**מבני נתונים – פרוייקט מספר 1 – עץ מאוזן**

מגיש 1:

יעקב אודסר

iakovodesser

209860188

מגיש 2:

איל ספיר

eyalsapir

206405417

להלן תיעוד חיצוני של המחלקות "צומת" ו"עץ" ושל הפונקציות שבהן:

**המחלקה AVLNode**

בכל צומת אנחנו מתחזקים מצביעים לשני הבנים ולאבא ולגובה (המרחק מהעלים). המצביע לאבא דרוש, לדוגמה, עבור פעולות האיזון. הגובה דרוש עבור שמירה על איזון תתי העצים.

פונקציות עזר: הגדרנו שלוש פונקציות עזר שישמשו אותנו בהמשך: is\_real\_node, max\_children\_height, ו-balance. כולן ניתנות לחישוב בהסתמך על ערכי הבנים שלהן, ולכן לא יפגעו בסיבוכיות של הפונקציות שוקראות להן.

**המחלקה AVLTree**

בכל עץ אנחנו מתחזקים מצביעים לשורש, למקסימום (עבור פעולות finger) לגודל העץ (עבור הפונקציה שמחזירה את גודל העץ) ולעלה וירטואלי בגובה מינוס 1 (כדי למנוע מצב של עלה ללא בנים, וככה להקל על ביצוע איזונים).

**פונקציית search:** הפונקציה קוראת לפונקציית עזר שעוברת על העץ בלוגיקה של חיפוש בינארי מהשורש עד המפתח הנחוץ. בחרנו במימוש איטרטיבי לשם הפשטות, ולכן ניתן לראות שמדובר בחיפוש בסיבוכיות O(log*n*). לאחר מכן הגדרנו פונקציית עזר בשם pointer\_only המחזירה את הצומת המבוקש ללא המסלול לשם פשטות המימוש בפונקציות שקוראות ל- search. פונקציית העזר ניגשת לערך הראשון ב-tuple בזמן קבוע.

**פונקציית finger\_search:** הפונקצייה נעזרת במצביע השמור למקסימום ומתקדמת ממנו איטרטיבית עד הצומת הראשון שהמפתח המבוקש k נמצא בתת העץ שלו (במימוש שלנו: פשוט בודקים מתי הגענו למפתח קטן מ-k). הילוך כזה לוקח O(log*k*) פעולות, ולאחר מכן ההגעה למפתח בעזרת חיפוש רגיל לוקחת זמן קבוע בלבד. בפונקציות הבאות נראה איך שומרים על המצביע למקסימום מבלי לפגוע בסיבוכיות הפעולות האחרות.

**פונקציית insert:** קודם כל אנחנו יוצרים צומת חדש. אם העץ ריק, אז רק צריך לעדכן את ערכי המקסימום ואת גודל העץ ל-1. אחרת, צריך להכניס אותו בתור בן של צומת קיים. בשביל זה בנינו את פונקציית העזר insert\_as\_child. בפונקציית העזר אנחנו שומרים שני ערכים: צומת שנצטרך לאזן בסוף התהליך, ומספר פעולות ה-promotion שביצענו. לאחר מכן אנחנו מחלקים למקרים:

ראשית אנחנו מבצעים חיפוש איטרטיבי של הצומת שאליו צריך לחבר את הערך החדש. חיפוש זה עולה O(log*n*). אנחנו מחברים את הערך לצומת שמצאנו ומגדירים אותו כעלה (גובה 0 ועל עלים וירטואליים כבנים), ואז מתחילים לבדוק אם צריך לעשות promotions. בתהליך אנחנו עולים לכל היותר עד השורש, לכן הוא לוקח O(log*n*) פעולות. בכל עלה אנחנו קוראים ל- max\_children\_height שמוצאת בזמן קבוע את הגובה של הילד המקסימלי ומצבעים promotion אם התגלה שהגובה של הילד שווה לגובה של הבן. בנוסף אנחנו קוראים ל- balance שבודקת את הפרש הגבהים בזמן קבוע, ואם מתגלה שיצרנו הפרש גבהים לא תקין, שומרים את הצומת לפונקציית האיזון.

בסוף התהליך אנחנו מעדכנים את הערך המקסימלי של העץ (בזמן קבוע) וקוראים לפונקציית האיזון אם היה צורך. זאת פונקציית עזר בשם rebalance. הפונקצייה ארוכה אבל חוזרת על עצמה אז נסביר רק את הרעיון. אנחנו שומרים מצביעים לצומת שצריך לאזן ולאבא שלו, ומתקדמים בשלבים: בודקים איזה צד "כבד" יותר – נניח במקרה הזה שהימני – ואז בודקים בתוכו: אם תת העץ הימני שלו מאוזן לחלוטין או עם הפרש גבהים 1 כך שהחלק הימני גדול יותר, זה המקרה ה"פשוט" שבו צריך לבצע רק רוטציה שמאלה. אחרת, זה המקרה ה"קשה" שבו צריך לבצע רוטציה ימינה ולאחר מכן רוטציה שמאלה. בסוף כל הפעולות אנחנו מעדכנים את הגבהים בפעולות שעולות זמן קבוע בלבד, וכאמור המקרה השמאלי סימטרי.

לסיכום: בתהליך ההכנסה לעץ אנחנו מבצעים חיפוש בסיבוכיות O(log*n*), לאחר מכן מחפשים צמתים לאזן ב- O(log*n*) ולבסוף מאזנים צומת אחד בלבד בזמן קבוע, ולכן פעולת החיפוש כולה לוקחת בסך הכול O(log*n*).

**פונקציית finger\_insert:** באופן אנלוגי לחלוטין ל- finger\_search אנחנו מתקדמים איטרטיבית מהמקסימום עד הצומת הראשון שהערך שאנחנו מעוניינים להכניס קטן ממנו, ואז קוראים ל- insert הרגילה, כמובן תוך התחשבות במקרה הקצה של עץ ריק. לפיכך הסיבוכיות תהיה O(log*k*) כמו finger\_search.

**פונקציית delete:** אנחנו מחפשים את הצומת, נעזרים במצביע לאבא, מחלקים למקרים ובסוף התהליך בודקים אם צריך לאזן את העץ. במקרה שהצומת שרצינו למחוק הוא עלה – פשוט מנתקים אותו מהעץ ומחברים את אבא לעלה הווירטואלי. אם לעלה יש רק בן אחד, אנחנו נעזרים בפונקיית עזר פשוטה בשם switcheroo בשביל להחליף בין הצומת לבן שלו ולמחוק את הצומת כעלה.

האפשרות האחרונה היא שלצומת יש שני בנים. במקרה הזה אנחנו נעזרים בפונקציית עזר בשם get\_successor. הפונקצייה מבצאת בצורה איטרטיבית את האלגוריתם שלמדנו בהרצאה למציאת יורש במקרה הגרוע צריך לעלות עד השורש, ולכן היא בסיבוכיות O(log*n*). אחרי שמצאנו את היורש אנחנו משנים את ערכי הצומת שלנו לערכי היורש ומוחקים את היורש.

בסך הכול, כמו שהוכחנו בכיתה, פעולת המחיקה דורשת במקרה הגרוע למצוא את הערך, למצוא יורש ולאזן את העץ – כולן פעולות שחסומות על ידי גובה העץ, ולכן עלות המחיקה כולה היא O(log*n*).

**פונקציית join:** הגדרנו שתי פוקנציות עזר, key\_size\_matters ו- height\_mattersשבודקות בזמן קבוע מי העץ בעל המפתחות הגדולים יותר ומי העץ הגבוה היותר. אנחנו מעדכנים את הגודל והערך המקסימלי של העץ בזמן קבוע ואז מחלקים למקרים: במקרה ששני העצים בגובה דומה, אנחנו מכניסים את שניהם כבנים של ערך הביניים ומגדירים את ערך הביניים כשורש. במקרה הקשה, אנחנו מטיילים למטה מהשורש של העץ הגבוה (מצד ימין או שמאל לפי גודל הערכים) ועוצרים בנקודה שבה הגבהים כמעט שווים. אנחנו מחברים את ערך הביניים כבן של הצומת שבו עצרנו, וכתתי עצים של ערך הביניים את העץ הקטן ואת מה שניתקנו מהצומת שאליו חיברנו את צומת הביניים.

לאחר התהליך אנחנו מעדכנים גבהים בזמן גבוה ובודקים אם צריך לבצע איזון. בשונה מהפעולות האחרות, במקרה הנ"ל ייתכן שנצטרך לבצע שני איזונים ולא רק אחד, ולכן אנחנו לא מגבילים את מספר האיזונים, אבל עדיין זה יוצא זמן קבוע, ולכן הסיבוכיות עולה לנו בסך הכול כמו שעולה המעבר מהשורש של העץ הגדול עד נקודת החיבור, כלומר O(heightT1 – heightT2 + 1).

**פונקציית split:** הגדרנו פונקצייה שמאתחלת שני עצי AVL חדשים בשמות left ו-right ומגדירה כשורשים שלהם את הבנים הימניים והשמאליים של הצומת שעליו עושים את הפיצול. לאחר מכן אנחנו מטיילים עד השורש באמצעות קריאות parent ובכל פעם מנתקים את תת העץ של האבא וקוראים ל-join עם עץ הימני או השמאלי, בהתאם. בגלל שאנחנו מאחדים שני עצי AVL מאוזנים, גם התוצאה יוצאת מאוזנת, והוכחנו בכיתה שסדרת איחודים כזאת מתרחשת בסיבוכיות O(log*n*).

**פונקציית avl\_to\_array:** הפונקצייה קוראת לפונקצייה רקורסיבית in\_order\_plus שמקבלת את השורש ומחזירה רשימה של ערכי העץ על בסיס הילוך in order. בתהליך אנחנו קוראים ל- in\_order\_plusעל הבן השמאלי, מחזירים את השורש וקוראים ל- in\_order\_plus על העץ הימני, וכך עד שמגיעים לעלים. כפי שהוכחנו בכיתה, הילוך כזה עולה O(n).

**פונקציית max\_node, size ו-** **get\_root:** הפונקציות מחזירות את הערך המקסימלי, השורש, והגודל שתיחזקנו בהגדרות העץ ולכן מוחזרים בזמן קבוע.

**החלק הניסויי/תיאורטי**

**שאלה 1:** "יש למלא בטבלה הבאה את סך עלויות האיזון ללא גלגולים עבור כל אחד מהניסויים. הסבירו מהו החסם העליון התאורטי על סך עלויות האיזון כולל גלגולים, והאם הערכים בטבלה מתאימים. לסיום, נמקו מדוע תוספת הגלגולים לספירה אינה משנה אסימפטוטית."

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| עלות איזון במערך עם היפוכים סמוכים אקראיים | עלות איזון במערך מסודר אקראית | עלות איזון במערך ממוין-הפוך | עלות איזון במערך ממוין | מספר סידורי |
| 596 | 482 | 644 | 644 | 1 |
| 1209 | 993 | 1308 | 1308 | 2 |
| 2505 | 1960 | 2638 | 2638 | 3 |
| 4938 | 3837 | 5300 | 5300 | 4 |
| 10081 | 8020 | 10626 | 10626 | 5 |
| 20094 | 15849 | 21280 | 21280 | 6 |
| 40110 | 32102 | 42590 | 42590 | 7 |
| 80677 | 63675 | 85212 | 85212 | 8 |
| 160893 | 127786 | 170458 | 170458 | 9 |
| 321566 | 255401 | 340952 | 340952 | 10 |

העלויות הגיוניות. נחשוב על זה כך: מערך ממוין ומערך ממוין הפוך ייתנו תוצאות סימטריות לחלוטין: בשני המקרים נצטרך להכניס את הערך החדש (בין אם מקסימלי או מינימלי) לקצה (הימני או השמאלי) של העץ ולטפס בדיוק את אותו מספר של שלבים כדי לבצע את התיקון (בין אם מצד ימין עד השורש או מצד שמאל עד השורש).

העלויות במערך עם היפוכים סמוכים יהיו כמעט זהות למערך ממוין, כי למעט מקרים בודדים שבהם נצטרך לטפס שלב אחד פחות, החל משלב מסוים סדרי הגודל שנצטרך לטפס יהיו דומים, כי הרשימה כמעט ממוינת. לעומת זאת במערך מסודר אקראית, יהיה לנו חיסכון ניכר במשאבי איזון כי חלק גדול מהערכים שנכניס ייכנסו קרוב לשורש ולכן יידרשו לטפס מעט מאוד שלבים כחלק מהאיזון.

בכיתה למדנו שהעלות האמורטייזד של הכנסת איבר לעץ כאשר כל האיברים מוכנסים ברציפות היא O(1), לכן החסם העליון התיאורטי של סך עלויות האיזון כולל גלגולים הוא O(n), דבר שמתיישב עם עלויות האיזון בטבלה שלינאריות בגודל הקלט (עלות האיזון עבור קלט בגודל n יוצאת על פי הטבלה בסביבות 3n עבור כל הערכים שבדקנו).

לבסוף, הוכחנו בכיתה גם שעלות החיפוש האמורטייזד של פעולת איזון כולל גלגולים היא O(1) ולכן תוספת הגלגולים לא משנה אסימפטוטית.

**שאלה 2:** "בהינתן מערך A בגודל n נגדיר היפוך בתור זוג אינדקסים i<j כך שמתקיים A[i]>A[j], ונסמן את מספר ההיפוכים הכולל במערך ב-I. ניתן לשים לב כי באופן כללי וככל שיש פחות היפוכים כך המערך קרוב יותר לממוין. יש למלא בטבלה הבאה את מספר ההיפוכים במערך הקלט עבור כל אחד מהניסויים (מספיק עד i=5)."

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי i | מספר היפוכים במערך ממוין | מספר היפוכים במערך ממוין-הפוך | מספר היפוכים במערך מסודר אקראית | מספר היפוכים במערך עם היפוכים סמוכים אקראיים |
| 1 | 0 | 24531 | 12006 | 116 |
| 2 | 0 | 98346 | 49585 | 231 |
| 3 | 0 | 393828 | 191376 | 432 |
| 4 | 0 | 1576200 | 799333 | 870 |
| 5 | 0 | 6306576 | 3107082 | 1792 |

**שאלה 3:** "יש למלא בטבלה הבאה את סך עלויות החיפוש עבור כל אחד מהניסויים."

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| עלות חיפוש במערך עם היפוכים סמוכים אקראיים | עלות חיפוש במערך מסודר אקראית | עלות חיפוש במערך ממוין-הפוך | עלות חיפוש במערך ממוין | מספר סידורי |
| 1415 | 3636 | 3559 | 1086 | 1 |
| 2844 | 8584 | 8023 | 2194 | 2 |
| 5921 | 18896 | 17841 | 4412 | 3 |
| 11995 | 41265 | 39255 | 8850 | 4 |
| 24945 | 93769 | 85637 | 17728 | 5 |
| 50976 | 204235 | 185507 | 35486 | 6 |
| 104009 | 448650 | 399457 | 71004 | 7 |
| 211114 | 974718 | 855775 | 142042 | 8 |
| 430651 | 2087582 | 1825245 | 284120 | 9 |
| 879659 | 4474449 | 3877851 | 568278 | 10 |