מסמך תיעוד - **RBTree**

להלן תיאור רכיבי המחלקה RBTree אשר מממשת עץ אדום-שחור.

שדות:

* root - שדה מסוג RBNode המייצג את שורש העץ. מאותל להיות null.
* minNode - שדה מסוג RBNode המייצג את הצומת עם המפתח המינימלי בעץ. מאותל להיות null.
* maxNode - שדה מסוג RBNode המייצג את הצומת עם המפתח המקסימלי בעץ. מאותל להיות null.
* size - שדה מסוג int המייצג את מספר הצמתים בעץ. מאותחל להיות 0.
* dummyNode - שדה final מסוג RBNode המייצג צומת null שאליו מופנים הצאצים של עלים והאבא של השורש. השדה נועד להקל על מקרי קצה והוא מאותחל להיות צומת שחור עם המפתח 1-.

מתודות:

* getRoot - מחזירה מצביע לשורש העץ, אם העץ ריק מחזירה null.  
  סיבוכיות: (1)O.
* size - מחזירה את מספר הצמתים בעץ.  
  סיבוכיות: (1)O.
* empty - מחזירה true אם העץ ריק ו- false אחרת.  
  אופן פעולה: השוואת השדה size ל- 0.  
  סיבוכיות: (1)O.
* min - מחזירה את הערך של הצומת עם המפתח הקטן ביותר בעץ, אם העץ ריק המתודה תחזיר null.  
  סיבוכיות: (1)O.
* max - מחזירה את הערך של הצומת עם המפתח הdsuk ביותר בעץ, אם העץ ריק המתודה תחזיר null.  
  סיבוכיות: (1)O.
* search - מקבלת מפתח k ומחזירה את הערך של הצומת עם המפתח k, אם המפתח לא נמצא בעץ המתודה תחזיר null.  
  אופן פעולה: תהליך איטרטיבי כך שהמפתח k מושווה למפתח של הצומת הנוכחי (מתחיל בשורש), אם המפתחות שווים יוחזר הערך שבצומת ואחרת: אם המפתח k גדול יותר הצומת הבא יהיה הבן הימני, ואחרת הוא יהיה הבן השמאלי. התהליך ייפסק כאשר המפתח של הצומת הנוכחי הוא 1- (צומת null).  
  סיבוכיות: (O(log n, כי זה הוא גובה העץ.
* keysToArray - מחזירה מערך ממויין של כל המפתחות בעץ, אם העץ ריק המתודה תחזיר מערך ריק.  
  אופן פעולה: יצירת מערך חדש בגודל מספר האיברים וקריאה לפעולת העזר generateKeysArray.  
  סיבוכיות: (O(n, בגלל הסיבוכיות של פעולת העזר generateKeysArray.
* valuesToArray - מחזירה מערך ממויין לפי המפתחות של כל הערכים של הצמתים בעץ, אם העץ ריק המתודה תחזיר מערך ריק.  
  אופן פעולה: יצירת מערך חדש בגודל מספר האיברים וקריאה לפעולת העזר generateValuesArray.  
  סיבוכיות: (O(n, בגלל הסיבוכיות של פעולת העזר generateValuesArray.
* generateKeysArray - מקבלת מצביע לשורש העץ, מערך שיהווה את מערך המפתחות (צריך להיות בגודל של מספר הצמתים בעץ) ואינדקס שצריך להיות 0 בקריאה לפעולה. מכניסה למערך את המפתחות שבעץ באופן ממויין.  
  אופן פעולה: פעולה רקורסיבית שבכל שלב קוראת לפעולה על הבן השמאלי של הצומת הנוכחי בכדי למלא את החלק השמאלי של המערך, אז מכניסה את המפתח של הצומת הנוכחי למערך ואחר כך קוראת לפעולה על הבן הימני של הצומת הנוכחי בכדי למלא את החלק הימני של המערך. הפעולה מסיימת כאשר הצומת הנוכחי הוא null.  
  סיבוכיות: (O(n, כי המתודה עוברת בכל צומת בעץ פעם אחת.
* generateValuesArray - מקבלת מצביע לשורש העץ, מערך שיהווה את מערך הערכים(צריך להיות בגודל של מספר הצמתים בעץ) ואינדקס שצריך להיות 0 בקריאה לפעולה. מכניסה למערך את הערכים שבצמתים של העץ באופן ממויין לפי המפתחות.  
  אופן פעולה: פעולה רקורסיבית שבכל שלב קוראת לפעולה על הבן השמאלי של הצומת הנוכחי בכדי למלא את החלק השמאלי של המערך, אז מכניסה את הערך של הצומת הנוכחי למערך ואחר כך קוראת לפעולה על הבן הימני של הצומת הנוכחי בכדי למלא את החלק הימני של המערך. הפעולה מסיימת כאשר הצומת הנוכחי הוא null.  
  סיבוכיות: (O(n, כי המתודה עוברת בכל צומת בעץ פעם אחת.
* insert - מקבלת מפתח k וערך v ומכניסה צומת T עם ערכים אלה לעץ תוך שמירה על תקינותו כעץ אדום-שחור, בנוסף מחזירה את מספר החלפות הצבע שנדרשו בשלב תיקון העץ. אם המפתח כבר נמצא בעץ המתודה לא תשנה דבר ותחזיר 1-.  
  אופן פעולה: אם העץ ריק הצומת T יהפוך להיות השורש ויוחזר הערך 1 כי צבעו של השורש הוא שחור. אחרת, המתודה תמצא את הצומת P ש- T ייכנס כצאצא שלו באמצעות פעולת העזר getNodeWithKey ואם המפתח כבר נמצא בעץ יוחזר הערך 1- ולא ישתנה דבר. כעת, T יהפוך להיות בן שמאלי או ימני של P (לפי ערך המפתחות) ויבוצע תיקון של העץ באמצעות פעולת העזר insertFixup שגם תחזיר את מספר החלפות הצבע שבוצעו.  
  כמו כן, יעודכנו שדות המינימום והמקסימום.  
  סיבוכיות: (O(log n, בגלל הסיבוכיות של פעולות העזר getNodeWithKey ו- insertFixup.
* delete - מקבלת מפתח k ומוחקת מהעץ את הצומת בעל מפתח בעל ערך k, אם המפתח לא קיים בעץ הפונקציה מחזירה -1, אחרת מוחזר מספר החלפות הצבע אשר בוצעו. הפונקציה שומרת על חוקיות העץ כעץ אדום שחור.  
  אופן הפעולה: אם המפתח לא נמצא בעץ הפונקציה מחזירה -1, אחרת אם המפתח נמצא בעץ ראשית כל נגיע למצב שבו הצומת שעלינו למחוק הוא בעל לא יותר מצאצא אחד, אם יש לו שני צאצאים נחליף אותו עם ה successor שלו (אם יש שני צאצאים בהכרח יש successor) ונקבל עץ תקין ואז נמחק את ה successor מהמיקום המקורי שלו.  
  כעת עלינו למחוק צומת אשר יש לו עד צאצא אחד.  
  אם הצומת שעלינו למחוק הוא אדום ניתן פשוט למחוק אותו מהעץ (ע״י קריאה ל- removeNodeWithUpToOneChild) ולהישאר עם עץ תקין.  
  אם הצומת אינו אדום אבל יש לו צאצא אדום ניתן לצבוע את הצאצא בשחור למחוק את הצומת מהעץ ולהישאר עם עץ תקין. במידה והצאצא אינו אדום נמחק את הצומת מהעץ (שוב ע״י קריאה ל- removeNodeWithUpToOneChild) וניוותר עם עץ אשר בכדי שהוא יהיה תקין עלינו לספור את הצאצא של הצומת אשר נמחק כבעל צביעת שחור כפולה (double black). לכן נקרא לפונקציה fixDeleteDoubleBlack עם הצאצא של הצומת שכרגע מחקנו כפרמטר.  
  סיבוכיות: O(log n) בגלל הסיבוכיות של fixDeleteDoubleBlack, getNodeWithKey ו- findMinimalNode, אשר כולן מתרחשות בסיבוכיות זו.
* *-* findMinimalNode: בהינתן צומת מחזירה את הצומת בעלת המפתר הקטן ביותר בתת העץ הנפרש ע״י הצומת הזו (אם תת העץ ריק אז מוחזר הצומת עצמו)  
  אופן הפעולה: המתודה עוברת על העץ ובכל פעם פונה שמאלה עד שהיא מגיעה אל dummyNode ומחזירה את הערך שנמצא לפני שהגיע אל ה- dummy.  
  סיבוכיות: O(log n) משום שבמקרה הגרוע ביותר אנו עוברים מהצאצא הימני של השורש עד לצומת בעל המפתח הקטן ביותר בתת העץ הימני (אשר יכול להיות במרחק log n מהשורש).
* - removeNodeWithUpToOneChild מקבלת מצביע אל צומת אשר יש לו עד ילד אחד ומסירה אותו מהעץ.  
  אופן הפעולה: אם לצומת אין צאצאים הפונקציה מחליפה את המצביע אל הצומת למחיקה (בהורה שלו) ל-dummy.  
  אם לצומת יש צאצא הפונקציה מוצאת איזה צאצא יש לו (שמאלי או ימני שאינו dummy) ומחליפה את המצביע אל הצומת למחיקה במצביע אל הצאצא של הצומת למחיקה (בהורה).  
  סיבוכיות: O(1) יש רצף סופי של פעולות אשר מתבצעות בכל מקרא אשר אינו תלוי במספר האיברים בעץ.
* getNodeWithKey - מקבלת מצביע לשורש העץ ומפתח requiredKey ומחזירה מצביע לצומת עם המפתח requiredKey בעץ, אם המפתח לא נמצא בעץ המתודה תחזיר null.  
  אופן הפעולה: מציאת הצומת המתאים להכנסת המפתח באמצעות פעולת העזר getLocationToInsertNodeAt. החזרת הצומת אם הוא בעל המפתח המתאים ו-null אחרת.  
  סיבוכיות: (O(log n, בגלל הסיבוכיות של פעולות העזר getLocationToInsertNodeAt.
* getLocationToInsertNodeAt- מקבלת מצביע לשורש העץ ומפתח requiredKey ומחזירה מצביע לצומת עם המפתח requiredKey בעץ, אם המפתח לא נמצא בעץ המתודה תחזיר מצביע לאבא של צומת עם המפתח במקרה והיינו מכניסים אותו לעץ.  
  אופן פעולה: תהליך איטרטיבי כך שהמפתח requiredKey מושווה למפתח של הצומת הנוכחי (מתחיל בשורש), אם המפתחות שווים יוחזר הערך שבצומת ואחרת: אם המפתח k גדול יותר הצומת הבא יהיה הבן הימני, ואחרת הוא יהיה הבן השמאלי. התהליך ייפסק כאשר נגיע לעלה ואז נחזיר אותו כי מתחתיו אמור להיכנס צומת עם המפתח requiredKey.  
  סיבוכיות: (O(log n, כי זה הוא גובה העץ.
* - fixDeleteDoubleBlack מקבלת מצביע לצומת אשר בכדי שהעץ יהיה תקין צריך לספור אותה כשחורה פעמיים. המתודה מחזירה את מספר החלפות הצבע שהיא ביצעה.  
  אופן הפעולה: בכל פעם הפונקציה פותרת את הבעיה או מעלה את הבעיה לרמה יותר גבוהה בעץ או מעבירה את העץ למצב שממנו ניתן להעלות את הבעייה לרמה יותר גבוה, אם הבעיה לא נפתרה באיטרציה הנוכחית הפונקציה פועלת באופן רקורסיבי לפתירתה.  
  אם הצומת הבעייתי (הצומת עם צביעת השחור הכפול) הוא השורש אז אין באמת בעייה וסיימנו, לאחר מכן מוודאים שהצומת הבעייתי הוא שחור, אחרת נצבע אותו שחור וסיימנו.  
  כעת ישנם מספר מקרים:
  + אח של הצומת הבעייתי שחור ושני הילדים של האח שחורים:
    - אם ההורה שחור נצבע את האח באדום ועכשיו השורש הוא הבעייתי (ונקרא לפונקציה שוב עליו).
    - אם ההורה אדם נצבע אותו בשחור ואת האח באדום וסיימנו.
  + אח של הצומת הבעייתי שחור והצאצא הימני של האח אדום:
    - אם הצומת הבעייתי הוא בן שמאלי: נצבע את האח בצבע של ההורה, את ההורה נצבע בשחור ואת הבן הימני של האח נצבע בשחור, נבצע סיבוב שמאלה סביב ההורה וכעת העץ תקין וסיימנו.
    - אם הצומת הבעייתי הוא בן ימני: נצבע את האח בצבע של ההורה את ההורה בשחור ונבצע סיבוב ימינה סביב הצומת הבעייתי, אם הצומת שלפני הסיבוב היה הבן השמאלי של אח של הצומת הבעייתי הוא אדום נצבע אותו שחור וסיימנו, אחרת הוא הצומת הבעייתי החדש ונריץ את הפונקציה עליו.
  + אח של הצומת הבעייתי שחור והצאצא השמאלי של האח אדום (ניתן להניח כי הצאצא הימני שחור):
    - אם הצומת הבעייתי בן שמאלי: נצבע את הבן השמאלי של אחיו בשחור, את אחיו באדום ונסובוב סביב הצומת הבעייתי, כעת הגענו למצב שאנו יודעים לטפל בו וסיימנו (נריץ את הפונקציה שוב על הצומת שגם קודם סומן כבעייתי).
    - אם הצומת הבעייתי בן ימני: נצבע את האח בצבע של ההורה, את ההורה בשחור ואת הבן השמאלי של האח בשחור, נסובב ימינה סביב ההורה ונגיע לעץ תקין.
  + הצומת הבעייתי שחור ואחיו אדום:
    - אם הצומת הבעייתי בן שמאלי: נצבע את אחיו בשחור את אביב באדום ונסובב שמאלה סביב האב, כעת הצומת הבעייתי הוא עדיין הצומת המקורי אבל הגענו למצב שבו אנו יודעים לטפל ולכן נקרא שוב למתודה עליו.
    - אם הצומת הבעייתי בן ימני: נצבע את אחיו בשחורף את אביו באדום ונסובב ימינה סביב האב, כעת הצומת הבעייתי הוא עדיין הצומת המקורי אבל הגענו למצב שבו אנו יודעים לטפל ולכן נקרא שוב למתודה עליו.

סיבוכיות: O(log n) משום שבמקרה הגרוע ביותר אנו נתחיל מאחד העלים ונעלה על לשורש כאשר בכל צעד אנו עושים מספר פעולות החסום ע״י קבוע.

* insertFixup - מקבלת מצביע node לצומת שצריך תיקון (לאחר שבוצעה הכנסה לעץ) ומתקנת את העץ לפי המקרים הבאים (כאשר האבא של node הוא צומת אדום כי אחרת העץ תקין):
  + נניח שהאבא של node הוא בן שמאלי:
    - אם הדוד של node הוא צומת אדום, צריך להפוך את האבא והדוד לצמתים שחורים ואת הסבא לצומת אדום.
    - אם הדוד של node הוא צומת שחור ו- node הוא בן ימני, צריך לבצע רוטציה שמאלה בכדי להגיע למצב 3.
    - אם הדוד של node הוא צומת שחור ו- node הוא בן שמאלי, צריך להפוך את האבא ואת הסבא לצמתים שחורים ולבצע רוטציה ימינה בכדי לפתור את הבדלי הגבהים שנוצרו.
  + נניח שהאבא של node הוא בן ימני (מקרה סימטרי):
    - אם הדוד של node הוא צומת אדום, צריך להפוך את האבא והדוד לצמתים שחורים ואת הסבא לצומת אדום.
    - אם הדוד של node הוא צומת שחור ו- node הוא בן שמאלי, צריך לבצע רוטציה ימינה בכדי להגיע למצב 3.
    - אם הדוד של node הוא צומת שחור ו- node הוא בן ימני, צריך להפוך את האבא ואת הסבא לצמתים שחורים ולבצע רוטציה שמאלה בכדי לפתור את הבדלי הגבהים שנוצרו.

אופן פעולה: תהליך איטרטיבי שבו בכל פעם נבדק האם הצומת הנוכחי דורש תיקון (כלומר, האם האבא שלו הוא צומת אדום). במקרה שכן, נבדק לאיזה מהמצבים הוא מתאים ואז מתבצעים השינויים בהתאם. כמו כן, נספר מספר החלפות הצבע שבוצעו ובכל השלב הצומת הבא יהיה האבא/הסבא בהתאם למי הוא הצומת הבא שעשוי לדרוש תיקון.

הרוטרציות מתבצעות באמצעות פעולות העזר leftRotate ו- rightRotate.

סיבוכיות: (O(log n, כי זה הוא גובה העץ (הפעולה מתחילה מלמטה ומתפסת לכל היותר עד שורש העץ).

* leftRotate - מקבלת מצביע node לצומת בעץ ומבצעת רוטציה שמאלה כפי שנלמד בשיעור.  
  אופן פעולה: מחליפה בין node והבן הימני שלו right באמצעות פעולת העזר replaceNodes, הופכת את הבן השאמלי של right להיות הבן הימני של node באמצעות פעולת העזר makeRightChild והופכת את node להיות הבן השאמלי של right באמצעות פעולת העזר makeLeftChild.  
  סיבוכיות: (1)O.
* rightRotate- מקבלת מצביע node לצומת בעץ ומבצעת רוטציה ימינה כפי שנלמד בשיעור.  
  אופן פעולה: מחליפה בין node והבן השמאלי שלו left באמצעות פעולת העזר replaceNodes, הופכת את הבן הימני של left להיות הבן השמאלי של node באמצעות פעולת העזר makeLeftChild והופכת את node להיות הבן הימני של left באמצעות פעולת העזר makeRightChild.  
  סיבוכיות: (1)O.
* makeLeftChild - מקבלת צומת parent וצומת child בעץ והופכת את child להיות הבן השמאלי של parent.  
  סיבוכיות: (1)O.
* makeRightChild- מקבלת צומת parent וצומת child בעץ והופכת את child להיות הבן הימני של parent.  
  סיבוכיות: (1)O.
* replaceNodes - מקבלת צומת nodeToReplace וצומת replacingNode בעץ ושמה את הצומת replacingNode במקום הצומת nodeToReplace.  
  סיבוכיות: (1)O.

מחלקה פנימית - **RBNode:**

מחלקה המייצגת צומת בעץ.

שדות:

* value - שדה String המייצג את הערך השמור בצומת.
* key - שדה int המייצג את המפתח של הצומת בעץ.
* left - שדה RBNode המייצג מצביע לבן השמאלי של הצומת בעץ.
* right - שדה RBNode המייצג מצביע לבן הימני של הצומת בעץ.
* parent - שדה RBNode המייצג מצביע לאבא של הצומת בעץ.
* is\_red - שדה boolean המייצג את צבעו של הצומת (true אם הצומת אדום ו- false אם הוא שחור).

מתודות:

* RBNode - שיטה בונה המקבלת מפתח key, ערך value וארגומנט בוליאני is\_red ומאתחלת שדות אלה בהתאם. השדות left, right ו- parent מאותחלים ל- null.  
  סיבוכיות: (1)O.
* isRed - מחזירה true אם הצומת אדום, ו- false אם הוא שחור.  
  סיבוכיות: (1)O.
* getLeft - מחזירה מצביע לבן השמאלי של הצומת, null אם אין לו בן שמאלי.  
  סיבוכיות: (1)O.
* getRight- מחזירה מצביע לבן הימני של הצומת, null אם אין לו בן ימני.  
  סיבוכיות: (1)O.
* getParent- מחזירה מצביע לאבא שלא הצומת, null אם אין לו אבא.  
  סיבוכיות: (1)O.
* getValue - מחזירה את הערך ששמור בצומת.  
  סיבוכיות: (1)O.
* getKey - מחזירה את המפתח של הצומת בעץ.  
  סיבוכיות: (1)O.

מדידות:

בכיתה למדנו שמספר החלפות הצבע בפעולות insert ו- delete הוא (O(1 בניתוח amortized ולכן ציפינו לקבל מספר החלפות ממוצע קבוע לכל פעולה בלי קשר למספר הפעולות שבוצעו. אכן קיבלנו מספר החלפות קבוע כאשר מתבצעות בערך 2.3 החלפות בפעולת insert ובערך 2.45 החלפות בפעולת delete.

|  |  |  |  |
| --- | --- | --- | --- |
| מספר סידורי | מספר פעולות | מספר החלפות צבע ממוצע לפעולת insert | מספר החלפות צבע ממוצע לפעולת delete |
| 1 | 10,000 | 2.3174 | 2.4679 |
| 2 | 20,000 | 2.3135 | 2.46485 |
| 3 | 30,000 | 2.31566 | 2.4753 |
| 4 | 40,000 | 2.3166 | 2.467975 |
| 5 | 50,000 | 2.30864 | 2.46538 |
| 6 | 60,000 | 2.31066 | 2.463716 |
| 7 | 70,000 | 2.32308 | 2.4645 |
| 8 | 80,000 | 2.317 | 2.463 |
| 9 | 90,000 | 2.3186 | 2.465388 |
| 10 | 100,000 | 2.32678 | 2.46656 |