מבני נתונים - פרויקט מספר 1 - עץ דרגות

מגישים:
שם: איתי צמח
שם משתמש: itaizemah
ת.ז: 209637453

שם:עודד כרמון
שם משתמש: ododecarmon
ת.ז: 208116517

תיעוד המחלקה:
המחלקה AVLTree מממשת עץ AVL המכיל מחרוזות ממוספרות על ידי מספרים טבעיים שונים.
צמתי העץ מוגדרים על ידי המנשק הפנימי IAVLNode שממומש על ידי המחלקה הפנימית AVLNode.

למחלקה 3 שדות:

- מספר שלם size המתאר את גודל העץ, מאותחל כ-0.
- צומת EXT, אובייקט אחד משותף המייצג את כל הצמתים הוירטאולים שיושבים בעץ כבנים של העלים.
- צומת root, שורש העץ, מאותחל כ-EXT.

מתודות המחלקה:

- public boolean empty()
  מתודה זו מחזירה True אם ורק אם הגודל של העץ הוא 0. פועלת בסיבוכיות $O(1)$ כיוון שמבצעת פעולת השוואה אחת על שדה של המחלקה ומחזירה את התוצאה.
- public String search(int k)
  מתודה זו מחזירה את המחרוזת המזוהה עם המפתח k אם הוא מופיע בעץ ואחרת מחזירה null. המתודה קוראת למתודת העזר searchNode(int k) הפועלת בסיבוכיות $O(\log n)$ ומקבלת ממנה את הצומת המתאים, לאחר מכן

מפעילה על צומת זה את getKey() הפועל בסיבוכיות $O(1)$ ומלבד זאת מבצעת רק כמות קבועה של פעולות לכן גם מתודה זו פועלת בסיבוכיות $O(\log n)$.

- private IAVLNode searchNode(int k)- מימוש האלגוריתם שהוצג בכיתה, לחיפוש צומת בעץ, והחזרתו. משתמשת בהיותו של עץ AVL עץ חיפוש בינארי, אך עוצרת בהתאם כדי להמנע מהגעה לצמתים וירטואליים בעץ. סיבוכיות: $O(\log n)$. עוברים בכל רמה פעם אחת בדיוק- השוואה עם צומת האב תעדכן את הפעולה אם ללכת לכיוון שמאל או ימין.
- private void rotate(IAVLNode parent, IAVLNode child)- סיבוכיות זמן ריצה: $O(1)$. מקבלת מצביע לאב ובנו בעץ, ועל פי המצביעים הנ"ל מבצעת רוטציה/ רוטציה כפולה לפי ההנחיות בשקף 31 במצגת על BST.\
- public int insert(int k, String i)- מכניסה לעץ צומת חדשה, במיקום שבו הייתה מוכנסת בעץ הנוכחי על פי החוקים של עץ בינארי פשוט, וקוראת לפעולה rebalanceInsert שמתקנת את העץ כך שחוקי AVL יישמרו. סיבוכיות זמן ריצה: $O(\log n)$. מחפשים את המיקום המתאים בעץ כמו בעץ חיפוש בינארי רגיל, וקוראים לrebalanceInsert שעולה אף היא $O(\log n)$. המתודה מחזירה לבסוף את מספר פעולות האיזון שנדרשו להכנסה לעץ תוך שמירה על תכונת הAVL, או -1 אם הצומת כבר בעץ.
- private int rebalanceInsert(IAVLNode x, IAVLNode y)- מבצע את כל התיקונים להכנסה של הצומת שהוכנסה זה עתה לעץ (או בין שני צמתים כלליים בהם יש חשד להפרה), על פי פירוט המקרים במצגת WAVL, שקף 22, והמקרים הסימטריים להם. סיבוכיות זמן ריצה: $O(\log n), \Omega(1)$. תיקון יחיד/ רוטציה יחידה בעץ עולות זמן קבוע, ובמקרה הטוב ביותר לא נצטרך לעלות ולתקן רמות גבוהות יותר בעץ, כך שזמן הריצה של המתודה יישאר קבוע. במקרה הגרוע ביותר, נצטרך לעלות ולתקן את העץ עד שנגיע לשורש, ואז נקבל כי זמן הביצועי של המתודה הוא $O(height) = O(\log n)$. תחזיר את מספר פעולות האיזון בעבור ההכנסה.
- public int delete(int k)- מוחק את הצומת שהמפתח שלו שווה לk, אם נמצא בעץ. המתודה תחזיר את מספר פעולות האיזון שנעשו כדי לשמור על האינווריאנטות של עץ AVL. במידה ולא נעשו פעולות תיקון, יוחזר 0, ואם המפתח לא בעץ, נחזיר -1.

- public String[] infoToArray()- מחזיר מערך
-

מדידות:

1.

מספר סידורי	מספר פעולות	מספר פעולות האיזון הממוצע לפעולת insert	מספר פעולות האיזון הממוצע לפעולת delete	מספר פעולות האיזון המקסימלי לפעולת insert	מספר פעולות האיזון המקסימלי לפעולת delete
1	10,000	3.4221	2.4035	15	29
2	20,000	3.41445	2.4027	17	31
3	30,000	3.4008667	2.4250333	17	37
4	40,000	3.404325	2.411225	17	32
5	50,000	3.41506	2.41682	18	35
6	60,000	3.4247	2.414	18	39
7	70,000	3.4026144	2.4136286	18	32
8	80,000	3.40765	2.412825	18	35
9	90,000	3.4206777	2.4145555	19	37
10	100,000	3.41414	2.4145555	19	38

התוצאות שהיינו מצפים לקבל, על סמך ההסבר התיאורטי של עצי AVL שנלמד בכיתה, היא שברצף של הכנסות וברצף של מחיקות - מספר פעולות האיזון הממוצע הוא $O(1)$, ומספר פעולות האיזון המקסימלי הוא $O(\log n)$.
התוצאות שקיבלנו בפועל תואמות את הציפיות, ניתן לראות כי כאשר מסתכלים על כמות הפעולות הממוצעות (כלומר האמורטייזד) היא קבועה ללא תלות בכמות האיברים בעץ, בעוד שהכמות המקסימלית עולה אך בקצב איטי ביותר.
משמעות המדידות שביצענו היא שהניתוח התיאורטי משקף את תפקוד מבנה הנתונים בפועל.

2.

מספר סידורי	מספר פעולות	עלות join ממוצע עבור split אקראי	עלות join מקסימלי עבור split אקראי	עלות join ממוצע עבור split של איבר מס בתת העץ השמאלי	עלות join מקסימלי עבור split של איבר מקסימלי בתת העץ השמאלי
1	10,000				
2	20,000				
3	30,000				
4	40,000				
5	50,000				
6	60,000				
7	70,000				
8	80,000				
9	90,000				

				100,000	10