מטלה מעשית במבני נתונים

AVLNode

תיאור המחלקה: המחלקה מייצגת צומת בעץ AVL כאשר לכל צומת נאתחל 2 בנים וירטואליים (שניתנים לשינוי) .

השדות של כל צומת יהיו :

key – המפתח של הצומת (מספר טבעי) .

info – ערך המפתח שיהיה מטיפוס בוליאני (שיכול להיות true \ false \null) .

right – מצביע לבן הימני .

left – מצביע לבן השמאלי .

parent – מצביע להורה של הצומת .

height – גובה הצומת .

successor – האיבר העוקב של הצומת .

predecessor – האיבר הקודם של הצומת .

bf – גובה תת עץ השמאלי פחות גובה תת העץ הימני.

xor – אם הצומת הינו עלה אז ה xor שלו יהיה שווה ל info שלו , אחרת יהיה ה Exclusive Or של הערכי ה info של הצמתים שנמצאים בתת העץ של הצומת (לא כולל הצומת עצמו) .

הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות : - AVLNode

AVLNode :

הסבר: בנאי עבור המחלקה בו אנו מאתחלים את ערך המפתח ,את ה info ואת ה xor עבור צומת .

סיבוכיות: אתחול שדות אלו נעשה על ידי פעולות קבועות שעולתן הינה : O(1) , לכן סה"כ סיבוכיות זמן הריצה : O(1).

VirtualNode :

הסבר: מתודה זו מאתחלת צומת וירטואלי בעל מפתח 1- (אשר מציין כי זהו צומת וירטואלי) ,ערך בוליאני שהינו null ואת הגובה להיות 1- .

סיבוכיות: בצענו קריאה לבנאי כאשר הוא מתבצע ב O(1), ובצענו אתחול לשדה הגובה שמתבצע ב O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

getKey :

הסבר: מתודה זו מחזירה את ערך המפתח של הצומת במידה והוא צומת אמיתי , אחרת זהו צומת וירטואלי ומחזירה 1- .

סיבוכיות: קריאה לפונקציה isRealNode שעלותה O(1) וגישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

getValue :

הסבר: מתודה זו מחזירה את הערך (info) של הצומת במידה והוא צומת אמיתי , אחרת זהו צומת וירטואלי ומחזירה null.

סיבוכיות: קריאה לפונקציה isRealNode שעלותה O(1) וגישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

setLeft :

הסבר: מתודה זו מבצעת השמה לבן השמאלי של הצומת.

סיבוכיות: שינוי שדה שעלותו O(1) , ולכן סה"כ סיבוכיות זמן הריצה הינו : O(1) .

getLeft :

הסבר: מתודה זו מחזירה את הבן השמאלי של הצומת.

סיבוכיות: גישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

setRight :

הסבר: מתודה זו מבצעת השמה לבן הימני של הצומת.

סיבוכיות: שינוי שדה שעלותו O(1) , ולכן סה"כ סיבוכיות זמן הריצה הינו : O(1) .

getRight :

הסבר: מתודה זו מחזירה את הבן הימני של הצומת.

סיבוכיות: גישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

setParent :

הסבר: מתודה זו מעדכנת את ההורה של הצומת .

סיבוכיות: בצענו השמה להורה של הצומת שזו פעולה קבועה ולכן עלותה O(1) , סה"כ סיבוכיות הריצה הינו : O(1) .

getParent :

הסבר: מתודה זו מחזירה את ההורה של הצומת .

סיבוכיות: בצענו קריאה לשדה של ההורה של הצומת שעלותו קבועה, ולכן סה"כ סיבוכיות הריצה הינו : O(1) .

isRealNode :

הסבר: מתודה זו מחזירה false אם הצומת הינו וירטואלי , אחרת הצומת הינו צומת אמיתי ותחזיר true .

סיבוכיות: גישה לשדה של צומת הינה פעולה קבועה שעלותה O(1), יתר הפעולות הינן פעולות קבועות ולכן סה"כ סיבוכיות הריצה הינה : O(1) .

setHeight:

הסבר: מתודה זו מעדכנת את הגובה של הצומת .

סיבוכיות: בצענו השמה שהינה פעולה קבועה ולכן מתבצעת ב O(1) , לכן סה"כ סיבוכיות הריצה הינו : O(1) .

Is_leaf:

הסבר: מתודה זו מחזירה true אם הצומת הינו עלה, אחרת הצומת אינו עלה ותחזיר false .

סיבוכיות: הפעולות הינן פעולות קבועות ולכן סה"כ סיבוכיות הריצה הינה : O(1) .

getHeight:

הסבר: מתודה זו מחזירה את הגובה של הצומת במידה והצומת הינו צומת אמיתי , אחרת אם הצומת וירטואלי תחזיר 1- .

סיבוכיות: קריאה לפונקציה isRealNode שעלותה O(1) וגישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

AVLTree

תיאור המחלקה: המחלקה מייצגת עץ AVL עם אברים בעלי מפתחות טבעיים וערכים בולאניים.

השדות של כל עץ יהיו :

root – מצביע לשורש העץ.

min – מצביע לאיבר המינימלי בעץ.

max – מצביע לאיבר המקסימלי בעץ.

size – מספר טבעי שמייצג את כמות האברים בעץ.

cnt_balance_delete – משתנה גלובלי שסופר את מספר הרוטציות שמתבצעות במהלך מחיקה של איבר בעץ, מאותחל לאפס בתחילת כל ביצוע מחיקה.

הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות : - AVLTree

AVLTree:

הסבר: בנאי למחלקה היוצר עץ AVL ריק כאשר השורש מאותחל ל null .

סיבוכיות: בצענו אתחול של שדה ולכן סה"כ סיבוכיות הריצה הינה : O(1) .

emtpy:

הסבר: בודק אם העץ ריק.

סיבוכיות: הפעולות הינן פעולות קבועות ולכן סה"כ סיבוכיות הריצה הינה : O(1) .

Search (int k):

הסבר: מקבלת מספר טבעי ובודקת האם קיים צומת עם המפתח הנ"ל. אם קיים מחזירה את ערכו (info) , אחרת מחזירה null. בצענו חיפוש על ידי השוואת המפתח של הצומת עליו אנו עומדים לבין k . אם מצאנו את k אז החזרנו את ערך ה info שלו , אם K גדול נרד לבן הימני , אם k קטן נרד לבן השמאלי. אחרת נחזיר null.

סיבוכיות: בצענו חיפוש כפי שנלמד בכיתה על עץ AVL כאשר בעת החיפוש אנו נרד כגובה העץ (במקרה הגרוע) אשר גובהו הוא O(logn), ולכן סיבוכיות זמן הריצה הינה O(logn). (n מייצג את מספר האברים שנמצאים בעץ)

Insert (int k, Boolean i):

הסבר: הפונקציה מקבלת מספר טבעי וערך בוליאני. נבדוק בעזרת הפונקציה search האם קיים איבר בעץ בעל המפתח k אם קיים איבר בעל מפתח זה, אזי לא תתבצע הכנסה לעץ של האיבר עם המפתח k והערך i . אחרת, נכניס אותו לעץ בתור עלה במקום המתאים לו אותו נמצא על ידי ירידה באופן דומה לתהליך שבצענו בפוקנציה search . לאחר ההכנסה בצענו עדכונים לשדות successor ו- predecessor בעזרת הפונקציה set_succ_and_pred_insert. בנוסף, עלינו במעלה המסלול מהצומת שהכנסנו לשורש העץ ובצענו עדכון לשדות הגובה וה balance factor (bf) על ידי הפונקציה updeate_heights_and_bf . עלינו פעם נוספת במסלול זה עד שהגענו לצומת עם bf השווה ל 2 או 2- . אם הגענו לצומת כזה בצענו גלגול על ידי הפונקציה rotations. במידה ובצענו גלגול אז אין צורך להמשיך לעלות כי אין יותר עבריני AVL בעץ , כפי שנלמד בכיתה . בצענו פעם נוספת קריאה לפונקציה updeate_heights_and_bf כדי לעדכן את הגבהים של הצמתים במסלול לאחר הגלגול. לאחר מכן, בצענו עדכון לשדות הxor של הצמתים במסלול על ידי הפונקציה update_xor_for_ancestors . במידה ולא היה גלגול, נחזיר את הערך שהתקבל מהקריאה לפונקציה update_heights_and_bf. במידה והיה גלגול, נחזיר את אורך המסלול מהצומת שהוכנס עד לצומת עליו נעשה הגלגול.

סיבוכיות: השתמשנו בפונקציה search - O(logn) . לאחר מכן ירדנו בעץ למקום ההכנסה - O(logn) בדומה לתהליך שבצענו בפונקציה search . השתמשנו בפונקציות: set_succ_and_pred_insert – O(logn) , updeate_heights_and_bf - O(logn). בצענו עליה בעץ לצורך מציאת עבריין AVL בעץ - O(logn) הסבר ב - * . לאחר מכן , השתמשנו בפונקציה rotations – O(1) ולבסוף השתמשנו בפונקציות : updeate_heights_and_bf - O(logn) , update_xor_for_ancestors - O(logn) . יתר הפעולות מתבצעות בזמן קבוע ולכן מתבצעות בזמן O(1) . לכן סה"כ סיבוכיות זמן הריצה הינה : O(logn).

delete:

הסבר: הפונקציה מקבלת מפתח k ובודקת האם קיים איבר עם מפתח זה , אם כן אז תבצע מחיקה של איבר זה . אחרת , תחזיר 1-. נאתחל את המשתנה cnt_balance_delete לאפס. לאחר מכן נשתמש בפונקציה search על מנת לבדוק האם האיבר נמצא בעץ. אם האיבר נמצא בעץ אז תחילה נגיע אליו על ידי הפונקציה find_Node , נבצע עדכון של ה successor ו- predecessor בעזרת הפונקציה set_succ_and_pred_delete , לאחר מכן קראנו לפונציה choose_delete שתבחר איך לבצע מחיקה של הצומת הנ"ל כתלות במספר הבנים שלו ותמחק אותו . נקרא לפונקציה eliminate אשר "מנתקת" את הצומת מהעץ. לבסוף , נחזיר את cnt_balance_delete – שמסופר את מספר הרוטציות שהתבצעו לאחר המחיקה.

סיבוכיות: תחילה השתמשנו בפונקציה search - O(logn) . לאחר מכן , קראנו לפונקציה find_Node - O(logn) . בצענו קריאה לפונקציה set_succ_and_pred_delete – O(1) , וקריאה נוספת לפונקציה choose_delete – O(logn) . יתר הפעולות מתבצעות בזמן קבוע ולכן מתבצעות בזמן O(1) . לכן סה"כ סיבוכיות זמן הריצה הינה : O(logn).

min:

הסבר: במידה והעץ אינו ריק הפונקציה תחזיר את ערך ה info של האיבר בעל המפתח המינימלי בעץ על ידי קריאה לשדה המינימום, אחרת תחזיר null.

סיבוכיות: הפעולות שבצענו הינן פעולות קבועות ולכן סה"כ סיבוכיות הריצה הינה : O(1) .

max:

הסבר: במידה והעץ אינו ריק הפונקציה תחזיר את ערך ה info של האיבר בעל המפתח המקסימלי בעץ על ידי קריאה לשדה המקסימום, אחרת תחזיר null.

סיבוכיות: הפעולות שבצענו הינן פעולות קבועות ולכן סה"כ סיבוכיות הריצה הינה : O(1) .

keysToArray:

הסבר: במידה והעץ אינו ריק הפונקציה תחזיר מערך ממוין של מפתחות של אברי העץ שיקבע על פי גודל המפתחות שלהם מהקטן לגדול, אחרת העץ ריק ותחזיר מערך ריק. הפוקנציה תבצע זאת תחילה על ידי אתחול מערך כגודל העץ , ניגש לאיבר המינימלי נכניס את המפתח שלו למערך ובאותו אופן נמשיך זאת על ידי קריאה לsuccessor עד שנגיע לאיבר האחרון בגודלו בעץ.

סיבוכיות: בצענו קריאה ל successor והכנסה של מפתח למערך n פעמים כל קריאה כזו מתבצעת ב O(1) ולכן סה"כ O(n) . יתר הפעולות מתבצעות בזמן קבוע לכן עולות O(1). לפיכך, סה"כ סיבוכיות זמן הריצה, הינו: O(n) .

infoToArray:

הסבר: במידה והעץ אינו ריק הפונקציה תחזיר מערך של ערכי ה Info אשר ממוין לפי מפתחות אברי העץ לגדול, אחרת העץ ריק ותחזיר מערך ריק. הפוקנציה תבצע זאת תחילה על ידי אתחול מערך כגודל העץ , ניגש לאיבר המינימלי נכניס את ערך ה Info שלו למערך ובאותו אופן נמשיך זאת על ידי קריאה לsuccessor עד שנגיע לאיבר האחרון בגודלו בעץ.

סיבוכיות: בצענו קריאה ל successor והכנסה של ערך info למערך n פעמים כל קריאה כזו מתבצעת ב O(1) ולכן סה"כ O(n) . יתר הפעולות מתבצעות בזמן קבוע לכן עולות O(1). לפיכך, סה"כ סיבוכיות זמן הריצה, הינו: O(n) .

size:

הסבר: הפונקציה תחזיר את מספר האברים בעץ על ידי קריאה לשדה ה size.

סיבוכיות: גישה לשדה אשר מתבצעת בזמן קבוע, לכן סה"כ סיבוכיות זמן הריצה, הינה: O(1) .

getRoot:

הסבר: הפונקציה תחזיר את השורש של העץ על ידי קריאה לשדה ה root .

סיבוכיות: גישה לשדה אשר מתבצעת בזמן קבוע, לכן סה"כ סיבוכיות זמן הריצה, הינה: O(1) .

Successor (AVLNode node) :

הסבר: הפונקציה מקבלת איבר בעץ ומחזירה את האיבר העוקב שלו במידה וקיים , אחרת לא קיים לא איבר עוקב ותחזיר null . הפונקציה תמצא זאת על ידי קריאה לשדה ה successor .

סיבוכיות: גישה לשדה אשר מתבצעת בזמן קבוע, לכן סה"כ סיבוכיות זמן הריצה, הינה: O(1) .

PrefixXor (int k) :

הסבר: הפונקציה מקבלת מפתח k אשר שייך לאיבר שנמצא בעץ, ומחזירה את ה Xor של ערכי ה info של האברים הקיימים בעץ אשר הם בעלי מפתחות שקטנים או שווים ל k . נבצע זאת באופן הבא: תחילה נאתחל cnt לאפס. לאחר מכן, נעלה במעלה העץ על ידי קריאה לשדה ה parent עד שנגיע לשורש העץ ועבור כל צומת נבדוק האם הפונקציה xor_addition על צומת זה מחזירה true וגם המפתח של צומת זה קטן או שווה ל k אם כן נגדיר את ערכו של cnt ב- 1 . נעיר כי מכל צומת המקיים את התנאים הללו יוחזר לנו מספר ערכי ה Info true מודולו 2 של אברי העץ בעלי המפתחות שקטנים או שווים לו . לכן סה"כ נקבל את סכום זה על פני כל האברים בעלי מפתחות שקטנים או שווים ל k אשר ערך ה info שלהם הינו true ולכן כאשר בסוף נבדוק האם הערך cnt זוגי או אי זוגי יוחזר לנו ערך ה xor של ערכי ה Info על פני כל האברים שערך המפתח שלהם קטן או שווה ל k . לבסוף, נבדוק אם cnt אי זוגי או זוגי . אם הוא אי זוגי נחזיר true ,אחרת נחזיר false.

סיבוכיות: בצענו מעבר על אברי העץ שנמצאים על המסלול של מהצומת ועד לשורש העץ כאשר השתמשנו בפונקציה xor_addition שעולה O(1) ולפי הערה * מסלול זה ידרוש יעלה לנו O(logn) . יתר הפעולות הינן מתבצעות בזמן קבוע, ולכן עולות O(1) . לכן, סה"כ סיבוכיות זמן הריצה הינו : O(logn) .

SuccPrefixXor(int k):

הסבר: הפונקציה מקבלת מפתח k אשר שייך לאיבר שנמצא בעץ, ומחזירה את ה Xor של ערכי ה info של האברים הקיימים בעץ אשר הם בעלי מפתחות שקטנים או שווים ל k . בצענו זאת באופן הבא: תחילה נגשנו לאיבר המינימלי בעץ על ידי קריאה לשדה המינימום , ובנוסף אתחלנו שדה cnt שסופר את מספר האברים שהמפתח שלהם הינו קטן או שווה ל k ובנוסף ערך ה info שלהם הינו true . בצענו לולאה בה כל עוד הגענו לאיבר שקיים בעץ אשר בעל מפתח שקטן או שווה מ k אז בצענו נגשנו לשדה ה info של איבר זה. אם ערך זה היה true אזי הגדלנו את cnt באחד. לאחר מכן התקדמנו לאיבר הבא על ידי קריאה לשדה ה successor של איבר עליו אנו נמצאים. לבסוף , נבדוק אם cnt אי זוגי או זוגי . אם הוא אי זוגי נחזיר true ,אחרת נחזיר false.

סיבוכיות: עברנו על אברי העץ החל מהאיבר המינימלי ועד לאיבר שהמפתח שלו קטן או שווה מ k על ידי קריאה לשדה ה successor של אותו האיבר. לכל היותר נצטרך לבצע n קריאות כאלו במידה והאיבר הינו בעל המפתח המקסימלי . יתר הפעולות הינן מתבצעות בזמן קבוע, ולכן עולות O(1) . לכן, סה"כ סיבוכיות זמן הריצה הינו : O(n) .

Update_xor_for_ancestors (AVLNode node):

הסבר: הפונקציה מקבלת צומת בעץ. עבור כל צומת במסלול בין הצומת הנתונה לשורש, נעדכן את שדה ה-xor של הצומת בעזרת הפונקציה update_xor_for_Node.

סיבוכיות: עלייה במעלה העץ – log(n) (הסבר בהערה *). בכל רמה של העץ קוראים לפונקציה update_xor_for_Node שסיבוכיות הזמן שלה היא O(1). לכן הסיבוכיות היא O(log(n)).

update_xor_for_Node(AVLNode Node):

הסבר: הפונקציה מקבלת צומת בעץ, ומעדכנת את ערך ה-xor שלה בעזרת הבנים שלה. נשים לב שמספר ה true בתת העץ של הצומת הנתונה שווה למספר הtrue בתת העץ של הבן הימני של הצומת ועוד מספר ה-true בתת העץ של הבן השמאלי של הצומת. לכן על מנת לקבל את ערך שדה ה-xor המעודכן נשתמש בערכי הxor של הבנים של הצומת. במידה והבנים אינם עלים יש צורך להוסיף את ערך ה-info שלהם לשיקלול (שכן השדה xor אינו מכיל את ערכים אלו במידה והצמתים אינם עלים). לכן נספור את מספר הtrue בשדות ה-xor של הבנים של הצומת הנתון, ואם הם אינם עלים נוסיף לספירה גם את כמות ערכי ה true שמופיעים בinfo שלהם. לבסוף נוסיף לספירה 1 אם בערך ה-info של הצומת עצמו יש true. נעדכן את השדה xor של הצומת הנתון להיות false במידה והספירה זוגית, אחרת true.

סיבוכיות: כל הפעולות שביצענו (בפרט גישה לשדות, ושימוש בפונקציות is_leaf ו-isRealNode) מתבצעות ב O(1) לכן הסיבוכיות היא O(1)

Set_succ_and_pred_insert(AVLNode Node)

הסבר: הפונקציה מקבלת צומת בעץ לאחר הכנסתה. במידה והמפתח של הצומת גדול מהמפתח המקסימאלי בעץ – לצומת לא יהיה successor, וה-predecessor שלה יהיה ההורה שלה. במידה והמפתח של הצומת קטן מהמפתח המינימאלי בעץ – לצומת לא יהיה predecessor וה-successor שלה יהיה ההורה שלה. אחרת – על מנת למצוא את הsuccessor שלה – נעלה שמאלה במעלה העץ ככל הניתן, ונגדיר את הsuccessor להיות הצומת לאחר הפנייה הראשונה ימינה. בהכרח קיימת צומת עם מפתח גדול יותר – אחרת המפתח של הצומת הנתון היה המקסימאלי – לכן בהכרח תתבצע פנייה ימינה. נעדכן את הקודם של הצומת בתור הקודם של העוקב, את העוקב של הקודם בתור הצומת, ואת הקודם של העוקב בתור הצומת.

סיבוכיות: עלייה במעלה העץ בסיבוכיות O(log(n)) (לפי הערה*), וכל שאר הפעולות שביצענו (בפרט גישות לשדה) מתבצעות בזמן קבוע. לכן הסיבוכיות היא O(log(n))

Rotations(AVLNode Node):

הסבר: הפונקציה מקבל עבריין AVL ובוחרת איזה גלגול/גלגולים נחוצים בהתאם ל-BF שלו ושל בניו כפי שראינו בשיעור. הגלגולים עצמם מתבצעים על ידי קריאה לפונקציה right_rotation, לפונקציה left_rotation, או לשתיהן.

סיבוכיות: הפונקציות right_rotation, left_rotation מתבצעות בזמן קבוע, ויתר הפעולות (בפרט, גישה לשדות) גם הן פעולות שמתבצעות בזמן קבוע. לכן הסיבוכיות היא O(1).

Update_heights_and_BF(AVLNode Node):

הסבר: הפונקציה מקבלת צומת ומעדכנת את הגובה והBF של האיברים במסלול מהצומת לשורש. העלייה מתבצעת באמצעות קריאה לשדה parent של כל צומת. הגובה של כל צומת במסלול מוגדר להיות המקסימום בין הגבהים של הבנים שלו ועוד 1. הbf של כל צומת מוגדר להיות הגובה של הבן השמאלי שלו פחות הגובה של הבן הימני שלו. הפונקציה מחזירה את מספר הצמתים ששינו את גובהם לאחר ההכנסה.

סיבוכיות: עלייה במעלה העץ מתבצעת בסיבוכיות O(log(n)) לפי הערה *. בכל רמה אנו מבצעים פעולות שמתבצעות בזמן קבוע כגון גישה לשדות הבנים, עדכון שדות וכו'. לכן הסיבוכיות היא O(log(n))

right_rotation (AVLNode Node):

הסבר: הפונקציה מקבלת צומת עליה נרצה לבצע גלגול ימינה. הפונקציה מבצעת גלגול כפי שנלמד בשיעור. נעדכן את שדות הגובה של הצומת ובנה השמאלי (עלולים להשתנות במהלך הגלגול). נעדכן את השדה BF של הצומת. נעדכן את שדה ה xor של הצומת, ושל ההורה שלו על ידי הפונקציה update_xor_for_Node.

סיבוכיות: הגלגול מתבצע על-ידי שינוי מצביעים (שחסום למשל על-ידי 10) ולכן לוקח זמן קבוע. עדכון שדות הגובה וה- BF מתבצע על ידי גישה לשדות בזמן קבוע. שימוש בפונקציה update_xor_for_Node לוקח זמן קבוע. לכן הסיבוכיות היא O(1).

left_rotation (AVLNode Node):

הסבר: הפונקציה מקבלת צומת עליה נרצה לבצע גלגול שמאלה הפונקציה מבצעת גלגול כפי שנלמד בשיעור. נעדכן את שדות הגובה של הצומת ובנה הימני (עלולים להשתנות במהלך הגלגול). נעדכן את השדה BF של הצומת. נעדכן את שדה ה xor של הצומת, ושל ההורה שלו על ידי הפונקציה update_xor_for_Node.

סיבוכיות: הגלגול מתבצע על-ידי שינוי מצביעים (שחסום למשל על-ידי 10) ולכן לוקח זמן קבוע. עדכון שדות הגובה וה- BF מתבצע על ידי גישה לשדות בזמן קבוע. שימוש בפונקציה update_xor_for_Node לוקח זמן קבוע. לכן הסיבוכיות היא O(1).

Update_heights_and_bf_2(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנמחק מהעץ. הפונקציה מעדכנת את הגבהים וה-BF של הצמתים במסלול מהצומת שנמחק לשורש. נעדכן את הגובה של ההורה של הצומת שנמחק להיות המקסימום מבין הגבהים של הבנים החדשים שלה ועוד 1. נעדכן את ה bf ההורה של הצומת שנמחק להיות הגובה של הבן השמאלי שלו פחות הגובה של הבן הימני שלו (לאחר המחיקה). נעדכן את הגבהים והBF של כל צומת במסלול מההורה של הצומת שנמחק עד לשורש. עדכון הגובה יתבצע על ידי מציאת המקסימום בין הגובה של הבן הימני של כל צומת לבן השמאלי שלו, והוספת אחד. עדכון הbf יתבצע באופן זהה לעדכון שנעשה בהורה של הצומת. בכל פעם שנעדכן את הגובה של צומת, נבדוק אם גובה הצומת לאחר העדכון זהה לגובה הצומת לפני העדכון, במידה והוא שונה – נוסיף למשתנה cnt_balance_delete 1.

סיבוכיות: עלייה במעלה העץ לוקחת O(log(n)) (לפי הערה*), בכל רמה אנו מבצעים פעולות שמתבצעות בזמן קבוע (גישה לשדות, עדכון שדות...). לכן הסיבוכיות היא O(log(n)).

Choose_delete(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנרצה למחוק מהעץ, בודקת כמה בנים יש לו, ולפי מספר הבנים מחליטה איזה מחיקה יש לבצע.

במידה ולצומת יש שני בנים, נגדיל את השדה size ב-1, כיוון שבמקרה זה אנו קוראים לפונקציה delete (שמורידה 1 מ-size) פעמיים, למרות שמתבצעת רק מחיקה אחת.

סיבוכיות: בדיקת מספר הבנים מתבצעת על ידי הפונקציות is_leaf ו-isRealNode שמתבצעות בזמן קבוע. פונקציה זו קוראת לאחת מהפונקציות choose_delete_0, choose_delete_1, choose_delete_2 שהסיבוכיות שלהן היא O(log(n)) . לכן הסיבוכיות היא O(log(n)).

Delete_0(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנרצה למחוק בעל 0 בנים. הפונקציה מוחקת את הצומת ומעדכנת את הבן של ההורה של הצומת שנמחק להיות צומת וירטואלי. נבצע עדכון של הגבהים והBF של הצמתים במסלול מההורה של הצומת שנמחק עד לשורש על ידי קריאה לפונקציה update_heights_and_BF_2. נקרא לפונקציה rotations_after_delete על מנת לבצע את הגלגולים הנדרשים על הצמתים שנמצאים במסלול בין ההורה של הצומת שנמחק עד לשורש.

סיבוכיות: הפונקציות rotations_after_delete ו-update_heights_and_bf_2 מתבצעות בסיבוכיות O(log(n)). יתר הפעולות מתבצעות בסיבוכיות קבועה. לכן הסיבוכיות היא O(log(n)).

Delete_1(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנרצה למחוק בעל בן אחד. נעדכן את הבן של ההורה של הצומת שנמחק להיות הבן של הצומת שנמחק. נבצע עדכון של הגבהים והBF של הצמתים במסלול מההורה של הצומת שנמחק עד לשורש על ידי קריאה לפונקציה update_heights_and_BF_2. נקרא לפונקציה rotations_after_delete על מנת לבצע את הגלגולים הנדרשים על הצמתים שנמצאים במסלול בין ההורה של הצומת שנמחק עד לשורש.

סיבוכיות: הפונקציות rotations_after_delete ו-update_heights_and_bf_2 מתבצעות בסיבוכיות O(log(n)). יתר הפעולות מתבצעות בסיבוכיות קבועה. לכן הסיבוכיות היא O(log(n)).

Delete_2(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנרצה למחוק בעל שני בנים. נקרא לפונקציה delete עבור העוקב של הצומת שנרצה למחוק. נעביר את העוקב של הצומת שנרצה למחוק למקום בו נמצא הצומת שאנו רוצים למחוק, על ידי עדכון השדות המתאימים.

הערה: אנו מגיעים לפונקציה זו דרך הפונקציה choose_delete אליה מגיעים דרך הפונקציה delete, ובתוך פונקציה זו אנו קוראים לפונקציה delete, לכן כביכול עלולה להיווצר רקורסיה אינסופית. הסיבה שלא נגיע למצב כזה, ושלא נגיע לאחר מכן לפונקציה delete_2 פעם נוספת, היא שלעוקב של הצומת שנרצה למחוק יש בהכרח בן אחד לכל היותר (הראנו בשיעור), לכן הפונקציה choose_delete_2 לא תיקרא פעם נוספת.

סיבוכיות:

מבצעים קריאה ל-delete עבור העוקב של הצומת שנרצה למחוק – פונקציה זו מתבצעת בסיבוכיות (O(log(n)). שאר הפעולות (גישה לשדות, עדכון שדות וכו'...) מתבצעות בזמן קבוע. לכן הסיבוכיות היא (O(log(n)).

Rotations_after_delete(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנרצה למחוק מהעץ. נעלה במסלול מהצומת שנרצה למחוק עד לשורש. עבור כל צומת במסלול נעדכן את הגובה, והBF שלו ושל ההורה שלו, ואת ה-xor שלו ושל ההורה שלו בעזרת קריאה לפונקציה update_xor_for_Node. במידה והצומת הוא גם עבריין AVL נקרא לפונקציה rotations על מנת לבצע את הגלגול הדרוש. עבור כל צומת ששינה את גובהו לאחר הגלגול נגדיל את המשתנה cnt_balance_delete ב-1.

סיבוכיות: עלייה במסלול מהצומת שנרצה למחוק עד לשורש עולה (O(log(n) לפי הערה *. סיבוכיות עדכון הגובה, הbf וה-XOR (באמצעות הפונקציה update_xor_for_Node שעלותה (O(1) עבור כל צומת במסלול היא (O(1. סיבוכיות קריאה לrotations במקרה והצומת הוא גם עבריין AVL היא (O(1. לכן הסיבוכיות היא (O(log(n)).

eliminate(AVLNode Node):

הסבר: הפונקציה מקבלת צומת אותו נרצה לנתק מהעץ. נשנה את שדות הבנים וההורה שלו לnull.

סיבוכיות: שינוי השדות מתבצע בזמן קבוע, לכן הסיבוכיות היא (O(1

Set_succ_and_pred_delete(AVLNode Node):

הסבר: הפונקציה מקבלת צומת שנרצה למחוק. אם המפתח של הצומת הוא המינימאלי בעץ – נעדכן את שדה ה-min להיות העוקב של הצומת שנרצה למחוק, ואת הקודם של העוקב של הצומת שנרצה למחוק ל-null. אם הצומת הוא המקסימאלי בעץ – נעדכן את שדה המקסימום להיות הקודם שלו, ואת העוקב של הקודם שלו להיות null. אחרת, נעדכן את העוקב של הקודם של הצומת שנרצה למחוק להיות העוקב של הצומת שנרצה למחוק, ואת הקודם של העוקב שלו להיות הקודם שלו.

סיבוכיות: ביצענו פעולות (גישה לשדות, עדכון שדות וכו'...) שמתבצעות בזמן קבוע. לכן הסיבוכיות היא (O(1.

xor_addition(AVLNode Node):

הסבר: הפונקציה מקבלת צומת ומחשבת את ה-xor המשותף שלו ושל תת העץ השמאלי שלו. נאתחל משתנה cnt ל-0. נבדוק את הxor של תת העץ השמאלי של הצומת על ידי גישה לשדה xor של הבן השמאלי של הצומת – אם הוא true נגדיל את cnt ב-1. אם הבן השמאלי אינו עלה, שדה הxor שלו אינו מתייחס לinfo שלו, ולכן יש צורך לשקלל גם אותו – ולכן אם הוא true נגדיל את cnt ב-1. בנוסף, נרצה להתייחס גם לערך info של הצומת עצמו, לכן אם הוא true נגדיל את cnt ב-1. נחזיר true אם cnt אי זוגי, אחרת false.

סיבוכיות: ביצענו פעולות (גישה לשדות, עדכון שדות וכו'...) שמתבצעות בזמן קבוע. לכן הסיבוכיות היא O(1).

Find_Node(int k):

הסבר: הפונקציה מקבלת מפתח של איבר שנמצא בעץ, ומחזירה איבר בעץ בעל מפתח זה. בדומה לחיפוש, נמצא את האיבר עם המפתח k. נחזיר איבר זה.

סיבוכיות: ירידה בעץ החל מהשורש עולה (O(log(n לפי הערה *. בכל רמה בעץ מבצעים השוואה אחת שלוקחת O(1), לכן הסיבוכיות היא (O(log(n.

הערה * : ירידה ועליה בעץ AVL מתבצעות בזמן (O(logn מכיוון שגובהו של עץ AVL הינו (O(logn כאשר n מבטא את מספר האברים הקיימים בעץ .

הערה**: הכוונה בגישה לשדות שמתבצעת בזמן קבוע הייתה עבור מספר שדות שאינו תלוי במספר האיברים במבנה.

מדידות :

1. ניסוי 1

מספר סידורי	עלות prefixXor ממוצעת (כל הקריאות)	עלות succPrefixXor ממוצעת (כל הקריאות)	עלות prefixXor ממוצעת (100 קריאות ראשונות)	עלות succPrefixXor ממוצעת (100 קריאות ראשונות)
1	1065.2 ננו שניות	2688.2 ננו שניות	678 ננו שניות	2513 ננו שניות
2	711.8 ננו שניות	3977.6 ננו שניות	786 ננו שניות	2518 ננו שניות
3	492.666667 ננו שניות	4874 ננו שניות	765 ננו שניות	2373 ננו שניות
4	419.65 ננו שניות	6192.2 ננו שניות	747 ננו שניות	2303 ננו שניות
5	400.84 ננו שניות	8542.2 ננו שניות	809 ננו שניות	2283 ננו שניות

מסקנות הטבלה :

- ניתן לראות כי עבור prefixXor ככל שכמות האברים בעץ גדולה יותר , ממוצע זמן הריצה על פני כל האברים בעץ קטן יותר. תוצאה זו אינה מתיישבת עם זמן הריצה אותו ציפינו לקבל וזאת מכיוון שפונקציה זו רצה בזמן לוגריתמי כתלות במספר האברים בעץ.
- מתוצאות זמני הריצה של succPrefixXor ניתן להסיק כי עלות זמן הריצה הממוצעת על פני כל אברי העץ עולה בערך בקצב לנארי כתלות במספר אברי העץ , כפי שציפינו לקבל מניתוח שבצענו לפונקציה זו.
- ניתן לראות כי עבור 100 הקריאות הראשונות שבצענו על הפונקציה PrefixXor ניתן לראות שזמני הריצה לא משתנים באופן משמעותי, וזאת בניגוד לצפייה שלנו שמספר האברים בעץ גדל אז כל קריאה לפונקציה תעלה לנו יותר זמן (לאור הניתוח של זמן הריצה של הפונקציה שהינו לוגריתמי).
- מתוצאות זמני הריצה של succPrefixXor על פני 100 הקריאות הראשונות ניתן לראות כי זמני הריצה יורדים (אך לא באופן משמעותי) ככל שמספר האברים בעץ גדל . תוצאה זו אינה מתיישבת עם צפי שלנו לקבל זמן ריצה שגדל בקצב לינארי כתלות במספר אברי העץ.
- כפי שציפינו לקבל ניתן לראות מתוצאות הניסוי שבצענו כי זמני הריצה של הפונקציה prefixXor נמוכים מזמני הריצה של הפונקציה succPrefixXor , כלומר הראשונה יעילה יותר. נוסיף כי ציפינו לקבל הפרשים גדולים יותר בין זמני הריצה של הפונקציות שנבע מכך שהראשונה רצה שזמן O(logn) לעומת השנייה שרצה בזמן O(n).

2. **ניסוי 2**

עלות הכנסה ממוצעת מספר סידורי i	עץ AVL סדרה חשבונית	עץ ללא מנגנון איזון סדרה חשבונית	עץ AVL סדרה מאוזנת	עץ ללא מנגנון איזון סדרה מאוזנת	עץ AVL סדרה אקראית	עץ ללא מנגנון איזון סדרה אקראית
1	4134.7 ננו שניות	8012.2 ננו שניות	1423 ננו שניות	4473 ננו שניות	3961.3 ננו שניות	1553.6 ננו שניות
2	2550.9 ננו שניות	8111.65 ננו שניות	1532 ננו שניות	5014 ננו שניות	2655.7 ננו שניות	901.1 ננו שניות
3	2003.8333 ננו שניות	8316.26667 ננו שניות	1461 ננו שניות	6775 ננו שניות	2176.7 ננו שניות	730.4333 ננו שניות
4	1785.425 ננו שניות	8911.125 ננו שניות	1418 ננו שניות	7486 ננו שניות	1962.4 ננו שניות	639.675 ננו שניות
5	1657.5 ננו שניות	10085.8 ננו שניות	1446 ננו שניות	8636 ננו שניות	1784.16 ננו שניות	559.58 ננו שניות

מסקנות הטבלה :

הכנסה של סידרה חשבונית:

- ניתן לראות כי עבור הכנסה לעץ AVL ככל שכמות האברים בעץ גדולה יותר , ממוצע זמן הריצה על פני כל ההכנסות לעץ AVL קטנה. תוצאה זו אינה מתיישבת עם זמן הריצה אותו ציפינו לקבל וזאת מכיוון שהפונקציה Insert רצה בזמן לוגריתמי כתלות במספר האברים בעץ.
- מתוצאות זמני הריצה של הכנסה של סידרה חשבונית לעץ ללא מנגנון איזון ניתן לראות כפי שציפינו לקבל כי זמני הריצה אכן עולים. עם זאת, נציין כי ציפינו לקבל הפרשים גדולים יותר פר הכנסה ככל שמספר האברים בעץ גדל וזאת מכיוון שהכנסה זו עולה O(n) .
- ניתן לראות מתוצאות זמני הריצה של ההכנסה של סידרה חשבונית עבור העץ AVL והעץ ללא מנגנון איזון כי הכנסה לעץ AVL מתבצעת בזמן ריצה נמוך יותר מאשר הכנסה לעץ ללא מנגנון איזון וזאת כפי שציפינו .

הכנסה של סידרה מאוזנת:

- ניתן לראות כי זמני הריצה של הכנסה לעץ AVL של סידרה מאוזנת לא משתנים באופן משמעותי ככל שמספר אברי העץ גדל, וזאת בניגוד למה שציפינו לקבל מאחר וככל שמספר האברים של העץ גדל אז כל הכנסה אמורה להתבצע בזמן ארוך יותר .
- מתוצאות זמני הריצה של הכנסה לעץ ללא מנגנון איזון ניתן לראות כי זמני הריצה גדלים כפי שציפינו לקבל מאחר וככל שמספר האברים של העץ גדל אז כל הכנסה אמורה להתבצע בזמן ארוך יותר .
- מהשוואת זמני הריצה של ההכנסה בין העץ AVL לעץ ללא מנגנון איזון ניתן להסיק כי זמן ההכנסה הממוצע לעץ ללא מנגנון איזון גדול באופן ניכר מזמן ההכנסה לעץ AVL . זאת בניגוד לצפי שלנו לכך שזמני ההכנסה לעצים הנ"ל יהיו פחות או יותר זהים כיוון שלא נצטרך לבצע גלגולים וכן אופן הבנייה של העצים זהה.

הכנסה של סידרה אקראית:

- ניתן לראות כי עבור הכנסה לעץ AVL ככל שכמות האברים בעץ גדולה יותר , ממוצע זמן הריצה על פני כל ההכנסות לעץ AVL קטנה. תוצאה זו אינה מתיישבת עם זמן הריצה אותו ציפינו לקבל וזאת מכיוון שהפונקציה Insert רצה בזמן לוגריתמי כתלות במספר האברים בעץ.
- ניתן לראות כי עבור הכנסה לעץ ללא מנגנון איזון ככל שכמות האברים בעץ גדולה יותר , ממוצע זמן הריצה על פני כל ההכנסות לעץ AVL קטנה. תוצאה זו אינה מתיישבת עם זמן הריצה אותו ציפינו לקבל וזאת מכיוון שהפונקציה Insert רצה בזמן לינארי במקרה הגרוע כתלות במספר האברים בעץ.
- מהשוואת זמני הריצה של ההכנסה בין העץ AVL לעץ ללא מנגנון איזון ניתן לראות כי הפרשי זמן ההכנסה הממוצע בין 2 העצים הללו קטן במעט ככל שמספר האברים בעץ גדל.

מטלה מעשית 1 –

שי גולדבורט – 207859760 - goldbourt

אורן שפיגלר – 318638152 - orenshpigler