עץ אדום-שחור: תרגיל מעשי

תום סגל 208945519, גל ורניק 208884213

המחלקה RBTree

שדות:

שם השדה	טיפוס	תפקיד
root	RBNode	שומר את השורש של העץ
size	int	גודל העץ- מספר הצמתים בעץ.
RBNode	מחלקה פנימית	מחלקה פנימית

המחלקה הפנימית RBNode

שדות:

שם השדה	טיפוס	תפקיד
isRed	boolean	שומר את צבע ה-node: true אם אדום ו-false אחרת.
key	int	מפתח הצומת
value	String	הערך שבצומת
parent	RBNode	האבא של הצומת
left	RBNode	בן שמאלי
right	RBNode	בן ימני

המתודות הפנימיות של המחלקה RBNode:

פעולות get עבור השדות key, left, right ו-isRed.

המתודות של RBTree:

RBTree
נראות: public
קלט: אין
פלט: אין
סיבוכיות: O(1)
המתודה היא בנאי לעץ אדום-שחור ריק.

getRoot
נראות: public
קלט: אין
פלט: RBNode
סיבוכיות: O(1)
המתודה מחזירה את שורש העץ. אם העץ ריק, היא מחזירה null.

empty
נראות: public
קלט: אין
פלט: boolean
סיבוכיות: O(1)
המתודה מחזירה true אם העץ ריק ו-false אחרת.

search
נראות: public
קלט: int k
פלט: String
סיבוכיות: O(logn)
המתודה קוראת ל-treeSearch ושולחת לו את שורש העץ, על מנת לבצע חיפוש בעץ כולו. היא מחזירה את ה-value השמור בצומת שהמפתח שלה k, ו-null אם לא קיים כזה בעץ.

treeSearch
נראות: private
קלט: RBNode node, int k
פלט: String
סיבוכיות: O(logn)
המתודה מקבלת צומת node וערך, ומבצעת חיפוש בתת-העץ שהשורש שלו הוא node. היא מחזירה את ה-value השמור בצומת שהמפתח שלה k, ו-null אם לא קיים כזה בתת-העץ.

insert
נראות: public
קלט: int k, String v
פלט: int
סיבוכיות: O(logn)
המתודה מכניסה את הצומת שהמפתח שלו k וה-value שלו v לעץ. לאחר מכן היא מפעילה את המתודה insertFixup, שמבצעת תיקונים בעץ. המתודה מקבלת מ-insertFixup את מספר התיקונים בעץ, ומחזירה את אותו המספר. אם כבר קיים בעץ צומת שהמפתח שלו k, המתודה מחזירה 1-.

insertFixup
נראות: private
קלט: RBNode z
פלט: אין
סיבוכיות: O(logn), Amortized O(1)

המתודה מקבלת צומת לאחר שהוכנסה לעץ. היא מבצעת תיקון לעץ על מנת לשמור את התכונות שלו כעץ אדום-שחור- היא מוודא שהגובה השחור נשאר תקין, וכן שאר התנאים. היא עושה זאת ע"י ביצוע סיבובים והחלפות כפי שנלמד בשיעור ובספר הקורס.

delete
נראות: public
קלט: int k
פלט: int
סיבוכיות: O(logn), Amortized O(1)
המתודה מקבלת מפתח, ומסירה את הצומת המתאימה שזהו המפתח שלה מהעץ. היא עושה זאת ע"י חיפוש הצומת המתאימה באמצעות treePosition ומחיקת הצומת. היא מבצעת סיבובים על מנת לשמור על תכונת העץ כעץ חיפוש בינארי. לאחר מכן, היא קוראת למתודה deleteFixup שמוודאה שהעץ שומר על תכונותיו כעץ אדום-שחור. היא מקבלת מdeleteFixup ומעבירה הלאה את מספר החלפות הצבע שנעשו בעץ. במידה ולא קיים k אותו אנו רוצים למחוק בעץ, היא מחזירה 1-.

deleteFixup
נראות: private
קלט: אין
פלט: int
סיבוכיות: O(logn), Amortized O(1)
המתודה מקבלת מפעולת ה-delete צומת ממנו יש להתחיל לתקן את העץ. היא מוודא שהעץ שומר על תכונותיו כעץ אדום-שחור: היא עושה זאת על ידי מעבר על כל אחד מארבעת הקייסים האפשריים לאחר מחיקה, ומבצעת את שינויי הצבע המתאימים. היא מחזירה את מספר שינויי הצבע שנעשו. היא מבצעת שימוש במתודה הפנימית transplant.

min
נראות: public
קלט: אין
פלט: String
סיבוכיות: O(logn)
המתודה מחיזרה את ה-value של הצומת בעל המפתח הקטן ביותר בעץ. אם העץ ריק, היא מחזירה null.

max
נראות: public
קלט: אין
פלט: String
סיבוכיות: O(logn)
המתודה מחיזרה את ה-value של הצומת בעל המפתח הגדול ביותר בעץ. אם העץ ריק, היא מחזירה null.

keysToArray
נראות: public
קלט: אין
פלט: [] int
סיבוכיות: O(n)
המתודה מחזירה מערך ממוין המכיל את כל המפתחות של העץ, או מערך ריק אם העץ ריק.

valuesToArray
נראות: public
קלט: אין
פלט: [] String
סיבוכיות: O(n)

המתודה מחזירה מערך המכיל את כל המחרוות של העץ, ממוין לפי סדר המפתחות המתאימים בעץ. כלומר, הערך ה i במערך הוא המחרוזת המתאים למפתח שיופיע במיקום ה i במערך הפלט של הפונקציה keysToArray(). גם הפונקציה הזאת מחזירה מערך ריק אם העץ ריק.או מערך ריק אם העץ ריק.

rank
נראות: public
קלט: int k
פלט: אין
סיבוכיות: O(k)
הפעולה מחזירה את מספר האיברים בעץ עם מפתח שקטן מ-k. היא עושה זאת ע״י מציאת המיקום שבו נמצא הצומת שהמפתח שלו k (או הצומת שהכי קרוב אליו), ומשם סופרת כמה פעמים עליה לקרוא ל-predeccesor עד שתגיע למינימום של העץ ותקבל null בקצה שלו.

size
נראות: public
קלט: אין
פלט: int
סיבוכיות: O(1)
המתודה מחזירה את השדה השמור של גודל העץ- מספר המפתחות בעץ.

successor
נראות: private
קלט: RBNode node
פלט: RBNode
סיבוכיות: O(logn), Amortized O(1)
המתודה הפנימית מקבלת צומת ומחזירה את הצומת העוקב אחריו בעץ, כלומר את הצומת הבא עם המפתח הקרוב ביותר למפתח שלו, שגדול ממנו.

predecessor
נראות: private
קלט: RBNode node
פלט: RBNode
סיבוכיות: O(logn), Amortized O(1)
המתודה הפנימית מבצעת את הפעולה ההפוכה מזו של succesor- היא מקבלת צומת ומחזירה את הצומת העוקב לפניו בעץ, כלומר את הצומת הבא עם המפתח הקרוב ביותר למפתח שלו, שקטן ממנו.

transplant
נראות: private
קלט: RBNode x, RBNode y
פלט: אין
סיבוכיות: O(1)
המתודה הפנימית מקבלת שני צמתים, x ו-y, ומחליפה את תת-העץ של x בתת העץ של y על ידי חיבור ההורה של x ל-y (משני הכיוונים).

leftRotate
נראות: private
קלט: RBNode x
פלט: אין
סיבוכיות: O(1)
המתודה הפנימית מקבלת צומת x, ומבצעת בו סיבוב שמאלי כפי שנלמד בהרצאה- כלומר הופכת את הבן הימני שלו לאבא שלו וכן הלאה.

rightRotate
נראות: private
קלט: RBNode x
פלט: אין
סיבוכיות: O(1)
המתודה הפנימית מקבלת צומת x, ומבצעת בו סיבוב ימני כפי שנלמד בהרצאה- כלומר הופכת את הבן השמאלי שלו לאבא שלו וכן הלאה.

treePosition
נראות: private
קלט: int k
פלט: RBNode
סיבוכיות: O(logn)
המתודה הפנימית מבצעת פעולה דומה ל-search: היא מקבלת מפתח k, ומחפשת את הצומת בעץ שערכו k. במידה ומצאה את הצומת, היא מחזירה אותו. במידה ולא, היא תחזיר את הסנטינל הפנימי nil.

toString
נראות: public
קלט: אין
פלט: String
סיבוכיות: O(n)
המתודה מייצרת באופן רקורסיבי מחרוזת המייצגת את העץ אדום-שחור. היא עושה זאת ע״י קריאה לפעולה הפנימית toString שפועלת באופן רקורסיבי.לאחר מכן היא מחזירה את המחרוזת שנוצרה.

toString
נראות: private
קלט: RBNode n
פלט: String
סיבוכיות: O(n)
המתודה מייצרת באופן רקורסיבי מחרוזת המייצגת את תת-העץ שמתחיל בצומת שניתנה כקלט. לאחר מכן היא מחזירה את המחרוזת שנוצרה.

isRed, right, left
נראות: private
קלט: RBNode n
פלט: RBNode עבור left, right; boolean עבור isRed
סיבוכיות: O(1)
שלושת המתודות הן מתודות עזר שנעשה בהן שימוש מיד פעם בקוד. תפקידן הוא למנוע מדי פעם, במידת הצורך, קריאה מפורשת לפעולות על הסנטינל- במקום nil.left, nil.right, nil.isRed הפעולות left(nil), right(nil), isRed(nil) מבטיחות מדי פעם בקוד שאנו מקבלים את התוצאה הרצויה.

מדידות

מספר ניסוי	מס׳ פעולות	מספר החלפות ממוצע להכנסה	מספר החלפות ממוצע למחיקה
1	10,000	2.337	0.9951
2	20,000	2.3219	0.9923
3	30,000	2.3115	0.9945
4	40,000	2.3145	0.9938
5	50,000	2.3116	0.9972
6	60,000	2.3197	0.993
7	70,000	2.3145	0.9966
8	80,000	2.3157	0.9941
9	90,000	2.3209	0.999
10	100,000	2.3195	0.995

המדידות שעשינו בעצם מודדות סיבוכיות ריצה amortized של הפעולות- הן מודדות ממוצע על פני סדרה של פעולות (ההנחה היא שהסדרה הזו דומה לסדרה שהיא worst case). על פי מה שלמדנו בהרצאה על עץ אדום שחור, זמן ה-amortized של פעולות insert ו-delete הוא O(1). כלומר, הזמן הממוצע לפעולה לא צריך להיות תלוי בגודל הקלט- צריך להתקיים שזמן הריצה של סדרת n פעולות הוא O(n).

אכן, ניתן לראות בניסוי שהזמן הממוצע לא תלוי בגודל הקלט. הסטיות המתקבלות נראות רנדומליות וזניחות (החל מ2-3 ספרות לאחר הנקודה). מתקיים בפעולת הכנסה שעבור הקבוע c=1, מספר ההחלפות בסדרה של n פעולות חסום ע״י c*n=O(n). באותו אופן, עבור c=2.35, מספר ההחלפות בסדרה של n פעולות הכנסה חסום ע״י c*n=O(n) כמו כן. לכן תוצאות המדידות תואמות באופן מלא את התאוריה שנלמדה בכיתה.