחלקה פועלות בסיבוכיות זמן קבועה Θ(1).

החשובה שבמתודות היא מתודת ה-compareTo אשר מאפשרת השוואה בין איברים מטיפוס הנ"ל.

- KeyMaxTree.java

כשמו כן הוא, זהו המפתח לעא"ש המחזיק את מפתחות הלקוחות לפי סידור לפי יתרותיהם.

כל אובייקט מכיל את השדות – שם, מספר לקוח ויתרה.

המתודה compareTo משתמשת על מנת שנוכל להשוות בין שי אובייקטים והיא החשובה מבין המתודות במחלקה.

כל המתודות במחלקה פועלות בסיבוכיות זמן קבועה Θ(1).

RBClientTree.java –

עא"ש ובו צמתים המסודרים לפי מספרי לקוח ייחודי.

המחלקה יורשת את המתודות של RedBlackTree ומשתמשת ב-KeyClient כמפתח.

RBMaxTree.java –

עא"ש ובו צמתים מסודרים לפי יתרות הלקוחות בבנק. בנוסף, בעא"ש משתנה מסוג צומת RBMaxTree המצביע על הצומת עם היתרה המקסימלית.

מטרת מבנה הנתונים הזה היא לתחזק שמירה של הלקוח בעל החשבון המקסימלי בבנק. לכן, בכל הכנסה או מחיקה של לקוח, עם מחיקת ערכו בעא"ש היתרות הזה, נחפש מיהו המקסימום החדש. לפיכך, דרסנו את שגרות ההכנסה והמחיקה לעץ והוספנו להם את פעולת חיפוש המקסימום:

**public** **void** **insert**(Comparable key) {

**public** **void** **remove**(RedBlackNode z) {

סיבוכיות זמן הריצה של מתודות המחיקה וההכנסה החדשות איננו משתנה ביחס לסיבוכיות זמן הריצה של המתודות המקוריות. שכן, הוספנו פעולה של חיפוש צומת מקסימלי לכל אחת מהן, כלומר למתודה הפועלת בסיבוכיות Θ(lgn) הוספנו פעולה של Θ(lgn) ובסה"כ סיבוכיות Θ(lgn).

בנוסף, קיימת מתודה המחזירה את הצומת ששדה הצומת המקסימלי מצביעה עליה, הפועלת, כמובן בסיבוכיות זמן קבועה Θ(1).

הייתי יכול לבחור במבנה נתונים אחר על מנת לתחזק מעקב אחרי הלקוח המקסימלי בבנק. אולם באף מבנה אחר לא הייתי מגיע לתחזוקה בסיבוכיות של Θ(lgn) כמוסבר לעיל.

מבנה הנתונים המתבקש לשמירה על איבר מקסימלי הינו ערימת מקסימום. אולם בענייננו, הערימה חייבת להיות דינאמית, כלומר אינה בנויה על מערך בגודל קבוע. בכל מימוש של ערימה שכזו ניתקל בקושי. אם נממש את הערימה על בסיס רשימה מקושרת – הרי שבמהלך שגרות התחזוקה (Maxheapify, HeapIncreaseKey) הגישה לתא מסויים ברשימה איננו נעשה בזמן קבוע כמו במערך אלא בתלות לינארית בגודל הרשימה.

אם נשתמש במערך המוגדל מעת לעת, מכיוון שאין הגבלה על מספר הלקוחות, הרי שפעולת הגדלת המערך היא הכרחית, והיא כוללת את העתקת כל איברי המערך למערך החדש ומכאן שאסימפטוטית, אנחנו שוב בתלות לינארית במספר ללקוחות הבנק.

גם האפשרות של החזקת רשימה מקושרת ובה היתרות, כאשר בכל פעם משתמשים באלגוריתם Select למציאת האיבר הגדול ביותר – גם היא נעשית בסיבוכיות זמן לינארית במספר הלקוחות בבנק.

BankManagementIO.java –

המחלקה הזו היא המעבד של הפעולות הנדרשות. היא מחזיקה כשדות את מבני הנתונים השונים הדרושים לנו כמוסבר לעיל. היא מקבלת את הקלט ומפעילה את הפעולות הנדרשות לתחזוקת מבנה הנתונים. המתודות במחלקה הן:

**public** **BankManagementIO**(){

בנאי לאתחול המחלקה ומבני הנתונים שבה.

**public** **void** **ReceiveInput**(String input){

מחלקה לקבלת מחרוזת קלט ממשתמש. מפצלת את הקלט למערך מחרוזות ומעבירה לטיפול המתודה הבאה, הבוררת את הפעולה הדרושה להלן. פועלת בסיבוכיות זמן קבועה Θ(1).

**public** **void** **ReceiveInput**(String input){

מחלקה המקבלת מערך מחרוזות שהוא הקלט מהמשתמש ומצמידה לו את המתודה המטפלת בפעולה הנדרשת.

**public** **void** **InsertClient**(String [] input){

מתודה זו אחראית על הפעולה של הכנסת לקוח חדש, לדוגמא:

+John Doe 015546134 3462211 300

המתודה בונה מפתח מטיפוס Keyclient ומפתח מטיפוס KeyMaxTree ומכניסה אותם ל-RBClientTree ול-RBMaxTree בהתאמה. לאחר הכנסת מפתח ל-KeyMaxTree נציב מצביע מהצומת ב-RBClientTree לצומת המתאים ב-RBMAxTree.

יש לפנינו שתי פעולות של הכנסה לעא"ש ולכן סיבוכיות זמן הריצה היא Θ(lgn).

בכניסת לקוח לבנק, לעולם יתרת הלקוח חיובית ולכן אין צורך להתעסק ברשימת הלקוחות שביתרת חוב.

**public** **void** **RemoveClient**(String [] input){

מתודה זו אחראית על הפעולה של הוצאת לקוח מהבנק, לדוגמא:

- 3462211

על בסיס המידע שנתקבל מהמשתמש נבנה מפתח לצורך חיפוש הלקוח בעא"ש RBClientTree. מספר הלקוח שניתן הוא מספיק על מנת למצוא את הצומת המתאים ולכן יתר השדות במפתח החיפוש יאותחלו לכלום.

בסה"כ – נעשות שתי פעולות מחיקה בסיבוכיות Θ(lgn) כל אחת, ובסה"כ – Θ(lgn).

שוב, גם במחיקת לקוח מהבנק, הרי שבטרם המחיקה אופסה יתרת הלקוח בבנק ולכן אין צורך להתעסק ברשימת הלקוחות שביתרת חוב.

**public** **void** **Query**(String [] input){

מקבלת מחרוזת שיכולה להוות כל אחת מ-3 השאילתות האפשריות במבנה הנתונים הנדון ובוחרת בשאילתא המבוקשת.

**public** **void** **Max**(){

מדפיסה את הלקוח עם היתרה המקסימלית. כאמור לעיל, מדובר בפעולה של החזרה צומת ממצביע כלומר פעולה הנעשית בסיבוכיות זמן קבועה Θ(1).

**public** **void** **Minus**(){

מדפיסה את רשימת הלקוחות שנמצאים ביתרת חוב. כאמור לעיל, השגרה מתבצעת בסיבוכיות זמן ריצה Θ(k) כש-k הוא מספר הלקוחות שנמצאים ביתרת חוב. במקרה הגרוע k = מספר הלקוחות בבנק ולכן במקרה הגרוע הסבוכיות היא Θ(n).

**public** **void** **SearchClient**(Integer ClientNumber){

המתודה מקבלת מספר לקוח בבנק מחפשת אותו ומחזירה את פרטיו. שוב, החיפוש הוא לפי מפתח פיקטיבי המכיל רק את השדה החשוב לנו – מספר לקוח. סיבוכיות זמן ריצה היא כסיבוכיות זמן ריצה של חיפוש בעא"ש Θ(lgn).

**public** **void** **DepositWithdrawl**(String [] input){

המתודה המרכזית בענייננו.

זוהי מתודת התחזוקה של הפקדות ומשיכות כספים בבנק. בכל הפקדה ומשיכה יש לעדכן את RBClientTree ו-RBMaxTree. תחילה, נחפש בעץ הראשי את הצומת המתאים. נבנה מפתח חדש מטיפוס KeyClient ובו השינויים הדרושים ביתרה. בנתיים נבנה גם מפתח חדש עבור עץ היתרות. נשים לב, אין לשנות את ערך היתרה בצומת הרלוונטי בעץ היתרות. שינוי כזה עלול להרוס את חוק עץ החיפוש הבינארי בעץ זה. לכן, יש להשתמש במצביע שבצומת בעץ הראשי כדי לגשת ישירות לצומת הרלוונטי בעץ היתרות ולמחוק אותו. נשים לב, שגם חיפוש בעא"ש לא היה מועיל לנו במציאת הצומת הדרוש, שכן ייתכנו צמתים רבים עם אותה יתרה. בהמשך, לאחר מחיקת הצומת מעץ היתרות נכניס אליו את הצומת החדש ונצביע עליו במפתח החדש שאנחנו בונים.

לאחר מכן, יש לבדוק האם יש צורך להתעסק ברשימת הלקוחות שביתרת חוב.

המקרים בהם נתערב ברשימה הם –

1. מקרה בו לקוח היה בעל יתרה חיובית ובעקבות משיכת יתר הגיע למצב של יתרה שלילית. במקרה זה נוסיף תא חדש לרשימה ובו מפתח הלקוח וגם נעדכן במפתח החדש מצביע לתא המתאים ברשימה.
2. מקרה בו הלקוח היה בעל יתרה שלילית ובזכות הפקדה יתרתו הפכה חיובית – נמחק את התא הרלוונטי ברשימה ונציב null בשדה המצביע על הרשימה במפתח בצומת בעץ במפתח החדש.
3. מקרה בו הלקוח היה בעל יתרה שלילית ובעקבות משיכה / הפקדה נותר ביתרה שלילית. נעדכן את התא הרלוונטי ברשימת הלקחות ביתרת חוב ולא נשה דבר במצביע במפתח.

לבסוף, נעדכן את המפתח החדש בצומת בעץ.

מבחינת סיבוכיות זמן הריצה, הרי שאנחנו מטפלים פה אך ורק בפעולות של חיפוש והוספה של צומת לעא"ש – Θ(lgn) והוספת תא לרשימה מקושרת - Θ(1) ועוד פעולות על מצביעים ומפתחות בסיבוכיות זמן קבועה. לכן סיבוכיות הזמן של המתודה היא Θ(lgn).

InputBank.java –

מחלקה לקבלת קלט מהמשתמש. פעולת באמצעות המחלקה Scanner של java.