<u>תיעוד פרוייקט מעשי 1 בקורס מבני נתונים - מימוש עץ AVL</u>

לירון כהן, lironcohen3, 207481268
יובל מור, yuvalmor, 209011543

<u>המחלקה AVLTree</u>

המחלקה מממשת עץ AVL, ובה שדה יחיד מסוג IAVLNode המצביע על שורש העץ.

כל אחד מצמתי העץ מממש מופע ממחלקת AVLNode, אודותיה נפרט בהמשך.

<u>מתודות AVLTree</u>

<u>AVLTree()</u>

המתודה בונה עץ ריק, כלומר מאתחלת את שורש העץ להיות null.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערכים של שדה.

<u>Empty()</u>

המתודה בודקת האם שורש העץ הוא null. אם כן, מחזירה אמת ואחרת שקר.

סיבוכיות המתודה היא $O(1)$ כיוון שהשוואת שורש העץ ל-null נעשית בזמן קבוע.

<u>Size()</u>

המתודה מחזירה את ערך השדה size של שורש העץ. אם השורש לא קיים, מחזירה 0.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

<u>getRoot()</u>

המתודה מחזירה את ערך השדה root של העץ.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

<u>getRank()</u>

המתודה מחזירה 1- אם העץ ריק. אחרת, מחזירה את גובה שורש העץ (שבעץ AVL הוא גם דרגת השורש ולכן גם דרגת העץ כולו).

סיבוכיות המתודה היא $O(1)$ כיוון שהשוואת שורש העץ ל-null ואחזור ערך שדה נעשות בזמן קבוע.

<u>Min()</u>

המתודה מתחילה עם מצביע לשורש העץ ויורדת עם המצביע לבן השמאלי כל עוד הבן השמאלי קיים (והוא לא עלה וירטואלי, כדי להגיע לעלה האמיתי הנדרש). המתודה מחזירה את העלה האמיתי הכי שמאלי בעץ.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שמדובר בעץ AVL (שהינו עץ חיפוש בינארי מאוזן), עומק העץ הוא $O(\log(n))$ וכדי להגיע לעלה השמאלי ביותר בעץ (בעל הערך המינימלי) נצטרך לרדת בלולאה $O(\log(n))$ פעמים.

Max()

המתודה מתחילה עם מצביע לשורש העץ ויורדת עם המצביע לבן הימני כל עוד הבן הימני קיים (והוא לא עלה וירטואלי, כדי להגיע לעלה האמיתי הנדרש). המתודה מחזירה את העלה האמיתי הכי ימני בעץ.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שמדובר בעץ AVL (שהינו עץ חיפוש בינארי מאוזן), עומק העץ הוא $O(\log(n))$ וכדי להגיע לעלה הימני ביותר בעץ (בעל הערך המקסימלי) נצטרך לרדת בלולאה $O(\log(n))$ פעמים.

Search(int k)

המתודה מקבלת מספר שלם המייצג מפתח בעץ ומבצעת חיפוש בינארי על צמתי העץ.

בכל שלב, המתודה משווה את מפתח הצומת הנבדק לעומת המפתח שהתקבל כארגומנט.

נתחיל ממצביע לשורש, ונמשיך בחיפוש כל עוד לא הגענו לצומת שהיא null:

אם המפתחות שווים, מחזירה את ערך הצומת.

אם המפתח שהתקבל קטן ממפתח הצומת, מצביע הצומת יעבור לבן השמאלי.

אם המפתח שהתקבל גדול ממפתח הצומת, מצביע הצומת יעבור לבן הימני.

אם לא נמצא המפתח עד לשלב זה, המפתח אינו נמצא בעץ ומוחזר null.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שמדובר בעץ AVL (שהינו עץ חיפוש בינארי מאוזן), עומק העץ הוא $O(\log(n))$ ובמקרה הגרוע בכל איטרציה של לולאת ה-while לא מוצאים את המפתח ולכן יורדים רמה בעץ, עד להגעה לעלה וירטואלי.

treePosition(int k) – פעולת עזר

המתודה מקבלת מפתח להכנסה ומחזירה את צומת ההורה של המיקום המתאים להכנסה.

אם המפתח קיים כבר בעץ, מחזירה את הצומת עם המפתח שהתקבל.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שמדובר בעץ AVL (שהינו עץ חיפוש בינארי מאוזן), עומק העץ הוא $O(\log(n))$ ובמקרה הגרוע בכל איטרציה של לולאת ה-while לא מוצאים את המפתח ולכן יורדים רמה בעץ, עד להגעה לעלה וירטואלי.

insertBST(IAVLNode) – מתודת עזר

המתודה מקבלת צומת ומכניסה אותו למקום המתאים בעץ לפי הגדרות עץ חיפוש בינארי.

המתודה משתמשת במתודת העזר treePosition שסיבוכיותה $O(\log(n))$.

לאחר מכן, המתודה בודקת האם מפתח הצומת להכנסה זהה למפתח שהתקבל מהפעולה, אם כן המתודה מחזירה 1- כיוון שהמפתח קיים בעץ.

אם לא, בודקת האם להכניס את הצומת כבן שמאלי או בן ימני, מעדכנת את ה-rank של הצומת החדש ולבסוף מחזירה 1.

מלבד פעולת העזר מתבצעות בדיקות תנאים ועדכוני שדות ולכן סיבוכיות המתודה היא $O(\log(n))$.

rankDiff(AVLNode p, AVLNode c) – מתודת עזר

המתודה מקבלת הורה ובן ומחזירה את הפרש שדות ה-rank ביניהם.

סיבוכיות המתודה היא $O(1)$ כיוון שמתבצעת פעולה אריתמטית והחזרת שדות.

updateSize(AVLNode n) – מתודת עזר

המתודה מקבלת צומת בעץ, מעדכנת את שדה ה-size שלו מתוך ערכי השדות של שני בניו (כפי שראינו בתרגול, נעשה ב-$O(1)$). המתודה עושה זאת עבור כל הורי הצומת עד להגעה לשורש.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שבמקרה הגרוע הצומת שהתקבל הוא עלה ולכן כמות האיטרציות תהיה גובה העץ, שהוא $O(\log(n))$.

Promote(AVLNode n), Demote(AVLNode n) – מתודות עזר

המתודות מקבלות צומת ומוסיפות/מורידות 1 מה-rank שלו.
סיבוכיות המתודות היא $O(1)$ כיוון שמתבצעת פעולה אריתמטית, עדכון והחזרת שדות.

leftRotate(AVLNode z, AVLNode n) ,rightRotate(AVLNode z, AVLNode n) – מתודות עזר

המתודות מקבלות הורה ובן (ביניהם הקשת שעלינו לסובב), מעדכנות את המצביעים הרלוונטיים בהתאם לצעדים שראינו בהרצאה וכן קוראות למתודת העזר UpdateSize על מנת לעדכן את שדות ה-size של הצמתים ששונו בפעולה. המתודות מחזירות 1 (עבור פעולת איזון אחת).

סיבוכיות המתודות היא $O(\log(n))$ כיוון שסיבוכיות מתודת העזר היא $O(\log(n))$ ומלבדה מתבצעים עדכוני מצביעים ועדכוני שדות.

leftRightRotate(AVLNode z, AVLNode n) ,rightLeftRotate(AVLNode z, AVLNode n) – מתודות עזר

המתודות מקבלות הורה ובן (ביניהם הקשת הראשונה שעלינו לסובב, כלומר הקשת הנמוכה יותר מבין שתי הקשתות), קוראות לפעולות הסיבוב המתאימות ומחזירות 2 (עבור שני הסיבובים).

סיבוכיות המתודות היא $O(\log(n))$ כיוון שמתבצעות קריאות לפעולות שסיבוכיותן $O(\log(n))$.

RebalanceInsert(AVLNode n) – מתודת עזר

המתודה מקבלת הורה של צומת שהוכנס ומאזנת מחדש את העץ בהתאם למצב הצומת (אל מול הבנים שלו). המתודה מחלקת למקרים לפי המקרים שנראו בהרצאה וקראת לפעולות האיזון המתאימות (promote ו-rotations) אם יש צורך.

המתודה מחזירה את מספר פעולות האיזון שנעשו.

סיבוכיות המתודה במקרה הגרוע היא $O(\log(n))$. מלבד כאשר יש צורך ב-promote, כל הפעולות נעשות ב-$O(1)$. אם התבצע promote, ישנה אפשרות שבעיית האיזון "עלתה למעלה" רמה אחת בעץ ולכן התבצעה קריאה נוספת ל-rebalanceInsert. ראינו בכיתה שמספר פעולות ה-promote הוא לכל היותר גובה העץ, אשר כמעט מאוזן ולכן מדובר ב-$O(\log(n))$ בסך הכל.

Insert(int k, String s)

המתודה מקבלת מפתח וערך ויוצרת צומת חדש בהתאמה.

אם העץ ריק, מכניסה את הצומת כשורש העץ ומחזירה 0 כיוון שלא בוצעו פעולות איזון.
אם העץ לא ריק, היא מכניסה את הצומת לעץ לפי חוקי עץ חיפוש בינארי (באמצעות מתודת העזר insertBST). אם המפתח היה בעץ, מחזירה 1-. אחרת:

- מעדכנת את ערכי השדה size של הצמתים הרלוונטיים (באמצעות מתודת העזר updateSize)
- מאזנת מחדש את העץ (באמצעות מתודת העזר rebalanceInsert)
- מחזירה את מספר פעולות האיזון שנעשו.

סיבוכיות המתודה היא $O(\log(n))$ כיוון ששלוש פעולות העזר נקראות באופן טורי והן בסיבוכיות של $O(\log(n))$. מלבד זאת מתבצעת יצירת צומת ובדיקות פשוטות שהן $O(1)$.

Successor(AVLNode n) – מתודת עזר

המתודה מקבלת צומת בעץ ומחזירה את הצומת העוקב שלו בהתאם לנלמד בהרצאה.

סיבוכיות המתודה היא $O(\log(n))$ כיוון שבמקרה הגרוע המתודה צריכה "לעלות" או "לרדת" את כל העץ כדי למצוא את הצומת העוקב, ומכיוון שמדובר בעץ AVL כמעט מאוזן, מדובר ב-$O(\log(n))$ רמות.

deleteBST(AVLNode) – מתודת עזר

המתודה מקבלת צומת ומוחקת אותו מהמקום המתאים בעץ לפי הגדרות עץ חיפוש בינארי. המתודה מחזירה את ההורה של הצומת שנמחק (במקרה שבו לצומת היו שני ילדים, מחזירה את ההורה של הצומת העוקב, שבמיקומו התבצעה המחיקה בפועל).

אם צריך, המתודה משתמשת במתודה successor שסיבוכיותה $O(\log(n))$. בכל מקרה אחר המתודה משנה את המצביעים הרלוונטיים למחיקה ומקטינה את גודל העץ ב-1.

מלבד הקריאה לפעולת העזר מתבצעות בדיקות תנאים ועדכוני שדות ולכן בסה"כ סיבוכיות המתודה היא $O(\log(n))$.

RebalanceDelete(AVLNode n) – מתודת עזר

המתודה מקבלת את ההורה של הצומת שנמחק ומאזנת מחדש את העץ בהתאם למצב הצומת (אל מול ההורה והבנים שלו). המתודה קוראת לפעולות ה-demote, promote וה-rotations לפי המקרים שנראו בכיתה ומחזירה את מספר פעולות האיזון שנעשו.

סיבוכיות המתודה במקרה הגרוע היא $O(\log(n))$. פעולות איזון טרמינליות נעשות בקריאה למתודות עזר שסיבוכיותן $O(1)$. אם בעיית האיזון "עלתה למעלה" רמה אחת בעץ התבצעה קריאה נוספת ל-rebalanceDelete, מספר ה"עליות" בעץ הוא לכל היותר גובה העץ, אשר כמעט מאוזן ולכן מדובר ב-$O(\log(n))$ בסך הכל.

Delete(int k)

המתודה מקבלת מפתח למחיקה ומוצאת באמצעות (מתודת העזר treePosition) את מיקום הצומת למחיקה. אם המפתח לא בעץ, מחזירה 1-. אחרת, מוחקת את הצומת מהעץ לפי חוקי עץ חיפוש בינארי (באמצעות מתודת העזר deleteBST).

לאחר מכן, המתודה מאזנת מחדש את העץ (באמצעות שליחת ההורה למתודת העזר rebalanceDelete), מעדכנת את ערכי השדה size של הצמתים הרלוונטיים (באמצעות מתודת העזר updateSize) ומחזירה את מספר פעולות האיזון שנעשו.

נציין שאם הצומת למחיקה היא שורש העץ, המתודה מוחקת את השורש מהעץ כרגיל אך שולחת ל-rebalanceDelete את השורש ולא את ההורה שלו (שהוא null) כדי לאזן את שורש העץ אם צריך.

סיבוכיות המתודה היא $O(\log(n))$ כיוון שכל פעולות העזר נקראות באופן טורי והן בסיבוכיות של $O(\log(n))$. מלבד זאת מתבצעות בדיקות פשוטות שהן $O(1)$.

nodesToArray() – מתודת עזר

המתודה מחזירה מערך צמתים המכיל את צמתי העץ מסודרים לפי מפתחות בסדר עולה.

המתודה מגדירה מחסנית ששומרת את כל הצמתים שהמצביע עבר בהם.

כל עוד המצביע לא הגיע לעלה וירטואלי או שיש עוד צמתים במחסנית, נכניס את הצומת הנוכחי למחסנית ונזוז לבן השמאלי. כשנגיע לעלה וירטואלי נכניס למערך את הצומת העליון במחסנית (המינימום באותו הרגע) ונזוז לבן הימני של הצומת שהכנסנו למערך (בדומה למעבר in-order שראינו בכיתה).

סיבוכיות המתודה היא $O(n)$ מכיוון שנבקר בכל צומת לכל היותר 3 פעמים (בדרך לבן השמאלי, בחזרה מהבן השמאלי אל הימני ובחזרה מהימני להורה), לכן נקבל $O(n)$ ביקורים. פעולות המחסנית, כמו גם בדיקות והחזרת שדות נעשות ב-$O(1)$ ולכן נקבל סיבוכיות לינארית.

KeysToArray()

המתודה מחזירה מערך שלמים שמכיל את מפתחות העץ בסדר עולה.

המתודה משתמשת במתודת העזר nodesToArray ומקבלת ממנה מערך צמתים מסודר לפי מפתחות בסדר עולה. המתודה עוברת על מערך הצמתים ומכניסה למערך שלמים את מפתחות הצמתים לפי הסדר.

סיבוכיות המתודה היא $O(n)$ מכיוון שמתודת העזר פועלת ב-$O(n)$ ומעבר על המערך נעשה גם כן ב-$O(n)$. שאר הפעולות נעשות בזמן קבוע.

infoToArray()

המתודה מחזירה מערך מחרוזות שמכיל את ערכי העץ מסודרים לפי מפתחות בסדר עולה.

המתודה משתמשת במתודת העזר nodesToArray ומקבלת ממנה מערך צמתים מסודר לפי מפתחות בסדר עולה. המתודה עוברת על מערך הצמתים ומכניסה למערך מחרוזות את ערכי הצמתים לפי הסדר.

סיבוכיות המתודה היא $O(n)$ מכיוון שמתודת העזר פועלת ב-$O(n)$ ומעבר על המערך נעשה גם כן ב-$O(n)$. שאר הפעולות נעשות בזמן קבוע.

Clone(AVLNode n) – מתודת עזר

המתודה מקבלת צומת ומחזירה צומת חדש משוכפל, המכיל את המפתח, הערך ודרגה של הצומת ללא מצביעים להורה ובנים.
סיבוכיות המתודה היא $O(1)$ כיוון שמדובר ביצירת צומת חדש ועדכון שדות.

Split(int x)

המתודה מקבלת מספר המייצג מפתח, ובהתאם לאלגוריתם שהצגנו בכיתה מפצלת את העץ הנוכחי לשני עצי AVL, כך שבעץ אחד נמצאים כל הצמתים בעלי המפתחות הקטנים מהמפתח ובעץ השני נמצאים כל הצמתים בעלי המפתחות הגדולים מהמפתח. המתודה מחזירה את שני העצים בתוך מערך עצים.

ראשית המתודה מאתחלת שלושה עצי AVL –

T1 שבו יהיו הצמתים עם המפתחות הקטנים, T2 שבו יהיו הצמתים עם המפתחות הגדולים ועץ temp נוסף בו נשתמש בתוך המתודה בלבד.

המתודה מוצאת את הצומת עם המפתח הנתון כארגומנט באמצעות קריאה למתודת העזר treePosition, ומאתחלת את שני העצים לפי תת העץ השמאלי והימני של הצומת בהתאמה (אם לצומת קיימים בנים אמיתיים).

לאחר מכן, המתודה עוברת עם מצביע המתחיל מהצומת שמצאנו קודם, ובהתאם להאם הוא בן שמאלי או ימני של ההורה שלו, מכניסה את ההורה ואת תת העץ השני של ההורה לעץ המתאים (באמצעות המתודה join ותוך קריאה למתודת העזר clone עבור הצומת הבודד שנשלח ל-join). המתודה מעלה את המצביע להורה שלו, עד להגעה לשורש כלומר לפיצול העץ כולו.

לבסוף המתודה מאזנת באמצעות rebalanceInsert את שני העצים שהתקבלו ומחזירה אותם כמערך.

כפי שראינו בהרצאה, סיבוכיות המתודה היא $O(\log(n))$ (המתקבל כחסם הדוק כאשר לוקחים בחשבון שכל קריאה לפעולת join היא בסיבוכיות של $O(height\ difference+1)$ ולא $O(\log(n))$).

findRankEquiv(AVLTree tree, int rank, char d) – מתודת עזר

המתודה מקבלת עץ, מספר המייצג דרגה ותו המייצג כיוון (r עבור ימין ו-l עבור שמאל). המתודה מחזירה את הצומת על השדרה המתאימה (שמאלית או ימנית) של העץ שדרגתה היא המקסימלית שקטנה או שווה ל-rank (עבור המתודה join).

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שמדובר בעץ AVL (שהינו עץ חיפוש בינארי מאוזן), עומק העץ הוא $O(\log(n))$ ובמקרה הגרוע נצטרך לרדת עד לעלה השמאלי ביותר בעץ, כלומר הלולאה תרוץ $O(\log(n))$ פעמים.

Join(IAVLNode x, AVLtree t)

המתודה מקבלת עץ וצומת, ובהתאם לאלגוריתם שהצגנו בהרצאה מאחדת את העץ הנוכחי לעץ הנתון כארגומנט בהוספת הצומת הנתונה.

אם אחד העצים ריק או ששניהם ריקים, קוראת למתודה insert ומעדכנת את שורש העץ הנוכחי במידת הצורך.

אם שניהם לא ריקים, מבצעת השוואת מפתחות ודרגות כדי לקבוע את תצורת האיחוד (מי ימני או שמאלי ומי גבוה או נמוך), מוצאת בעץ בעל הדרגה הגבוהה את הצומת המקביל (מבחינת דרגה) לשורש העץ בעל הדרגה הנמוכה (באמצעות מתודת העזר findRankEquiv), משנה את המצביעים הרלוונטיים ומעדכנת את שורש העץ במידת הצורך.

לאחר מכן, המתודה קוראת למתודת העזר rebalanceInsert על מנת לאזן את העץ במידת הצורך וכן למתודת העזר updateSize על מנת לעדכן את ערכי השדות size של הצמתים הרלוונטיים.

המתודה מחזירה את הפרשי הגבהים של שני העצים + 1.

סיבוכיות המתודה היא $O(\log(n))$ מכיוון שמתודות העזר כולן בסיבוכיות $O(\log(n))$, הן פועלות בצורה טורית וביניהן מתרחשות בדיקות ושינויי מצביעים הנעשים ב-$O(1)$. נציין כי לפי מה שראינו בכיתה חסם הדוק יותר יהיה $O(height\ difference + 1)$.

המחלקה AVLNode

המחלקה מממשת צומת בעץ AVL לפי המנשק IAVLNode.

שדות AVLNode

- Int Key – המפתח של הצומת
- String info – הערך של הצומת
- IAVLNode parent – מצביע להורה של הצומת
- IAVLNode left – מצביע לבן השמאלי של הצומת
- IAVLNode right – מצביע לבן הימני של הצומת
- Boolean isReal – האם הצומת אמיתי או וירטואלי
- Int height – גובה / דרגת הצומת
- Int size – גודל תת העץ שתלוי מהצומת

מתודות AVLNode

AVLNode(int key, String info)

המתודה בונה עצם מסוג AVLNode, מאתחלת את המפתח והערך שלו למפתח והערך שהתקבלו כארגומנטים.

בנוסף, אם המפתח שהתקבל אינו 1- (כלומר צומת אמיתי), משנה את ערך ה-isReal לאמת (כיוון שבאופן דיפולטי משתנים בוליאניים מאותחלים לשקר), מגדירה את שני הבנים של הצומת לצמתים וירטואליים חדשים ואת ערך שדה ה-size שלו ל-1.

אם המפתח שהתקבל הוא 1-, מגדירה את דרגתו כ1- ואת שדה ה-size שלו ל-0.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מבצעת השמות ובדיקות שנעשות בזמן קבוע.

getKey()

המתודה מחזירה את המפתח של הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

getValue()

המתודה מחזירה את הערך של הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

setLeft(IAVLNode node)

המתודה מגדירה את שדה הבן השמאלי של הצומת לצומת שהתקבל כארגומנט.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערך של שדה.

getLeft()

המתודה מחזירה את צומת הבן השמאלי של הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

setRight(IAVLNode node)

המתודה מגדירה את שדה הבן הימני של הצומת לצומת שהתקבל כארגומנט.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערך של שדה.

getRight()

המתודה מחזירה את צומת הבן הימני של הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

setParent(IAVLNode node)

המתודה מגדירה את שדה ההורה של הצומת לצומת שהתקבל כארגומנט.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערך של שדה.

getParent()

המתודה מחזירה את צומת ההורה של הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

isRealNode()

המתודה מחזירה אמת אם הצומת אמיתי ושקר אם היא עלה וירטואלי.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

setHeight(int height)

המתודה מגדירה את שדה הגובה של הצומת לערך שהתקבל כארגומנט.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערך של שדה.

getHeight()

המתודה מחזירה את גובה הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

setSize(int size)

המתודה מגדירה את שדה ה-size של הצומת לערך שהתקבל כארגומנט.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא קובעת ערך של שדה.

getSize()

המתודה מחזירה את ערך שדה ה-size הצומת.

סיבוכיות המתודה היא $O(1)$ כיוון שהיא מחזירה ערך של שדה.

חלק המדידות

שאלה 1 סעיף א'

מספר סידורי	גודל המערך	כמות החילופים במיון רגיל עבור מערך אקראי	כמות החילופים במיון רגיל עבור מערך ממוין הפוך	עלות החיפושים במיון AVL עבור מערך אקראי	עלות החיפושים במיון AVL עבור מערך ממוין הפוך
1	10,000	25,063,646	49,995,000	221,782	231,358
2	20,000	100,023,653	199,990,000	480,520	502,688
3	30,000	224,839,071	449,985,000	777,310	788,113
4	40,000	401,139,380	799,980,000	1,056,505	1,085,346
5	50,000	625,021,906	1,249,975,000	1,334,497	1,386,195
6	60,000	900,795,981	1,799,970,000	1,614,641	1,696,195
7	70,000	1,225,062,394	2,449,965,000	1,986,235	2,010,660
8	80,000	1,599,255,957	3,199,960,000	2,233,915	2,330,660
9	90,000	2,032,584,245	4,049,955,000	2,522,302	2,650,660
10	100,000	2,504,360,436	4,999,950,000	2,885,898	2,972,357

<u>ציפיות ומסקנות מהטבלה</u>

<u>מיון רגיל - השוואה של מערך ממוין הפוך ומערך אקראי</u>

מבחינת קלטי המערכים האפשריים לאלגוריתם המיון, מערך ממוין בסדר הפוך הוא הגרוע יותר מבחינת כמות הפעולות. מערך זה הכי רחוק ממערך ממוין בסדר עולה ולכן נצפה לכמות חילופים גדולה מאוד ביחס למערך אקראי, אשר יכול להגריל סדר ערכים (באמצעות פעולת shuffle) היוצר מערך הקרוב יותר למערך הממוין הדרוש ולכן ידרוש מספר נמוך יותר של חילופים.

<u>מיון AVL – השוואה של מערך ממוין הפוך ומערך אקראי</u>

ראשית נדגיש שעקב אופן בניית המערכים, האיברים בעץ ה-AVL לאחר כל סבב הכנסות יהיו זהים (כלומר איברים מ-0 ועד גודל המערך פחות 1, אשר הוכנסו לעץ בסדר שונה). עקב העובדה שעץ AVL הוא עץ חיפוש, אלגוריתם ההכנסה שלו מסדר את העץ כך שישמור על כללי עץ חיפוש וכן מאזן את העץ. לכן, נצפה שלא יהיה הבדל משמעותי בין מערכים שונים שהוכנסו למיון העץ. נציין שעקב סדר ההכנסה במערך ממוין הפוך, ועקב העובדה שככל שמתקדמים במערך האיברים קטנים יותר וגם העץ גדל, נוכל לשער שעלות החיפושים תגדל מעט יותר בסדר הכנסה זה.

<u>השוואה של כמות החילופים במיון רגיל אל מול עלות החיפושים במיון AVL</u>

מכיוון שעץ AVL הוא עץ מאוזן, עלות חיפוש תהיה ב-$O(\log(n))$ (סדר גודל של גובה העץ), לעומת מיון רגיל שבכל איטרציה עובר על סדר גודל של כל המערך כדי לבצע את ההשוואות, כלומר $O(n^2)$. לכן, נצפה שעלות החיפושים במיון AVL תהיה קטנה משמעותית מכמות החילופים במיון רגיל.

<u>כמות החילופים כתלות בגודל המערך</u>

מכיוון שכמות החילופים במיון רגיל תלויה בגודל המערך וכן גם עלות החיפושים במיון AVL, נצפה שככל שנגדיל את גודל המערך כך יתבצעו יותר חילופים ועלות החיפושים תעלה.

ניתן לראות שהתוצאות שהתקבלו תואמות את הציפיות.

ניתוח כמות החילופים ב-Insertion Sort עבור מערך אקראי

מכיוון שמדובר במערך אקראי, נחשב את תוחלת כמות החילופים הממוצעת, $E[H]$ על פני כל המערכים האפשריים. נסמן את המערך ב-Arr.

עבור כל איטרציה של הלולאה באלגוריתם המיון, נגדיר עבור $j > i$ אינדיקטור $I_{i,j} = \begin{cases} 1 & are\ i,j\ swapped \\ 0 & \text{אחרת} \end{cases}$.

נשים לב שעבור $j > i$ מתקיים:

$$P(are\ i,j\ swapped) = P(Arr[i] > Arr[j]) = \frac{1}{2}$$

נקבל:

$$E[H] = E\left[\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} I_{i,j}\right] = \sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} E[I_{i,j}] = \sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1}\frac{1}{2} = \frac{1}{2}\sum_{i=0}^{n-1}(n-1-(i+1))$$
$$= \sum_{i=0}^{n-1} n+i-2 = n^2-2n+\sum_{i=0}^{n-1} i$$

מסכום סדרה חשבונית נקבל:

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

ובסך הכל נקבל:

$$E[H] = n^2-2n+\frac{n(n-1)}{2} = \frac{3n^2-5n}{2} = O(n^2)$$

מהגדרת אלגוריתם המיון, מכיוון שמבצעים איטרציה על כל איברי המערך ומבצעים H חילופים סך הכל, סיבוכיות האלגוריתם תהיה:

$$O(n+H) = E(n+H) = n+E[H] = n+O(n^2) = O(n^2)$$

ניתוח כמות החילופים ב-Insertion Sort עבור מערך ממוין הפוך

נשים לב שמדובר בקלט ספציפי ולכן נחשב באופן ישיר כמה חילופים מתבצעים עבורו.

נניח שהמערך בגודל n.

- עבור האיבר הראשון במערך, נבצע 0 חילופים.
- עבור האיבר השני במערך נשווה אותו לראשון, הם יהיו בסדר הפוך ולכן נבצע חילוף אחד.
- עבור האיבר השלישי הקטן מהשניים שלפניו, נבצע שני חילופים.
- באופן כללי, עבור האיבר ה-i נבצע $i-1$ חילופים.

נסכום את מספר החילופים ונקבל סדרה חשבונית :

$$\sum_{i=1}^{n} i-1=\sum_{i=0}^{n-1} i=\frac{n(n-1)}{2}=\frac{n^2-n}{2}$$

ולכן מספר החילופים עבור מערך זה הינו $\frac{n^2-n}{2}$ חילופים, כלומר $O(n^2)$ חילופים.

<u>ניתוח עלות החיפושים ב-AVL insertion sort עבור מערך אקראי</u>

ראשית ננתח את עלות החיפושים כתלות ב-n וב-H (כמות החילופים שהייתה מתבצעת אם המיון היה $insertion\ sort$ רגיל).

נסמן בתור h_i את כמות החילופים שנעשו לאיבר במקום ה-i במערך, כך שמתקיים :

$$H=\sum_{i=0}^{n-1} h_i$$

עבור חיפוש $finger\ search$ של איבר k במערך נעבור על log של מספר האיברים הנמצאים לפני האיבר במערך וגדולים ממנו (כלומר הוכנסו לפניו לעץ ונעבור דרכם בחיפוש), כלומר h_i קשתות. נסמן את כמות הקשתות הכוללת בתור $Cost$, ונקבל :

$$Cost \le \sum_{i=0}^{n-1} \log(h_i)=\log(h_0\cdot h_1\cdot \ldots \cdot h_{n-1}) \le \log\left(\left(\frac{h_0+h_1+\cdots+h_{n-1}}{n}\right)^n\right)$$

$$=\log\left(\left(\frac{H}{n}\right)^n\right)$$

$$Cost=O\left(n\cdot\log\left(\frac{H}{n}\right)\right)$$

כאשר אי השיוויון האחרון נובע ממניפולציה אלגברית על אי שיוויון הממוצעים והשיוויון האחרון נובע מחוקי לוגריתמים.

לבסוף נזכור שיש לבצע סריקת $in\ order$ של העץ ולכן נקבל $O\left(n+n\cdot\log\left(\frac{H}{n}\right)\right)$.

קיבלנו חסם כללי על עלות החיפושים הכוללת ולכן זהו גם חסם על התוחלת המבוקשת.

בבירור $h_i \le n$ (מכיוון שזוהי הכמות המקסימלית של ההחלפות עבור איבר במערך), ולכן :

$$H=h_0+h_1+\cdots+h_{n-1} \le n\cdot n=n^2$$

ומכאן נקבל $Cost=O\left(n\cdot\log\left(\frac{n^2}{n}\right)\right)=O(nlog(n))$.

ניתוח עלות החיפושים ב-AVL insertion sort עבור מערך ממוין הפוך

נשים לב שמדובר בקלט ספציפי ולכן נחשב באופן ישיר כמה חילופים מתבצעים עבורו.

ראינו בהרצאה שחיפוש איבר עם מפתח k בעץ $finger\ Search$ נעשה בסיבוכיות של $O(logk)$, ובמקרה הגרוע נקבל $2\log(k)$ קשתות לכל חיפוש.

מחוקי לוגריתמים נקבל:

$$\sum_{k=0}^{n-1} 2\log(k) = 2\sum_{k=0}^{n-1}\log(k) \le 2\sum_{k=0}^{n-1}\log(n-1) = nlog(n-1) = O(nlog(n))$$

שאלה 1 סעיף ב׳

כפי שהראינו בסעיף הקודם, החסם העליון על כמות החיפושים הוא $O\left(nlog\left(\frac{H}{n}\right)\right)$. מלבד עלות החיפושים, עבור חסם על הסיבוכיות נתחשב גם בפעולות איזון שמתבצעות על עץ ה-AVL.

במקרה הגרוע (עבור חסם עליון), נקבל שכל פעולת איזון חסומה ע״י $\log(n)$ (שמתקבלת עבור שרשרת $promote$ ובמימוש שלנו מתקבלת תמיד עקב פעולת $updateSize$). מתבצעת פעולת איזון עבור כל הכנסה, כלומר n פעולות איזון ולכן נקבל:

$$TotalCost = O\left(n\left(log\left(\frac{H}{n}\right) + \log(n)\right)\right) = O(nlog(H))$$

שאלה 2 סעיף א׳

מספר סידורי	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של איבר מקסימלי בתת העץ השמאלי	עלות join מקסימלי עבור split של איבר מקסימלי בתת העץ השמאלי
1	2.833	6	2.769	16
2	2.769	7	3.083	17
3	2.750	10	2.929	18
4	2.286	6	2.923	18
5	2.700	6	2.846	18
6	2.800	5	2.667	19
7	2.714	5	2.688	19
8	2.733	7	2.600	19
9	3.067	7	2.867	20
10	3.200	6	3.000	20

ניתוח עלות ממוצעת ל-join עבור split של האיבר המקסימלי בתת העץ השמאלי

נשים לב שאם ישנם n צמתים בעץ, גובה העץ יהיה כ-$\log(n)$. עלות $join$ ממוצע תהיה:

$$avg\ cost\ of\ join = \frac{\sum_{i=0}^{\log(n)} cost\ of\ the\ i_{th}\ join}{number\ of\ joins} = \frac{cost\ of\ split}{number\ of\ joins} = \frac{O(\log(n))}{O(\log(n))} = O(1)$$

כאשר מבצעים $split$ על האיבר המקסימלי בתת העץ השמאלי של העץ, כמות פעולות ה-$join$ שנבצע תהיה כגובה העץ כולו, שהוא $O(\log(n))$. בנוסף, ראינו בהרצאה שמחיר פעולת $split$ היא כסכום מחירי פעולות ה-$join$ המתבצעות בתוך פעולת ה-$split$ ומכאן נקבל את העלות המבוקשת.

ניתוח עלות ממוצעת ל-join עבור split אקראי

כפי שראינו, במקרה הגרוע (האיבר המקסימלי בתת העץ השמאלי) העלות הממוצעת תהיה $O(1)$ ובפרט חסם קבוע זה הוא גם חסם גם על התוחלת עבור מקרה אקראי.

ניתוח עלות מקסימלית ל-join עבור split אקראי

בהנחה שהעץ מאוזן, ראינו בתרגול שכמות הצמתים ברמה שבגובה i היא $n\frac{1}{2^{i+1}}$ (כלומר ברמת העלים יש $\frac{1}{2}$ מהצמתים, ברמה שמעליה יש $\frac{1}{4}$ מהצמתים וכו').

עבור $split$ לפי צומת ברמה ה-i, ה-$join$ המקסימלי יעלה כמספר המקסימלי של הקשתות שעולות "באותו הכיוון" (כלומר שרשרת של בנים ימניים או שרשרת של בנים שמאליים).
עלות פעולת ה-$join$ המקסימלי תהיה כגובה העץ פחות גובה הרמה, כלומר $\log(n) - i$.

נחשב לפי משפט התוחלת השלמה את תוחלת כמות פעולות ה-$join$ בחלוקה לפי רמות בהן ה-$split$ מבוצע:

$$\sum_{i=0}^{\log(n)} P(the\ node\ is\ in\ the\ i'th\ level) \cdot cost\ of max\ join\ from\ i'th\ level$$

$$= \sum_{i=0}^{\log(n)} \frac{1}{2^{i+1}} \cdot (\log(n) - i) = \log(n) \sum_{i=0}^{\log(n)} \frac{1}{2^{i+1}} - \sum_{i=0}^{\log(n)} \frac{i}{2^{i+1}} \leq \log(n) \sum_{i=0}^{\log(n)} \frac{1}{2^{i+1}}$$

$$\leq \log(n) \sum_{i=0}^{\infty} \frac{1}{2^{i+1}} = \log(n) \cdot \frac{\frac{1}{2}}{\frac{1}{2}} = \log(n) = O(\log(n))$$

ניתוח עלות מקסימלית ל-join עבור split של האיבר המקסימלי בתת העץ השמאלי

עקב המיקום הייחודי של האיבר המקסימלי בתת העץ השמאלי, נשים לב שה-$join$ המקסימלי יתקבל ב-$join$ האחרון.

בהתחלה יבוצעו $h-1$ (גובה העץ פחות 1) פעולות $join$ המחברות "מכיוון עלייה שמאלה" (בנים ימניים) ולכן הפרשי גבהי העצים בפעולות אלו יהיו נמוכים.

לעומת זאת, פעולת ה-$join$ האחרונה תבצע $join$ בין עץ ריק (תת העץ הימני של האיבר המקסימלי בתת העץ השמאלי, שהוא עלה ולכן תת העץ ריק) לבין כל תת העץ הימני של העץ כולו.

הפרש הגבהים במקרה זה הוא $h-1$ וזו תהיה העלות המקסימלית ל-$join$ כחלק מ-$split$ זה. נקבל:

$$maxCost = O(h-1) = O(\log(n))$$

שאלת בונוס

נשים לב שעבור $split$ של איבר אקראי, עלות ה-$join$ המקסימלי היא אורך הרצף המקסימלי של עליות מהאיבר לשורש "מאותו הכיוון" (כלומר שרשרת של בנים שמאליים או שרשרת של בנים ימניים).

נשים לב, שאיבר הוא בן שמאלי בהסתברות $p = \frac{1}{2}$ ובן ימני בהסתברות $p = \frac{1}{2}$.

לכן, אם נתבונן באיבר האקראי הנבחר וברצף האבות ממנו עד השורש, נרצה לספור את כמות הבנים הרצופים מאותו הכיוון, מה שתואם לצורת התפלגות גיאומטרית אותה ראינו בקורס מבוא להסתברות.

ובפרט:

- רצף של שני איברים מאותו הכיוון יתקבל בהסתברות $\left(\frac{1}{2}\right)^1$ ועלות ה-$join$ תהיה 2.
- רצף של שלושה איברים מאותו הכיוון יתקבל בהסתברות $\left(\frac{1}{2}\right)^2$ ועלות ה-$join$ תהיה 3.
- רצף של כל האיברים מאותו הכיוון יתקבל בהסתברות $\left(\frac{1}{2}\right)^{h-1}$ ועלות ה-$join$ תהיה h.

ממשפט התוחלת השלמה נקבל את הסכום:

$$maxCost = \sum_{i=1}^{h} \frac{i}{2^{i-1}}$$

מחיפוש ברשת האינטרנט נמצא כי אורך הרצף המקסימלי (בדומה לבעיית רצפים בהטלת מטבע) הוא $O(\log(h) = O(\log(\log(n))$. אם אורך הרצף המקסימלי הוא $O(\log(h))$, עלות ה-$join$ תהיה $O(\log(h))$ ומכאן נקבל את התוצאה $O(\log(\log(n))$.