מבני נתונים 1

234218

תרגיל יבש

2

מספר

הוגש ע"י:

חגי קריטי	301781613
שם	מספר זהות
תם נלסון	204805824
שם	מספר זהות

<u>ציון</u>:

לפני בונוס הדפסה:

כולל בונוס הדפסה:

נא להחזיר לתא מס':

13

שאלה 1

א. מבנה הנתונים הוא עץ 2-3 עם מצביעים next ו-previous בעלים.

Init()

מאתחלים עץ עם עלה אחד עבור קומה 0 ומצביע current לעלה זה.
עץ עם עלה אחד - O(1)
אתחול מצביע - O(1)
סה"כ O(1)

AddStop(k)

מוסיפים את הצומת k לעץ, לאחר מכן מעדכנים את מצביעי next ו-previous של השכן משמאל ומימין (בהתאמה) וכן את מצביעי next,previous של הצומת החדש.

נמצא את אחד השכנים באופן הבא: במקרה שהעץ הוא לא עלה בודד, לצומת האב של הצומת k (לפני הפיצול שאולי יקרה בעקבות ההוספה של k) יהיו עוד בנים אחרים (אחרת צומת זו היא עלה והעץ לא מאוזן אחרי ההוספה). נבחר את הצומת השכן משמאל, ואם הוא לא קיים נבחר את השכן מימין. אם העץ הוא עלה בודד, השכן משמאל הוא העלה הבודד הזה (שכן הוא העלה של קומה 0, הקומה הנמוכה ביותר).

נשתמש בצומת השכן (בה"כ משמאל) ונעדכן את המצביע next שלו ל-next=k. את השכן מימין נמצא לפי next הישן של השכן משמאל ונעדכן בו את previous=k. את next, previous של הצומת k נעדכן לשכנים המתאימים.
הוספת צומת k לעץ: O(log(n))
מציאת השכן: O(1)
עדכון מצביעים: O(1)

סה"כ: O(log(n))

NextStop()

נשתמש במצביע current כדי להגיע לעלה של הקומה הנוכחית, ונקדם אותו לעלה הבא בעזרת מצביע ה-next שלו. לאחר מכן נדפיס את המפתח (הקומה) של העלה החדש.
מעבר עלה: O(1)
הדפסה: O(1)
סה"כ: O(1)

ב. נתון כי בAddStop הi ניתן להוסיף רק עד הקומה הi2 לכן נבנה את המבנה נתונים שלנו על בסיס מערך דינאמי אשר כפי שנלמד כל פעולה בו היא בסיבוכיות משוערכת של O(1).

במערך (floors), נשמור בתא הj 0 אם המעלית לא עוצרת בקומה הj או 1 אם המעלית עוצרת בתא הj.

בנוסף נשמור int שמייצג את הקומה הנוכחית של המעלית (current_floor).

Init():
איתחול מערך דינאמי – O(1).
השמת current_floor ל-0 – O(1).
סה"כ: O(1).

AddStop(k):
מוסיפים למערך 2 מקומות שהם 0
משנים את הערך של floors[k] = 1.

NextStop():
מעבירים את current_floor למקום הבא במערך floors אשר יש בו 1.

נראה כעת כי הסיבוכיות המשוערכת של AddStop(k) ו-NextStop() היא O(1).

נגדיר שפעולות AddStop(k) לוקחת O(4):
2*O(1) פעולות באופן משוערך על מנת להכניס 2 איברים למערך דינאמי.
O(1) פעולות כדי לשנות ערך במערך
ועוד O(1) שנשמר בצד לטובת פעולת NextStop() עתידית.
סה"כ הפעולה לוקחת O(1) משוערך.

עבור פעולת NextStop() אנחנו צריכים להתקדם במערך לערך הבא שהוא לא 0. בAddStop הi יש לנו מערך בגודל 2i (גם אם הוא לא באמת בגודל הנ"ל, זה הגודל שאנחנו יכולים לגשת אליו באותו רגע) אשר יש בו בדיוק i איברים מלאים. בנוסף יש לנו בצד i פעולות של O(1) ששמרנו בצד לשימוש עתידי. במקרה הגרוע של NextStop האיבר המלא הבא נמצא אחרי מספר מקסימלי של איברים ריקים – i צעדים. "נשלם" על צעדים אילו באמצעות הפעולות שלא השתמשנו בהם קודם וסה"כ נקבל שעשינו צעד אחד – הצעד האחרון מאיבר ריק לאיבר מלא. כלומר סה"כ עלות הפעולה היא O(1) משוערך.

<u>שאלה 2</u>

א

המבנה יכלול 2 עצי דרגות - אחד עבור ערך הX והשני עבור ערך הY של כל קטע. בנוסף נשמור את K.

בכל עץ הצמתים יכילו מפתח, משקל, דרגה לתת העץ השמאלי ודרגה לתת העץ הימני. המפתח יהיה ערך הקואורדינטה (X או Y) והמשקל יהיה מספר הקטעים שמתחילים או נגמרים בקואורדינטה זו. הדרגה של תת עץ היא מספר הצמתים בתת העץ, כאשר כל צומת נספר לפי המשקל שלו. העץ עבור X ממוין בסדר עולה והעץ עבור Y ממוין בסדר יורד.

מימוש הפעולות:

<u>Init(k):</u>

מאתחלים שני עצי דרגות ריקים ושומרים את K.

סיבוכיות: O(1)

<u>Insert(x, y):</u>

עבור קואורדינטת X, מחפשים את x בעץ. אם היא קיימת, מקדמים את המשקל של הצומת שלה ב-1. אם היא לא קיימת, מכניסים אותה לעץ עם משקל 1. לאחר מכן מעדכנים את הדרגה של כל הצמתים במסלול עד השורש.

באופן דומה מטפלים בקואורדינטת Y.

סיבוכיות:

חיפוש בעץ: O(logn) לכל קואורדינטה

הכנסה לעץ: O(log) לכל קואורדינטה

סה"כ: O(4*logn) = O(logn)

<u>Delete(x,y):</u>

עבור קואורדינטת X, מחפשים את x בעץ ומורידים את המשקל ב-1. אם המשקל של הצומת הגיע ל-0, מוציאים אותה מהעץ. לאחר מכן מעדכנים את הדרגות של הצמתים עד השורש.

באופן דומה מטפלים ב-Y.

סיבוכיות:

חיפוש בעץ: O(logn)

מחיקה מהעץ: O(logn)

סה"כ: O(logn)

<u>IsCentric(x,y)</u>

מחפשים כל קואורדינטה בעץ המתאים ומשתמשים בדרגות כדי לקבוע את המיקום שלה ברשימה ממוינת של כל הקואורדינטות. הסידור העולה של עץ X ייתן כמה קואורדינטות יש שקטנות\שוות ל-x (כמה קטעים מתחילים לפני או עם הקטע הנתון), והסידור היורד של עץ Y ייתן כמה קואורדינטות יש שגדולות\שוות ל-y (כמה קטעים מסתיימים אחרי או עם הקטע הנתון). אם שני המספרים האלה גדולים או שווים ל-K, נחזיר אמת.

סיבוכיות:

חיפוש בעץ: O(logn)

השוואה בין K לדירוג של כל קואורדינטה: O(1)

סה"כ: O(logn)

ב

נשמור את הקטעים משתי הקבוצות בעץ דרגות אחד, כאשר נשמור לכל צומת דרגה עבור הצמתים של קבוצה A ודרגה עבור צמתים של קבוצה B.

כל צומת בעץ יכיל את הערכים הבאים:

- מפתח שמורכב מקואורדינטת ה-X ומזהה הקבוצה (A או B), כאשר המיון הראשוני נעשה לפי הקואורדינטה ולמיון המשני נגדיר ש A>B.

- ערך הצומת, המכיל רשימה מקושרת של ערכי ה-Y של הקטעים שמתחילים בקואורדינטת X זו.
- משקל הצומת, המייצג את מספר הקטעים במערכת שמתחילים בקואורדינטת X זו (אורך הרשימה המקושרת).
- דרגת תת עץ השמאלי לצומת A, המכיל את את סכום המשקלים של הצמתים בתת העץ השמאלי מקבוצה A.
- דרגת תת עץ הימני לצומת B, המכיל את סכום המשקלים של הצמתים בתת העץ הימני מקבוצה B.

מימוש הפעולות:
<u>()Init</u>
מאתחלים עץ ריק.
סיבוכיות: O(1)

<u>Insert((x,y), G)</u>
מחפשים את המפתח (x,G) בעץ. אם הוא קיים, מקדמים את המשקל של הצומת ב-1 ומוסיפים את y לראש הרשימה המקושרת. אם הוא לא קיים, מכניסים צומת חדש עם משקל 1 ורשימה מקושרת שמכילה את y. לאחר עדכון הצומת, מקדמים את הדרגה של הקבוצה G (משמאל עבור A, מימין עבור B) בכל הצמתים במסלול עד השורש.

סיבוכיות:
חיפוש בעץ: O(log(n)
הכנסה לעץ: O(logn)
הכנסה לראש רשימה מקושרת: O(1)
עדכון הדרגות: O(logn)
סה"כ: O(logn)

<u>()StartBeforeInA=StartAfterInB</u>
נסמן ב rankABefore את מספר הקטעים בקבוצה A עם ערך X הקטן או שווה לערך ה-X של הצומת הנוכחי, וב rankBAfter את מספר הקטעים בקבוצה B עם ערך X הגדול או שווה לערך ה-X של הצומת הנוכחי. נראה איך מוצאים ערכים אלה בהמשך.

נבצע חיפוש בעץ, החל מהשורש, בצורה הבאה: אם rankABefore > rankBAfter נלך ימינה ואם rankABefore < rankBAfter נלך שמאלה. אם הגענו לסוף העץ מחזירים שגיאה. אם rankBAfter == rankABefore שולפים את הערך הראשון ברשימה המקושרת של הצומת הנוכחי לתוך משתנה y ומחזירים את (x,y).

כדי לחשב את rankABefore של צומת מסוים: לכל צומת בדרך, אם הולכים ממנו ימינה אז ל- rankABefore מתווספים דרגת תת העץ השמאלי של הצומת הנוכחי + משקל הצומת הנוכחי. אם הולכים שמאלה rankABefore לא משתנה. לבסוף כאשר מגיעים לצומת שאת דירוגו אנחנו רוצים לחשב, מוסיפים את המשקל שלו ל rankABefore. באופן דומה נחשב את rankBAfter, רק שהכיוונים מתהפכים. נשים לב שמכיוון שהגדנו במיון B>A, גם עבור קטעים שמתחילים באותה קואורדינטה בשתי הקבוצות ספירה זו עובדת: הצומת עבור הקטעים מ-B ימוקם ימינה מהצומת עבור הקטעים מ-A, ולכן rankABefore של הקטעים מ-B יספור את הקטעים מ-A ו rankBAfter של הקטעים מ-A יספור את הקטעים מ-B.

סיבוכיות:
חיפוש בעץ: O(logn)
שליפת ראש של רשימה מקושרת: O(1)
סה"כ: O(logn)

שאלה 3

א. עבור כל עץ נחפש את הגובה שלו (H1, H2) ע"י ספירת הצמתים שעוברים עד ההגעה לעלה הגדול ביותר: O(log(n1)+log(n2)).

עבור T1 נחפש את הערך המקסימלי בעץ: O(log(n1))

עבור T2 נחפש את הערך המינימלי בעץ: O(log(n2)).

נרכיב את העץ בגובה הנמוך על העץ הגבוהה יותר.

נניח כי T1 נמוך יותר (אם לא, זה מתבצע באופן זהה אבל ע"י חיפוש המקסימום של T2 בT1).

נחפש בT2 את הערך המקסימלי של T1, מובטח לנו שהחיפוש הנ"ל יוביל אותנו למסלול לערך הכי קטן בT2 ע"פ הנתון. הולכים על המסלול הנ"ל |H2-H1|-1 צמתים – כלומר עוצרים כשמגיעים לצומת שהגובה שלה זהה לגובה של T1. במקרה הגרוע הולכים O(|H2-H1)-1) = O(log(n1)) .

בנקודה זו מוסיפים לצומת ערך פיצול שהוא המקסימום של T1 ומחברים אליו את השורש של T1. O(1) פעולות.

במקרה שהעץ הנוצר הוא במבנה של עץ 2-3, סיימנו. אם לא העץ יעשה פיצולים ואיחודים על מנת לאזן את עצמו לאורך מסלול החיפוש (כמו בהוספה רגילה לעץ 2-3) – O(|H2-H1)-1) = O(log(n1))פעולות.

סה"כ עשינו O(log(n1)+log(n2)) = O(max{log(n1), log(n2)})

אם T1,T2 באותו גובה, יוצרים שורש מאוחד חדש לשניהם כאשר הערך המפריד בו הוא הערך המקסימלי של T1 – O(1) פעולות.

סה"כ O(log(n1)+log(n2))=O(max{log(n1),log(n2)})

כלומר בכל מקרה אנחנו עושים את הפעולה בO(max{log(n1), log(n2)})

ב. אם נסתכל על הנתונים כאן לעומת השאלה הקודמת, כעת אין צורך לעשות את החיפוש המקדים של הגבהים של העצים והערכים המקסימליים ומינימליים שלהם, אלא רק צריך לעשות את האיחוד של 2 העצים. נשים לב כי כבר בסעיף הקודם רואים כי הסיבוכיות של האיחוד בלבד היא O(|H2-H1| +1) משום שהולכים במקרה הגרוע על מסלול של O(|H2-H1|) צמתים ובסוף עושים O(1) פעולות לצורך האיחוד.

סה"כ: O(|H2-H1|+1)

ג. אם נסתכל על כל זוג T(i), T(i+1) אנחנו נמצאים במצב המתאים לאיחוד שתיארנו בסעיף ב. לכן הזמן שייקח לאחד את 2 העצים הוא O(H(i+1)-h(i) + 1). (כאשר אפשר להוריד את הערך המוחלט משום שנתון כי H(i+1)>=H(i)). לכן נתחיל לעשות את האיחודים של העצים החל מהעץ הקטן ביותר, באופן הבא:

תחילה לוקחים את 2 העצים הראשונים ומאחדים אותם. נקבל עץ שהוא או בגובה של H2 (H1 < H2) או עץ שהוא בגובה H2+1 (H1 = H2). ואז לוקחים את העץ המאוחד ומאחדים אותו עם העץ הבא בתור וכך הלאה באופן רקורסיבי עד שמאחדים את כל העצים.

נשים לב כי ההנחה שהעץ המאוחד עד כה (sum) תמיד קטן שווה לעץ הבא אחריו T(j+1) משום שמובטח שאין 3 עצים באותו גובה בסדרה המקורית, כלומר גם אם 2 העצים האחרונים היו באותו גובה ולכן H(sum) =H(j)+1 מובטח כי H(j+1) > H(j) כלומר H(j+1) >= H(j) + 1 = H(sum).

אם נסכום את כל עלויות הפעולות של האיחודים השונים נקבל:

$$\Sigma_{n=1}^{k-1} O(h_{n+1} - h_n + 1) = O\left(\Sigma_{n=1}^{k-1} h_{n+1} - h_n + 1\right) = O(h_k - h_1 + k)$$

ד. נעבור על העץ T ונפצל אותו ל-2 סדרות של תתי-עצים, סדרה אחת שבה כל האיברים בכל תת-עץ גדולים מ-X (big), וסדרה שנייה שבה כל האיברים בכל תת-עץ קטנים מ-X (small).

תחילה מחפשים את הגובה של העץ: O(logn) פעולות.

דרך ההפרדה:

עבור כל צומת בעץ החל מהשורש נעשה תהליך זהה: הצומת הבא שנטפל בה תהיה הצומת שהיינו הולכים אליה כדי לחפש את X בעץ. כל האחים מימין של הצומת הבאה מכילים רק איברים שגדולים מX ולכן ישתייכו לקבוצה big, כאשר עבור כל אחד כזה אנחנו יודעים חסם תחתון על הערכים בעץ לפי המפתח בשורש. כל האחים משמאל של הצומת הבאה מכילים רק איברים שקטנים מX ולכן ישתייכו לקבוצה small , כאשר עבור כל אחד כזה אנחנו יודעים חסם עליון על הערכים בתת-עץ לפי המפתח בשורש. עבור כל תת-עץ שמוסיפים את לקבוצה שומרים גם את הגובה שלו. עבור עלה, ניקח את הערך המקסימלי\מינימלי להיות ערך העלה.

נשים לב, כי בכל צעד למטה בעץ אנחנו מוסיפים בין 2-0 עצים לכל קבוצה.

אם מוסיפים עץ אחד לכל קבוצה, זה יהיה התת-עץ היחיד בגובה הנ"ל בקבוצה כי גובה התת-עצים שמפרידים קטן בכל צעד.

אם מוסיפים 2 תתי-עצים, נוסיף אותם לתוך הקבוצה לפי הסדר הנ"ל : בקבוצה big התת-עץ הראשון מביניהם יהיה התת-עץ הימני יותר, ובקבוצה small התת-עץ הראשון יהיה התת-עץ השמאלי יותר.

נשים לב כי בצורה זאת חילקנו את T ל2 סדרות נפרדות כפי שרצינו, סך הכל עשינו O(logn) פעולות לטובת הדבר כי זה כמו ללכת מהשורש של עץ 3-2 לעלה כלשהו.

נתבונן כעת בסדרה big ונרצה לאחד אותה לעץ אחד. העץ הראשון בסדרה הוא העץ הגבוה ביותר שבו יש את הערכים הגדולים ביותר, ומתקיים כי הגבהים השונים של העצים מקיימים כי h(i-1)<= h(i) ובנוסף מתקיים כי הערכים בתת עץ הi קטנים מהערכים בתת עץ הi-1. בנוסף יש לנו את חסם תחתון על הערך בכל עץ. נשים לב כי עבור העץ הi-1 המינימום של העץ הi מהווה חסם עליון עבור ערכיו. כלומר עבור כל עץ פרט לעץ הראשון יש לנו 2 חסמים על ערכי העץ – מקסימום ומינימום.

אם נסתכל שוב על האלגוריתם בסעיף ג' נראה שאלה בדיוק הנתונים שהוא באמת משתמש בהם – על מנת לאחד עצים בגובה יורד עם ערכים יורדים, צריך לדעת לעץ הקטן יותר מביניהם את המקסימום של העץ על מנת ליצור רשומה חדשה עבורו בעץ הגדול.

לכן נוכל לאחד את כל הסדרה big לעץ יחיד באיטרציה אחת באורך הפרש הגבהים בין העץ הגבוה ביותר בסדרה לבין העץ הנמוך ביותר בסדרה – במקרה הגרוע בין עץ בגובה logn ועץ בגובה n : סה"כ O(logn) פעולות.

נשים לב כי עבור הקבוצה small נשתמש באותו אלגוריתם כמו בbig רק במקום לאחד לפי המקסימום של כל עץ, כעת נאחד לפי המינימום (שהוא המקסימום של העץ הקודם בסדרה) כאשר עדיין מאחדים לתוך העץ הגדול (שערכיו הם הכי קטנים בסדרה). – סה"כ O(logn) פעולות.

לכן בסך הכל כדי להפריד את העץ ל2 עצים מבצעים O(logn) פעולות.

<u>שאלה 4</u>

המבנה הכללי הוא 2 עצי 2-3, כאשר העלים בעץ הם ערכים במערך וההליכה בעץ היא לפי אינדקסים של המערך. נשתמש בהרחבה על המבנה הנ"ל שהוכחנו בשאלה 3.

<u>Init():</u>

מקצים 2 עצי 2-3 עם n עלים: O(n) פעולות

עוברים על המערך ושמים בעץ אחד (array_tree) את הערכים לפי הסדר במערך: O(n) פעולות

עוברים על המערך הפוך, ושמים בעץ השני (mirror_tree) את הערכים, כך שנוצר תמונת מראה של המערך בעץ: O(n) פעולות.

סה"כ O(n) פעולות.

<u>Get(k):</u>

מחפשים בעץ array_tree את האינדקס המבוקש (חיפוש בעץ 2-3) : O(logn) פעולות.

<u>Reverse(i,j):</u>

עושים split פעמיים על כל עץ – פעם אחת לפי i ופעם אחת לפי j: O(4*logn) פעולות.

כעת לוקחים את העץ המרכזי של array_tree ומאחדים אותו עם העצים הקיצוניים של mirror_tree ע"י 2 פעולות join. ועושים את אותה פעולה על העצים הנותרים: O(4*logn) פעולות.

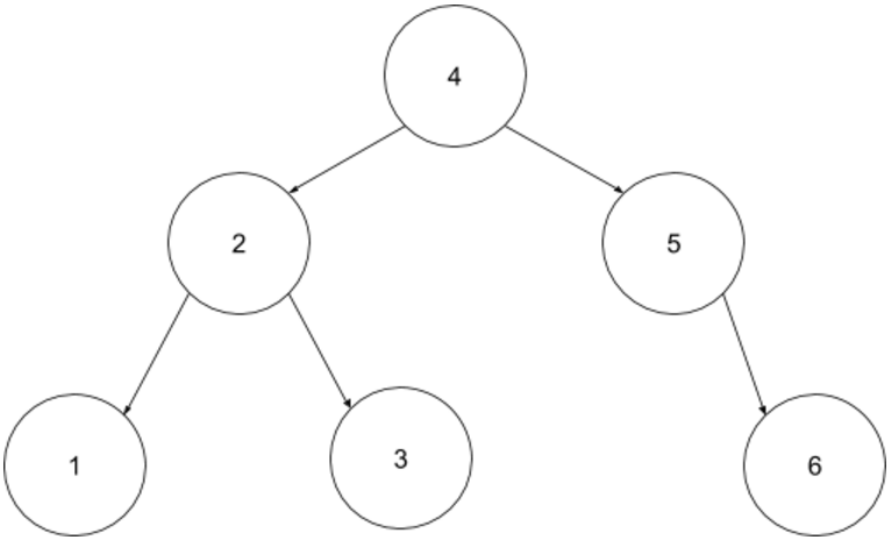
נשים לב, כי בכל זמן נתון, mirror_tree מהווה תמונת מראה של array_tree, וכך גם לכל תת קטע של אינדקסים בעצים. לכן, ע"י החלפה של קטע אינדקסים בין העצים, אנחנו שומרים על כך שזוהי תמונת מראה בכל שלב.

סה"כ O(logn) פעולות.

שאלה 5

א. נפריך. נסמן את השורש בעץ r וצומת כלשהי תחתיו ב-v. אם v צומת משמאל, בסיור inorder הסדר יהיה: … r v… ובסידור pre-order הסדר יהיה … v r . אם v צומת מימין, בסיור post-order הסדר יהיה v r … ובסיור pre-order הסדר יהיה … v r.

ב. נפריך. נקח את העץ הבא:



מתקיים 3<6 אבל המסלול מ-3 ל-6 עובר דרך 2

ג. נפריך. בעץ 2-3 מספר הצמתים תלוי בגובה ובכמה בנים יש לכל צומת. לכל צומת חוץ מהשורש יש מינימום 2 בנים ומקסימום 3. בנוסף הגובה של עץ 2-3 הוא קבוע, ולכן הגובה של T_L שווה לגובה של T_M ו-T_R. לכן מתקיים:

L, M, R = $\Omega(2^h)$
L, M, R = O(3^h)
L + M + R = O(3^h) != $\Omega(2^h)$

לכן:

L != Ω(L+M+R) != Θ(L+M+R)

ד. נפריך. נניח שלפני delete הערך x מופיע ב-n רמות של רשימת הדילוגים. אחרי delete הערך x לא מופיע כלל, ואחרי ה-insert השני מספר הפעמים שיופיע הוא הסתברותי ואין הבטחה שיהיה שווה ל-n.

ה. נוכיח. לפני פעולת insert רשימת הדילוגים בעלת n רמות. פעולת ה-insert לא משנה אף ערך קיים ברשימה, ובהסתברות מסוימת מוסיפה עוד רמה כך שמספר הרמות ברשימה הוא n+1 והרמה העליונה מכילה רק את הערך x. לאחר פעולת delete כל ערכי x הוסרו, ואם קודם לכן התווספה רמה עליונה היא תהיה ריקה ותוסר, כך ששוב הרשימה מכילה n רמות. שאר הערכים לא השתנו, ולכן מבנה הרשימה נשאר זהה.