ספיר בן דוד 316224104
אורי ניסנקורן 207381948

פרויקט מס' 1 – עץ דרגות

$AVLTree$:

מחלקה זו מייצגת עץ דרגות AVL.

שדות המחלקה:

- $root$ שם נשמר שורש העץ.
- $Size$ – שומר את גודל העץ
- $minNode$- מצביע לאיבר עם המפתח המינימלי בעץ.
- $maxNode$- מצביע לאיבר עם המפתח המקסימלי בעץ.

פונקציות במחלקה:

1. $\boldsymbol{Empty() - O(1)}$: קריאה לפונקציות $getRoot()$ ו $getValue()$ אשר שתיהן פועלות בסיבוכיות זמן $O(1)$. השוואה והחזרת ערכים $O(1)$.
2. $\boldsymbol{Search(int\ k) - O(logn)}$: במקרה הגרוע המפתח לא נמצא בעץ ולכן נעבור על כל גובה העץ, אשר הוא לכל היותר $O(logn)$.
3. $\boldsymbol{Insert(int\ k, Boolean\ s) - O(logn)}$: נפריד את הניתוח לחלקים:

- מציאת המקום המתאים להכנסה - $O(logn)$, נצטרך לרדת עד לעלה המתאים ולכן נעבור את כל גובה העץ שהוא $O(logn)$, אם קיימת כבר צומת מתאימה נעצור פה ואחרת נמשיך לשלב הבא.
- עדכון שדות העץ והמפתח, הוספת צומת חדשת ועדכון מינימום ומקסימום - כל פעולות אלו משתמשות בגישה לשדות של צמתים והשוואות ולכן פונקציות אלו פועלות בסיבוכיות זמן $O(1)$.
- איזון העץ – כפי שנלמד בהרצאות יש לכל היותר גלגול אחד בעץ.

- נצטרך לעבור על המסלול מהצומת שהוכנס עד השורש או עד שנגיע לצומת שלא דורשת איזון ולבדוק אם קיימת צומת במסלול זה שדורשת איזון. במקרה הגרוע צריך לאזן את השורש, ולכן נבצע $O(logn)$ איטרציות.
- הפונקציות $rightRotate()$ ו$leftRotate()$ משתמשות בגישה לשדות של צמתים ועידכונם. פעולות אלו מתבצעות בסיבוכיות זמן קבועה, ולכן הסיבוכיות של הפונקציות הנ"ל היא $O(1)$.
- הפעלת הפונקציה $rotate()$ מבצעת סיבוב יחיד ומפעילה את $rightRotate()$ או $leftRotate()$ ולכן הפעלת עולה $O(1)$.

כל הפעולות הנ"ל מתבצעות בטור אחת אחרי השנייה ולכן לאחר ניסכום את הסיבוכיות ונקבל שסיבוכיות הפונקציה היא $O(logn)$.

4. $\boldsymbol{Delete(int\ k) - O(logn)}$: נפריד את הניתוח לחלקים:
   - מציאת צומת למחיקה- במקרה הגרוע נעבור על כל גובה העץ ולכן $O(logn)$.
   - עדכון שדות העץ וצמתים, עדכון מינמום/מקסימום – משתמשי בגישה לשדות של הצמתים או העץ ולכן פועלות בסיבוכיות זמן קבועה.
   - מחיקת הצומת – שינוי מצביעים של צמתים בעץ ולכן פועל בסיבוכיות זמן קבועה.
   - איזון העץ – כמו בהכנסה אנו עולים כל פעם לאב של הצומת ומבצעים פעולת איזון עליו עד שמגיעים לשורש. במקרה הגרוע יכולים להיות $O(logn)$ פעולות איזון כשכל פעולה עולה $O(1)$. נקבל בסה"כ העלות לאיזון העץ היא $O(logn)$.

   כל הפעולות הנ"ל מתבצעות בטור בזה אחרי זה. ולכן לאחר סכימת הסיבוכיות נקבל שסיבוכיות הפונקציה היא $O(logn)$.
5. $\boldsymbol{Min() - O(1)}$: המימוש מחזיק מצביע לצומת עם המפתח המינימלי. לכן הפעלת הפונקציה $empty$ וגישה לערך של הצומת עם המפתח המינימלי דורש $O(1)$.
6. $\boldsymbol{Max() - O(1)}$: המימוש מחזיק מצביע לצומת עם המפתח המקסימלי. לכן הפעלת הפונקציה $empty$ וגישה לערך של הצומת עם המפתח המקסימלי דורש $O(1)$.
7. $\boldsymbol{keysToArray() - O(n)}$ **:**
   - $in\_order\_keys\_array(int\ []\ arr, AVLNode\ node, int\ []index)$

     סקירת $inorder$ של העץ אשר בכל צומת יש $O(1)$ עבודה עבור גישה למפתח הצומת השמתו במערך והגדלת האינדקס. לכן בסה"כ סיבוכיות הזמן של פונקציה זו $O(n)$.
   - הפעלת $empty$ עולה $O(1)$, איתחול מערך בגודל n עולה $O(n)$ והפעלת הפונקציה $in\_order\_keys\_array$ עולה $O(n)$.
   - הפעולות הנ"ל מתבצעות בתור ולכן נקבל שהסיבוכיות היא $O(n)$.
8. $\boldsymbol{infoToArray() - O(n)}$: באופן דומה לניתוח של הפונקציה הקודמת:
   - $in\_order\_value\_array(boolean\ []\ arr, AVLNode\ node, int\ []index)$

     סקירת $inorder$ של העץ אשר בכל צומת יש $O(1)$ עבודה עבור גישה למפתח

הצומת השמתו במערך והגדלת האינדקס. לכן בסה"כ סיבוכיות הזמן של פונקציה זו $O(n)$.

- הפעלת $empty$ עולה $O(1)$, איתחול מערך בגודל n עולה $O(n)$ והפעלת הפונקציה $in\_order\_values\_array$ עולה $O(n)$.
- הפעולות הנ"ל מתבצעות בתור ולכן נקבל שהסיבוכיות היא $O(n)$.

9. $\boldsymbol{size() - O(1)}$: גישה לשדה של העץ.

10. $\boldsymbol{getRoot() - O(1)}$: קריאה לפונקציה $getson()$ של המחלקה $AVLRoot$ עולה $O(1)$ סיבוכיות זמן מפני שמצבעת גישה לשדות.

11. $\boldsymbol{successor(AVLNode\ node) - O(logn)}$, נפצל לשני מקרים:
    - אם ל$node$ בן ימני : נבצע צעד ימינה ואז שמאלה עד לשורש
    - אם ל$node$ אין בן ימני: נעלה למעלה עד צעד ראשון ימינה

במקרה הגרוע נצטרך לעלות את גובה העץ. כמו כן גישה לשדות של צמתים והשוואות עולות $O(1)$, ולכן בסה"כ הסיבוכיות תהייה $O(logn)$.

12. $\boldsymbol{prefixXor(int\ k) - O(logn)}$:
    - הפונקציה $FindNodeByKey(int\ key)$: פונקציה זו מקבלת מפתח ומחזירה את הצומת שkey המפתח שלו. לפי הנתונים קיים צומת שמפתחו key, ולכן במקרה הגרוע צומת זה הוא עלה, ולכן במקרה זה הפונקציה תעבור על כל הגובה של העץ שהוא לכל היותר $O(logn)$.
    - הפעלת הפונקציה $FindNodeByKey$ וגישה לשדה של צומת יעלו בסה"כ ביחד $O(logn)$ כנדרש.

13. $\boldsymbol{succPrefixXor(int\ k) - O(klogn)}$: במקרה הגרוע k הינו המפתח של הצומת המקסימלי בעץ, ולכן נצטרך לבצע לכל היותר k פעולות $successor$, אשר כל פעולה עולה במקרה הגרוע $O(logn)$. כמו כן שאר הפעולות מתבצעות בסיבוכיות קבועה, ולכן בסה"כ נקבל שהסיבוכיות של פונקציה זו היא $O(klogn)$.

ספיר בן דוד 316224104
אורי ניסנקורן 207381948

AVLNode

מחלקה זו מייצגת צומת בעץ דרגות AVL.

שדות המחלקה:

- $info$ – שומר את ערכו של הצומת
- Key- שומר את המפתח של הצומת.
- $left\_son$- שומר מצביע לצומת המוגדר להיות בן שמאלי של הצומת בעץ.
- $right\_son$- שומר מצביע לצומת המוגדר להיות בן ימני של הצומת בעץ.
- $parent$- שומר מצביע לצומת שמוגדר להיות ההורה של הצומת בעץ.
- $height$- שומר את גובה הצומת בעץ.
- Sub_tree_xor - שומר את ה- XOR של הערכים הבוליאניים של הצמתים הנמצאים בעץ עם מפתח קטן או שווה לצומת עצמו. ---- לשנות שם שדה ! ----

פונקציות במחלקה:

במחלקה זו הפונקציות מבצעות שינויים בשדות הצומת והשוואות ולכן סיבוכיות הזמן שלהן $O(1)$ כנדרש.

AVLRoot

שדות המחלקה:

- son- מחזיר את הבן של הצומת, זה יהיה השורש האמיתי של העץ. המחלקה עושה העמסה לכל הפעולות left ו-right עבור הצומת ומחזירה את אותו בן עבור שניהם.

מחלקה היורשת מהמחלקה $AVLNode$ ומשמשת להגדרת שורש העץ ותפעולו בפעולות השונות. כמו כן כל הפעולות הנוספות במחלקה זו עובדות בסיבוכיות זמן $O(1)$.

בכל עץ יש עצם אחד מסוג זה ותפקידו הוא לשמש כאב וירטואלי לשורש האמיתי של העץ, לצורך אחידות ופשטות ביישום פעולות הרוטציה והמחיקה של העץ. בגלל שהשורש האמיתי מוגדר כבן של השורש הווירטואלי לא נצטרך להחריג בפעולות את צורת הטיפול בצומת במידה והיא שורש.

ספיר בן דוד 316224104
אורי ניסנקורן 207381948

חלק ב' – בדיקות

הערות כלליות לשתי הבדיקות:

- הזמנים בננו שניות
- נייחס הבדל בזמני הריצה כהבדל בסדר גודל, כדי להתעלם מהבדלים טכניים התלויים במחשב ובהגרלת המספרים.

בדיקה 1

	כל הקריאות		100 קריאות ראשונות	
עלות ממוצעת / מס' בדיקה	$prefixXor$	$succPrefixXor$	$prefixXor$	$succPrefixXor$
1	487.4	3639	865	1111
2	219.3	6896.9	208	762
3	296	9496.4667	158	505
4	184.45	16142	166	7291
5	1583.64	23230.76	5234	17103

מהמדידות ניתן לראות שכשהצמתים שאנו מבצעים עליהם את הפעולה $succPrefixXor$ רחוקים יותר מהמינימום הפעולה לוקחת יותר זמן. תכונה זו ניכרת מהשוואה בין ממוצע 100 הצמתים הקטנים לכל הקריאות.

ב$prefixXor$ לעומת זאת, סיבוכיות הפעולה דומה בכל מקום בעץ בכל בדיקה, כלומר

באופן כללי ניתן לראות שככל שמגדילים את מספר הצמתים בעץ הזמן בשתי הפונקציות עולה כמצופה ובנוסף ניתן לראות כי $prefixXor$ יותר יעיל מ- $succPrefixXor$ גם כמצופה.

ספיר בן דוד 316224104
אורי ניסנקורן 207381948

בדיקה 2

	סדרה חשבונית		סדרה מאוזנת		סדרה אקראית	
סוג עץ / מס' בדיקה	**עץ AVL**	**עץ ללא מנגנון איזון**	**עץ AVL**	**עץ ללא מנגנון איזון**	**עץ AVL**	**עץ ללא מנגנון איזון**
1	0.3	0.2	0.1	0.2	0.2	0.1
2	0.1	0.15	0.15	0.1	0.15	0.15
3	0.1	0.06667	0.1	0.06667	0.1	0.0667
4	0.15	0.075	0.05	0.05	0.05	0.05
5	0.14	0.1	0.06	0.08	0.04	0.04

- סדרה חשבונית:

במקרה זה ציפינו שההכנסה לעץ לא מאוזן תהיה מהירה יותר כי בעץ AVL נבצע מספר רב של גלגולים וזאת למרות שעומק העץ המאוזן קטן יותר.

ואכן נראה שזמן פעולות האיזון גדולה יותר מזמן ההכנסה לעץ לא מאוזן.

- סדרה מאוזנת:

נצפה ששני העצים יתנהגו באופן דומה, מכיוון שבמקרה זה נצפה שלא יהיו כמעט גלגולים ולכן ההכנסה לשני העצים תהייה כמעט זהה. כתוצאה מכך נצפה שזמני הריצה יהיו דומים.

ואכן זה המצב.

- סדרה אקראית:

נצפה שהסדרה תהיה יותר מאוזנת מחשבונית, ולכן העצים יתנהגו באופן דומה אחד לשני, ובדומה למקרה של הסדרה המאוזנת. ואכן זה קרה.

מהתוצאות שלנו רואים שזמן ההכנסה הממוצע יורד ככל שגודל העץ גדל, בניגוד לציפיות שלנו. אנו משערים שזה קורה כתוצאה מקיצורי דרך שעושה הcompiler המשפיעים יותר על ההכנסות מאוחרות לעומת הכנסות מוקדמות.