מבני נתונים- תרגיל מעשי 1

עץ אדום שחור- מסמך תיעוד

דמיטרי אורלוב, ת.ז. 321892366, שם משתמש- dmitruyo, מייל- dmitruyo@post.tau.ac.il

נעם מזור, ת.ז. 305373458, שם משתמש- noammazor, מייל- noammaz@gmail.com

תיאור המחלקה:

- **מחלקת עץ אדום שחור**

```
public class RBTree {
        private int size=0;
        private RBNode root;
        private RBNode min;
        private RBNode max;
```

המחלקה מגדירה יישות חדשה של עץ אדום שחור עם התכונות המתאימות.
הישות החדשה מטיפוס עץ אדום שחור מוגדרת על ידי 4 פרמטרים המופיעים בתור שדות המחלקה:
שדה size מטיפוס מספר שלם int המכיל מידע על גודל העץ הנוכחי(במונחים של מספר הצמתים).
שדה root מטיפוס צומת של עץ אדום שחור RBNode המצביע על שורש העץ.
שדה min המצביע לצומת עם המפתח המינימלי.
שדה max המצביע לצומת עם המפתח המקסימלי.

- **מחלקת צומת של עץ אדום שחור**

```
public class RBNode{
        public RBNode father;
        public RBNode Left;
        public RBNode Right;
        public int key;
        public boolean red;
```

המחלקה מגדירה יישות חדשה של צומת של עץ אדום שחור עם התכונות המתאימות.
הישות החדשה מטיפוס צומת של עץ אדום שחור מוגדרת על ידי 5 פרמטרים המופיעים בתור שדות המחלקה:
שדה father מטיפוס צומת של עץ אדום שחור RBNode המצביע על האב של הצומת הנוכחית.
שדה Left מטיפוס צומת של עץ אדום שחור RBNode המצביע על הבן השמאלי של הצומת הנוכחית.
שדה Right מטיפוס צומת של עץ אדום שחור RBNode המצביע על הבן הימני של הצומת הנוכחית.
שדה key מטיפוס מספר שלם int המכיל מידע על ערך המפתח הנמצא בצומת הנוכחית.
שדה red מטיפוס בולאני boolean המכיל מידע על צבע הצומת הנוכחית (אמת אם הצומת בצבע אדום).

תיאור וסיבוכיות זמן ריצת המתודות כתלות במספר האיברים בעץ:

פונקציות של מחלקת RBNode:

```
public RBNode getSon(boolean right)
```

הפונקציה מקבלת פרמטר right מטיפוס בולאני המסמן את הבן אותו נדרש להחזיר(ימני אם אמת, ושמאלי אם לא) ומחזירה את המצביע לבן המתאים של הצומת הנוכחית. הקריאה לפונקציה מתבצעת מהצומת הנוכחית והמידע על הבנים של אותה

הצומת מאוחסן בתוך שדות של הצומת. לכן הפונקציה מבצעת מספר קבוע של פעולות עבור הקריאה. לכן סיבוכיות זמן ריצת הפונקציה הינה O(1).

```
public void setSon(boolean right,RBNode node)
```

הפונקציה מקבלת פרמטר right מטיפוס בולאני המסמן את הבן אותו נדרש להציב(שמאלי או ימני) לצומת הנוכחית. פרמטר נוסף של הפונקציה מטיפוס RBNode המצביע לצומת המיועד להיות בן של הצומת הנוכחית. הפונקציה מציבה את הצומת בתור בן מתאים של הצומת הנוכחית.
הקריאה לפונקציה מתבצעת מהצומת הנוכחית והמידע על הבנים של אותה הצומת מאוחסן בתוך שדות של הצומת. לכן הפונקציה מבצעת מספר קבוע של פעולות עבור הקריאה. לכן סיבוכיות זמן ריצת הפונקציה הינה O(1).

פונקציות של מחלקת RBTree:

```
public boolean empty()
```

הפונקציה מחזירה "true" אם העץ הנוכחי הינו עץ ריק ו-"false" אחרת.
מידע זה מתקבל על סמך גודל העץ השמור בתור שדה size של המחלקה. הפונקציה מבצעת בדיקה בודדת האם גודל העץ שווה ל-0. לכן סיבוכיות זמן ריצת הפונקציה הינה O(1).

```
public int search(int i)
```

הפונקציה מפעילה את פונקציית העזר searchNode ומחזירה 1- אם העץ בו מתבצע החיפוש הינו עץ ריק, אחרת הפונקציה מחזירה את ערך המפתח הנמצא בצומת המתאים. ערך זה זהה לערך הפרמטר i אם הפרמטר נמצא בעץ, אחרת ערך זה הינו ערך הכי קרוב לפרמטר i(אך שונה ממנו).
סיבוכיות זמן ריצת הפונקציה searchNode הינה O(log(n)) ולכן סיבוכיות זמן ריצת פונקציית search היא גם O(log(n)), זאת כיוון שהיא מבצעת מספר קבוע של פעולות בנוסף לקריאה לפונקציית searchNode.

```
private RBNode searchNode(int i)
```

הפונקציה מקבלת פרמטר i מטיפוס מספר שלם ומחפשת את הצומת בעץ המכילה את הפרמטר i בתור מפתח. הפונקציה מחזירה את הצומת המכיל את הפרמטר i אם הוא נמצא בעץ, אחרת היא מחזירה את הצומת המכיל את המפתח בעל הערך הקרוב ביותר לפרמטר i(אך שונה ממנו).
במקרה הכי גרוע הפונקציה תבצע O(log(n)) פעולות ולא תמצא את הצומת המכיל את הפרמטר הדרוש, אם הצומת המכיל את האיבר הקרוב ביותר הוא בשורש- היא תצטרך לבצע O(log(n)) פעולות. בסה"כ סיבוכיות זמן ריצת הפונקציה הינה O(log(n)).

```
public int insert(int i)
```

הפונקציה מקבלת פרמטר i מטיפוס מספר שלם ומוסיפה את הפרמטר לתוך עץ אדום שחור אם הפרמטר אינו נמצא באחד הצמתים של העץ. אחרת הפונקציה אינה מוסיפה את הפרמטר לעץ. הפונקציה מחזירה את מספר הסיבובים שנדרשו ע"מ לשמור על תקינות העץ.
הפונקציה מחפשת את המקום המתאים ביותר להכנסת i, אם בדרך היא מוצאת צומת עם ערך זה היא עוצרת. סיבוכיות תהליך זה הוא O(log(n)). לאחר הוספת צומת חדשה לעץ, הפונקציה בודקת את חוקיות העץ החדש שנוצר לאחר ההוספה, כיוון שנוספה צומת אדומה, הכלל האדום עלול לא להתקיים לאחר ההוספה. לכן הפונקציה קוראת לפונקציית עזר fixTree המתקנת את העץ לפי המקרים שנלמדו בכיתה. סיבוכיות של פונקציית fixTree הינה O(log(n)). בנוסף, הפונקציה מעדנכת את המינימום והמקסימום אם נדרש, בסיבוכיות O(1) (בודקת אם הערך החדש קטן מהמינימום או גדול מהמקסימום).
לכן הסיבוכיות הכוללת של פונקציית insert הינה O(log(n)).

```
public int fixTree(RBNode newNode)
```

הפונקציה מתקנת את העץ לפי הצורך. אם הכלל האדום מופר, היא צובעת מחדש את הצמתים ומבצעת סיבובים כך שהכלל האדום ישמר.

סיבוכיות הצביעה מחדש הינה O(1) כיוון שחסומה ע"י מספר קבוע של פעולות בכל קריאה. סיבוכיות ביצוע הסיבוב הינה O(1) כיוון שבכל סיבוב מתבצע מספר קבוע של פעולות של שינויי צבע והעברת מצביעים מתאימים. במקרה הגרוע ביותר ידרש לבצע O(log(n)) שינויי צבע ע"מ לתקן את העץ. לכן הסיבוכיות הכוללת של הפונקציה הינה O(log(n)).

```
public int delete(int i)
```

הפונקציה מקבלת פרמטר i מטיפוס מספר שלם ומוחקת את הפרמטר מתוך העץ אם העץ מכיל את הפרמטר בתוכו. הפונקציה מחזירה את מספר הסיבובים שנדרשו ע"מ לשמור על תקינות העץ לאחר המחיקה. אם הפרמטר אינו נמצא בעץ, הפונקציה מחזירה 0.
הפונקציה מפעילה את פונקציית עזר searchNode ע"מ למצוא את הצומת אותה נדרש למחוק. סיבוכיות פונקציית העזר הינה O(log(n)). לאחר מכן הפונקציה מוחקת את הצומת ע"י העברת מצביעים מתאימים. כמו כן, נדרש לשמור על תקינות העץ ולכן הפונקציה מבצעת את הרוטציות הדרושות ע"מ להחזיר את העץ לתקינותו לאחר המחיקה. סה"כ מספר התיקונים עבור פעולת מחיקה בודדת הינו O(log(n)). בנוסף, הפונקציה מעדכנת את המקסימום והמינימום אם נדרש בסיבוכיות של O(log(n)) במקרה הגרוע. לכן הסיבוכיות הכולל של פונקציית delete הינה O(log(n)).

```
public int min()
```

הפונקציה מחזירה את האיבר המינימלי בעץ. אם העץ ריק, הפונקציה מחזירה 1-. הפונקציה מחזירה את ערך המפתח של הצומת שהשדה min מצביע אליה. (או 1-אם הוא מצביע על null) לכן הסיבוכיות הכוללת של פונקציית min הינה O(1).

```
private RBNode subTreeMin(RBNode node)
```

הפונקציה מקבלת פרמטר מסוג RBNode המסמן צומת ממנה מתחילה פעילות הפונקציה. הפונקציה ומבצעת סריקה של העץ החל מצומת הנוכחית אל העלה הכי שמאלי של העץ. בעלה זה נמצא ערך מינימלי. אם העץ ריק – מוחזר ערך null. אחרת מוחזר צומת בעל מפתח הכי קטן בעץ.
במקרה הגרוע ביותר הפונקציה מתחילה מהשורש של העץ ותרד עד לעלה הכי שמאלי של העץ. סיבוכיות זמן ריצת הפונקציה במקרה זה הינה O(log(n)).

```
public int max()
```

הפונקציה מחזירה את האיבר המקסימלי בעץ. אם העץ ריק, הפונקציה מחזירה 1-. הפונקציה מחזירה את ערך המפתח של הצומת שהשדה max מצביע אליה. (או 1-אם הוא מצביע על null) לכן הסיבוכיות הכוללת של פונקציית זו הינה O(1).

```
private RBNode subTreeMax(RBNode node)
```

הפונקציה מקבלת פרמטר מסוג RBNode המסמן צומת ממנה מתחילה פעילות הפונקציה. הפונקציה ומבצעת סריקה של העץ החל מצומת הנוכחית אל העלה הכי ימני של העץ. בעלה זה נמצא ערך מקסימלי. אם העץ ריק – מוחזר ערך null. אחרת מוחזר צומת בעל מפתח הכי גדול בעץ.
במקרה הגרוע ביותר הפונקציה מתחילה מהשורש של העץ ותרד עד לעלה הכי ימני של העץ. סיבוכיות זמן ריצת הפונקציה במקרה זה הינה O(log(n)).

```
public void arrayToTree(int[] aa)
```

הפונקציה מקבלת כפרמטר מערך של מספרים שלמים הממוין מהקטן אל הגדול. הפונקציה יוצרת עץ אדום שחור חדש עם הערכים מהמערך, משנה את המצביע לשורש כך שיצביע לשורש העץ החדש (O(1)) ומעדכנת את המינימום והמקסימום.
הפונקצייה קוראת לפונקציית עזר makeTree שבונה את העץ ממערך ומעדכנת את המינימום והמקסימום בסיבוכיות של O(n).
הסיבוכיות הכוללת של הפונקציה הינה O(n).

```
private RBNode makeTree(int[] array,int start,int end,int blackHigh)
```

הפונקציה מקבלת מערך של מספרים שלמים הממוין בסדר עולה והופכת אותו לעץ אדום שחור חוקי.

בכל שלב הפונציה לוקחת את האיבר האמצעי במערך והופכת אותו לשורש. מהחצי הראשון היא בונה בצורה רקורסיבית את העץ שהוא הבן השמאלי של השורש, ומהחצי השני את העץ שהוא הבן הימני של השורש. את הצמתים ברמה התחתונה בעץ היא צובעת באדום אם הרמה לא מלאה. בנוסף, מעדכנת את המינימום והמקסימום. הפונקציה מתוארת ע"י נוסחאת הנסיגה:
$T(n)=2T(n/2)+1$
לכן הסיבוכיות של הפונקציה הינה $O(n)$.

```
private void rotate(boolean right,RBNode node)
```

הפונקציה מקבלת פרמטר מסוג RBNode המצביע על הצומת שנדרש לבצע לו סיבוב, הפרמטר השני של הפונקציה right הינו פרמטר בולאני המסמן את כיוון הסיבוב. כאשר ערכו "true" הסיבוב יתבצע ימינה וכאשר ערכו "false" הסיבוב הוא שמאלה. הפונקציה מזמנת את אחת הפונקציות עזר rotateRight או rotateLeft, בהתאם לערכו של הפרמטר הבולאני right. סיבוכיות כל אחת הפונקציות עזר היא $O(1)$. לכן הסיבוכיות הכוללת של פונקציית rotate הינה $O(1)$.

```
private void rotateRight(RBNode node)
```

הפונקציה מקבלת פרמטר מסוג RBNode המסמן את הצומת שעליה מפעילים את הסיבוב. הפונקציה מבצעת סיבוב ימינה תוך ביצוע מספר קבוע של פעולות.
לכן סיבוכיות הפונקציה הינה $O(1)$.

```
private void rotateLeft(RBNode node)
```

הפונקציה מקבלת פרמטר מסוג RBNode המסמן את הצומת שעליה מפעילים את הסיבוב. הפונקציה מבצע סיבוב שמאלה תוך ביצוע מספר קבוע של פעולות.
לכן סיבוכיות הפונקציה הינה $O(1)$.

```
private boolean isRed(RBNode node)
```

הפוקציה מחזירה אמת אם הצומת בצבע אדום, שקר אם הצומת בצבע שחור או שהצומת null. סיבוכיות הריצה היא $O(1)$.

מדידות:

מספר סידורי	מספר רוטציות ממוצע לפעולה בודדת, מקסימום לאורך סדרה	מספר צעדי תיקון מירבי לפעולה בודדת
1	0.71875	2
2	0.678571	2
3	0.875	2
5	0.6	2
6	0.666667	2
7	0.848485	2
8	0.666667	2
9	0.666667	2
10	0.666667	2

מכיוון שמספר הרוטציות המקסימלי בכל פעולת הכנסה הוא 2 (מקרים 2 ו-3 הם המקרים בהם יש רוטציות. מקרה 2 הוא סופי, ומקרה 3 ממשיך רק למקרה 2), לא יכול להיות מקסימום גבוה מזה. מכיוון שהכנסנו הרבה אברים, סביר מאוד שיהיה מקרה בו יצטרכו את שני המקרים על מנת לתקן את העץ. לכן ציפינו לתוצאה דומה.
מסיבה זו, ברור שהממוצע המקסימלי צריך להיות נמוך מ- 2 (בהכנסה הראשונה בטוח לא מתבצעת רוטציה), וגם סביר שיהיה נמוך יותר, מכיוון שבמקרה 1 לא מבוצעות רוטציות ושתי רוטציות נדרשות רק במקרה 3. ציפינו לתוצאה קרובה ל-1, כשהתוצאות בפועל יצאו נמוכות יותר (המקסימום נמוך מ-0.9).

הממוצע מראה שכמות הרוטציות לפעולה חסום ע"י $O(1)$. עובדה זו לא מפתיעה מכיוון שמספר הרוטציות המקסימלי לפעולה הוא 2. לא ניתן להסיק על חסם תחתון מכיוון שמדובר על ממוצע מקסימלי (שתאורתית יכול להתקבל ב-10 ההכנסות הראשונות ולאחר מכן הוא יורד משמעותית). בנוסף, ניתן להסיק שברוב הסדרות לא בוצעו רוטציות לפחות בשליש מההכנסות, ובאחרות לא בוצעו רוטציות לפחות בשמינית מהם. (אם הממוצע המקסימלי היה X, ובכל פעולה היתה מקסימום רוטציה אחת, לפחות ב- $n(1-x)$ מהמקרים לא בוצעו רוטציות. בפועל לא ניתן לדעת מנתונים אלו בדיוק בכמה מקרים בוצעו רוטציות- מכיוון שבמקרה 3 מבוצעות 2 רוטציות להכנסה בודדת). לכן, מהמדידות ניתן להסיק חסם עליון על מספר הרוטציות בפעולת הכנסה בודדת (טריוואלי) וחסם עליון הדוק יותר (בקבוע) על סדרה של פעולות הכנסה.

המשמעות של לקחת מקסימום של הממוצע לאורך סדרת הכנסות היא בעצם לעשות amortized למספר פעולות הרוטציה בכל הכנסה. מספר הרוטציות לפעולה בסדרה הכי גרועה שנתקלנו בה במהלך המדידות הוא הממוצע המקסימלי בכל המדידות. לכן ניתן להסיק שמספר הרוטציות amortized לפעולה הוא בערך 0.8, כלומר נמוך מ-1.