עבודת בית 3

שאלה 1:

סעיף א:

נניח כי x הוא עלה בעץ AVL צ"ל ש $d(x) \geq \lceil \frac{h}{2} \rceil$ כאשר $d(x)$ העומק של x ו- h גוב העץ.

נוכיח באינדוקציה על גובה העץ: נסמן $d(x)$ את העומק של עלה כלשהוא x

מקרה בסיס עבור h=0 :

במקרה זה יש בעץ רק קדקוד אחד לכן העומק שלו שווה ממש ל 0 כלומר מתקיים

$0 \geq \lceil \frac{0}{2} \rceil \leftarrow 0 = 0 \quad d(x) \geq \lceil \frac{h}{2} \rceil$

ה.א: נניח שלכל n<h מתקיים $d(x) \geq \lceil \frac{n}{2} \rceil$

צעד האינדוקציה: נוכיח עבור h שמתקיים $d(x) \geq \lceil \frac{h}{2} \rceil$

יהי h אז $d(x) = h - 1$ או $d(x) = h$ כי הוא עץ AVL נחלק למקרים:

מקרה ראשון:

$d(x) = h$: $d(x) \geq \lceil \frac{d(x)}{2} \rceil \leftarrow 2d(x) \geq d(x) + 1$ וזה נכון כי $h = d(x) \geq 1$

מקרה שני:

$d(x) = h - 1$: $d(x) + 1 = h \leftarrow d(x) \geq \lceil \frac{d(x)+1}{2} \rceil \leftarrow 2d(x) \geq d(x) + 1$

לכן מתקיים $d(x) \geq \left\lceil \frac{h}{2} \right\rceil$.

** קל מאוד לראות שהחסם ההדוק לעץ AVL עם גובה של h הוא כאשר העומק המינימלי לעלה כלשהו יהי שווה ממש ל $\lceil \frac{h}{2} \rceil$ כלומר כל עצי AVL המינימליים בגובה של h המקיימות $d(x) = \lceil \frac{h}{2} \rceil$.

סעיף ב:

נתבונן בעץ AVL מינימלי נסמנו בT_m עבור k קדקודים (ידוע לנו כי מחיקה בעץ מינימלי תדרוש $\Omega(\log n)$ רוטציות) ונוכיח עבורו כי מחיקת קדקוד ממנו תדרוש לפחות $c * \log(|T_m|)$ רוטציות, כאשר c קבוע חיובי לא תלוי ב-n ו-$|T_m|$ מספר הצמתים בעץ המינימלי.

T_m עץ AVL מינימלי עם גובה h נשתמש כעת בעץ שהצד הימני עם גובה פחות מצד שמאלי, וכל קדקוד מגובה i שני הבנים שלו עם גבהים i-2 או i-1 בהתאם.

נגדיר $a_1, \dots, a_d$ המהוות לנו את המסלול מהשורש אל האיבר הכי ימני ב T_m שאנו רוצים למחוק (הראינו בסעיף קודם כי $d = \left\lceil \frac{h}{2} \right\rceil$).

אחרי קריאה לשיטת המחיקה בעץ המינימלי T_m אנו מקבלים כי כל קדקוד a_i כאשר $1 \le i \le d - 1$ צריך איזון (כי אנחנו בעץ AVL מינימלי).

אנו מזהים כי האב הקדמון של האיבר המחוק הפך להיות לא מאוזן וצריך לאזנו, כנ"ל לגבי האב של האב הקדמון של האיבר המחוק וכך הלאה נאזן עד שמגיעים לשורש, גם אחרי מחיקת העלה גובה העץ T_m יפחת ב 1.

לעשות את כל הבלגן הזה עולה לנו $\Omega(d)$ סיבובים, שזה שווה ל $\Omega(h)$

$\Omega(h) = \Omega(\log(|T_m|)$. כנדרש

סעיף ג:

לכל קדקוד בעץ נוסיף שני מצביעים $successor, predessor$, נסרוק את העץ בסריקת - *inorder* ומקבלים את הקדקודים בסדר עולה: $x_1, x_2, \dots, x_n$.

נניח כי מכניסים איבר x בין x_k לx_{k+1} נעדכן ה $successor$ של x להיות x_{k+1} כיוון שסדר העץ *inorder* כנ"ל לגבי ה $predessor$ להיות x_k.

בעת ההכנסה לעץ קודם נמצא את המקום המתאים לאיבר הנכנס, ואחר כך נחפש את ה $successor, predecessor$ ונעדכנם. זמן הריצה עדיין $\theta(\log n)$

בעת המחיקה מהעץ נחפש את ה $successor, predecessor$ של האיבר שרוצים למחוק נסמן את $successor$ ב s ואת ה $predecessor$ב p, ואחר כך נעדכן $s.predecessor \leftarrow p$, $p.successor \leftarrow s$ זמן הריצה עדיין $\theta(\log n)$.

בפעולת *Find(x, k)* נמצא את *x*. כעת יש לנו את ה*successor* שלו ולכן נעבור על מצביעי ה*successor* *k-1* פעמים או עד שמגיעים ל*null*, וכך נקבל את k האיברים הגדולים מ-x. זמן הריצה $\theta(\log n + k)$.

שאלה 2:

נשתמש עץ AVL אחד שמכיל 3 שדות:

שדה ל ערך x

שדה ל ערך y

שדה לגובהה h

נגדיר בנוסף לשדות החוליה בעץ מצביע הדדי שיצא מהבטן (אמצע) של החוליה לאם ישנם בעץ ערכי x-ים שווים אזי המצביע יחברם כאחים.

פעולה	**תיאור**
$Init()$	אתחול עץ AVL דורש O(1)
$Insert(x, y, h)$	נכניס ל עץ את ה x,y,h המתקבלים לפי תכונות עץ ה AVL, כלומר **אם** העץ ריק אז נוסיפם ללא אילוץ. **אחרת** נכניס את האיבר הבא בתור לפי ערכי ה x-ים נבדוק את גודלו של x אם קטן מ הx הקיים אזי הבא בתור ייכנס בצד שמאל ואם גדול ייכנס בצד ימין. אם הגענו באיזשהו מקרה להכנסת x שווה ל x שכבר נמצא אזי פשוט מעדכנים מצביע שיצא מה x הקיים לחדש (כלומר בן אמצעי של ה x הקיים) ונמיינו בצורת עץ לפי ה y שלו כלומר אם ה y הנכנס קטן מה y של ה x הקיים אזי נלך שמאל ונחפש מקומו שם (לפי ה y-ים) ונכניסו אם גדול ייכנס בצד ימין. כיוון שאנו בעצי AVL אזי בכל פעם שאנו מוצאים המקום המתאים להכנסה בודקים את איזון העץ ושומרים על תכונת הגובה שלא יהיה יותר מ 1 ולכן זה דורש $O(\log n)$.
$Delete(x, y)$	בעת המחיקה של הערכים מהעץ אנו מחפשים קודם איפה נמצאים ה x, y כיוון שזה עץ AVL זה לוקח $O(\log n)$, **אם** הערכים אינם נמצאים בעץ אזי מדפיסים על המסך הודעת שגיאה שהאיבר לא נמצא, **אחרת** יש לכך 2 מקרים: <u>מקרה ראשון:</u> אין לאיבר שרוצים למחוק אחים. נגיע לאיבר שרוצים למחוק ובך יהיה לנו 3 מקרים:

1. אם הוא עלה כלומר אין לו בנים אזי אומרים לאב שלו שיצביע ל null , ולא לשכוח לאזן את העץ.
2. אם לאיבר שרוצים למחוק יש בן יחיד לא משנה אם הוא ימני או שמאלי אזי נאמר לאבא של האיבר שרוצים למחוק אם קיים להצביע לבן של האיבר המחוק, ונסיר כל המצביעים שלו אם יש.
3. אם לאיבר שרוצים למחוק יש שני בנים אזי שואלים אותו אם יש לו אב אם כן יש לו אזי נגיד לאב ש ה $successor$ הוא הבן שיחליף את הבן הנמחק שלו כלומר האבא יצביע ל $successor$ וצד השמאלי של ה $successor$ יהיה הצד השמאלי של האיבר המחוק כנ"ל הצד הימני של ה $successor$ יהיה הצד הימני של האיבר המחוק, אחרת רק מעדכנים את שני הצדדים של האיבר המחוק להיות בשני הצדדים של ה $successor$, ונסיר כל המצביעים של האיבר שרוצים למחוק אם יש.

ולבסוף מוחקים את ה $successor$ מהעץ שלא יופיע פעמים, ומאזנים את העץ.

<u>מקרה שני :</u> לאיבר שרוצים למחוק יש אחים.
נגיע ל x של האיבר שרוצים למחוק ובכך נוצרו לנו שני מקרים :

אם האיבר הרצוי למחוק הוא הראשון לפני שנכנס לרשימת האחים אזי נחליפו ב$successor$ שברשימת האחים שלו ונמחק את ה $successor$.

אחרת נרד ל קבוצת האחים שלו ונחפש את האיבר שרוצים למחוק שם לפי ה y, ובזה נוצר לנו אותם מקרים של מקרה ראשון.

$Find(x, y)$	בחיפוש אנו מחפשים לפי ה x **אם** נגיע ל x שרוצים למחוק נבדוק את ה y **אם** הוא שווה אזי פשוט נחזיר את הגובה של נקודת הציון, **אחרת** נרד לאחים של ה x אם קיימים ונחפש שם אם ה y גדול ממה שקיבלנו אזי ממשיכים בחיפוש בצד שמאל של העץ וברגע שנמצא את ה x,y המתאימים ל input אזי מחזירים את הגובה של נקודת הציון, המקרה האחרון שאם ה y קטן ממה שקיבלנו אזי ממשיכים בחיפוש בצד ימין של העץ וברגע שנמצא את ה x,y המתאימים ל input אזי מחזירים את הגובה של נקודת הציון, כיוון שהעץ AVL יוצא לנו זמן הריצה $O(\log n)$. **אחרת** נקודת הציון לא קיימת ונחזיר null.
$PrintAll(x)$	בהדפסת הגבהים של כל נקודות הציון שקואורדינטות האורך שווה ל x אנו מחפשים קודם את המופע של ה x, זה לוקח $O(\log n)$ זמן ריצה כיוון שהעץ AVL, אחר כך מדפיסים את ה x ויורדים אל האחים שלו אם קיימים, נעבור על k המופעים של x (k מספר הפעמים בהם x מופיע) ונדפיסם אחד אחר השני. ובסוף זה לוקח לנו $O(\log(n) + k)$ זמן ריצה.

שאלה 3:

סעיף א:

כדי להראות שהאלגוריתם שאנו בונים לוקח לכל היותר $o(|h_1 - h_2|)$ נחלק למקרים:

עבור $h_1 > h_2$:

נתחיל ראשית מהשורש בעץ T_1 נתון לנו כי כל הקדקודים בעץ קטנים מ X וגם הגובה של T_1 גדול מעץ T_2 אז נעבור על הצד הימני עד שנגיע לקדקוד שגובהו הינו שווה ממש לגובה של T_2, נטבל במקרה שלא קיים בן ימני מלכתחילה תכף.
כשנפגוש בקדקוד היעד שלנו Y נגדיר את הבן שמאלי של הקדקוד X יצביע על Y (נזכיר כי Y >X) וכמובן האבא של Y להיות X והאבא של Y הקודם נגדיר הבן ימין החדש שלו להיות X.
X קטן מכל העץ T_2 אז נציב את השורש של T_2 להיות הבן הימני של X וכמובן האבא של השורש להיות X.
אנו פתוחים שהעץ החדש הינו ממוין רק חסר את האיזון, נעבור חזרה לקדקודים הקודמים של העץ עד למעלה שנגיע לשורש נפעיל רוטציה על העץ שזה לוקח *O(1)* וכך נוודה שהעץ יהי מאוזן
אם לא היה בן ימני אז פשוט נחלק לכל המקרים:
העץ T_1 יהי רק משני קודקודים אם T_2 היה רק וקדקוד אחד אז נציב X להיות השורש בעץ החדש ולימינו את השורש של T_2 ומשמאל נציב השורש של T_1
ואם היה עץ ריק פשוט נציב X להיות מצד ימין של השורש של T_1 .

עבור $h_1 = h_2$:

רק נציב לימין X את השורש של T_2 ומשמאלו את השורש של T_1
כל צד ימין קטן מצד שמאל והגבהים שווים לכן העץ החדש כן מאוזן.

עבור $h_1 < h_2$:

נתחיל ראשית מהשורש בעץ T_2 נתון לנו כי כל הקודקודים בעץ גדולים מX וגם הגובה של T_1 קטן מעץ T_2 אז נעבור על הצד השמאלי עד שנגיע לקדקוד שגובהו הינו שווה ממש לגובה של T_1, נטבל במקרה שלא קיים בן שמאלי מלכתחילה תכף.

כשנפגוש בקדקוד היעד שלנו Y נגדיר את הבן הימני של הקדקוד X יצביע על Y (נזכיר כי Y <X) וכמובן האבא שלY להיות X והאבא שלY הקודם נגדיר הבן שמאל החדש שלו להיות X.

X גדול מכל העץ T_1 אז נציב את השורש שלT_1 להיות הבן השמאלי שלX וכמובן האבא של השורש להיות X.

אנו פתוחים שהעץ החדש הינו ממוין רק חסר את האיזון, נעבור חזרה לקדקודים הקודמים של העץ עד למעלה שנגיע לשורש נפעיל רוטציה על העץ שזה לוקח *O(1)* וכך נוודא שהעץ יהי מאוזן.

אם לא היה בן ימני אז פשוט נחלק לכל המקרים :

העץT_2 יהי רק משני קודקודים אם T_1 היה רק קודקוד אחד אז נציב X להיות השורש בעץ החדש ומשמאלו את השורש של T_1 ומימין נציב את השורש של T_2

ואם היה עץ ריק פשוט נציב X להיות מצד שמאל של השורש של T_2 .

<u>סעיף ב :</u>

בהינתן עץ T ומפתיח K נרצה לחפשו עד שנגיע אליו זה מצריך $\theta(\log n)$

כל פעם שנפגוש בקדקוד שהוא קטן מK נסמנו X_i כך ש $0 \le i \le \log n$ לכל קדקוד קטן מK יש גם תת עץ משמאלו כל איבריה קטנים מK נסמן כל תת עץ כזאת ב t_i כך ש t_i היא תת עץ שלX_i.

באופן דומה אם נפגוש במהלך החיפוש עלK בקדקוד גדול מK נסמנו ב Y_i וכל תת עץ מימין ל Y_i חייבת להכיל בה רק איברים שגדולים מ K נסמן כל תת עץ כזאת ב r_i .

כשנגיע ל K נחבר שתי תתי העץ t_i עם t_{i-1} ו הקדקוד X_{i-1} לפי סעיף הקודים כי $\text{Max } t_{i-1} < X_{i-1} < \text{Min } t_i$ נסמן העץ החדש שיצא לנו ב T_j נשלב בשלב הבא בין T_j ל t_{i-2} וכן הלאה עד שנגיע ל$i=0$ כך יצא לנו עץ שבו כל האיברים הקטנים מK ורק נשאר לנו להשלים עם הK עצמו, נעשה פעולת insert(k) שלוקחת לכל היותר $\theta(\log n)$.

נשים לב כי לכל T_j הגובה שלו יהי קטן או שווה ל גובה של t_0 (כי אחרת לא יהי עץ מאוזן) לפי פעולת השילוב בסעיף הקודם קיבלנו כי כל פעולת שילוב שני עצים לוקחת לכל היותר $o(|h_1 - h_2|)$ נעשה הפעולה הזאת מספר קבוע של פעמים שהוא מספר הקודקודיים שסימנו אותם שקטנים מ K לכן נקבל שהפעולה תצריך גם $o(|h_1 - h_2|)$

במקרה הגרוע ביותר $h_2 = 0$ וגם $h_1 = \log n - 1$ וזה גם ייתן לנו $\theta(\log n)$ לייצר את העץ של הקודקודים הקטנים מ K .

<u>סעיף ג:</u>

נאתחל שני עצי AVL T_1, T_2 כך שכל קדקוד עם צבע שחור ייכנס לעץ T_1 אחרת ייכנס לעץ T_2.

פעולה	תיאור	זמן ריצה
Add(x)	בפעולת ההכנסה נבדוק תחילה מה הצבע של הקדקוד הנכנס, אם הוא שחור אז נכניס אותו לעץ T_1 ואם הוא לבן אז נכניס אותו לעץ T_2 . כל הכנסה רק נחפש את המקום המתאים להכניס את X ולכן נצטרך $O(\log n)$ במקרה הגרוע ביותר כמובן כי אנו מדברים על שני עצי חיפוש AVL .	$O(\log n)$
Color(x)	נגדיר משתנה בוליאני found יהי בהתחלה false עד כדי מציאת האיבר עם המפתח X אם בסוף החיפוש בשני העצים לא מצאנו את האיבר עם המפתח הנדרש (כלומר המשתנה found נשאר שלילי) אז נחזיר ערך *null* אך אם הוא כן נמצא נבדוק באיזו עץ נמצא האיבר, אם הוא ב T_1 אז הצבע שלו שחור ואם הוא ב T_2 אז הצבע שלו לבן.(אפשר לעשות לזה גם ערך בוליאני בשם *black* והוא יתחיל ב *true* אם הוא כן נמצא בעץ של שחורים אז נחזיר שהצבע שחור אחרת אם האיבר נמצא בעץ של הלבנים אז נחזיר שהצבע לבן אחרת נחזיר *null*) <u>הערה חשובה</u>: לא נרצה לבדוק בשדה מה הצבע כי אם הפעלנו את פעולת FlipColors(k) נשנה את	$O(\log n)$

	הצבע שלו בלי לשנות את השדה וזה בשביל לקצר זמן כמה שניתן. זמן הריצה כמובן לוקח כמו החיפוש על האיבר עם המפתח X שזה $O(\log n)$ שוב כי אנחנו מדברים על עצי חיפוש וברגע מציאת האיבר, בדיקת הצבע שלו תיקח $O(1)$ ולכן סך הכל הפעולה צריך $O(\log n)$.	
FlipColors(k)	כדי להחליף את הצבעים בזמן ריצה הכי קצר נשתמש בפונקציית *Split* כאשר נבצע אותה בשתי העצים T_1 ו T_2, כעת יצא לנו ארבע עצים (שניים קטנים שווים לK אחד שחור והשני לבן ושניים גדולים מ K אחד שחור והשני לבן) נשתמש בפונקציית השילוב בסעיף הראשון נחבר בין תתי העץ הקטנים שווים לK השחורים ל עץ הלבן T_2 והגדולים נשאיר אותם כי לא רוצים להחליף את צבעיהם אותו דבר לגבי תת העץ הקטן שווה לK הלבן נחבר אותו לעץ השחור T_1 ושאר הקדקודים כיוון שהם גדולים מ K לא נגע בהם. כך נוודה שכל קדקוד קטן או שווה לK החליף את הצבע שלו (בפעולת השילוב צריך לתת קדקוד גדול מכל העץ הראשונה וקטן מכל העץ השנייה, ניתן לצייר קדקוד זבל כדי להפעיל את השיטה ולאחר מכן נוכל לזרוק אותו זה לוקח $O(\log n)$) מבחינת זמן ריצה אנו רק מחפשים על הקדקוד K וברגע שנמצא אותו זה לוקח $O(\log n)$ נפעיל על העצים פעולות הפיצול והשילוב שלוקחות גם הן $O(\log n)$ אז סך הכל פעולת ההחלפה FlipColors(k) תצטרך $O(\log n)$.	$O(\log n)$

שאלה 4:

נמצא האיבר ה k בגודלו בעץ B באמצעות הוספה של שדה $size$ בדומה למה שלמדנו בתרגול על BST, אך אנו בעץ זה מוסיפים לכל איבר בתוך ה $size\ node$ אחר, ה $size$ מייצג את כמה בנים יש ל $x.keys[i]$ (x=node) כך ש $0 \le i < 2 * t - 1$ (ההוספות והמחיקות וכל הפעולות האחרות שלמדנו מתבצעות כרגיל לפי מה שלמדנו בהרצאות בדיוק, רק עם השינויים של $size$ המתעדכנים תוך כדי כלומר בעת הכנסת בן נוסיף 1 ובעת הסרת בן נקטן את ה $size$).

נחזור לאופן מימוש פעולת ה k בגודלו, קודם **אם** אנו ב $root$ של עץ B והאורך שלו שווה ל אפס אזי אין איברים בכלל נחזיר $null$.

נמשיך באופן רקורסיבי **אם** הגענו לסוף המערך של ה $keys$ אזי נשאר לנו את הבן הכי גדול של האיבר האחרון במערך $keys$ אם קיים כיוון ש במערך עם i איברים יש $i + 1$ בנים, נלך לבן הזה ונבדוק אם שווה

אם יש עוד איברים במערך $keys$ נגדיר משתנה שישמור לנו את ה size של האיבר שאנו נמצא בו ועוד 1 (האיבר עצמו) (כלומר $temp = x.key[i].size + 1$), אחר כך נשווה את מה שיצא לנו עם K הנתון **אם** מה שיצא לנו קטן מ k אזי נחזור ל פונקציית ה k בגודלו (כלומר רקורסיה) עם העץ עצמו, דהיינו איפה שאנו נמצאים כרגע אך נקפוץ באחד ימינה לאיבר אחריי (כלומר $i + 1$) ונחסר את התשובה שקיבלנו מה k, ($temp$) **אחרת אם** מה שיצא לנו גדול מ k אזי נחזור ל פונקציית ה k בגודלו (כלומר רקורסיה) עם תת העץ של הבן של x במקום ה i , k הנתון ומתחילים מהבן הראשון של תת העץ.

אחרת בוודאות מה שיצא שווה ל k אזי זה האיבר שאנחנו רוצים ופשוט מחזירים אותו.

תיאור בבסודה-קוד $\downdownarrows$

pseudocode

מתחילים עם $x = \text{B}\ tree$, $i = 0$ הנתונה וK משתנה המתקבל.

C מייצג את מערך הבנים, x מייצג החוליה שנמצאים בה (כמובן שיש בתוכה מערך).

```
K-thElement (x, k, i)
If (x.keys.length==0)
     Return null
If (i==x.keys.lenght)
    Return K-th element (x.c[i+1], k, 0)
Temp=x.keys[i].size+1
If (temp<k)
   Return K-th element (tree, k-temp, i+1)
ELSE IF (temp>k)
  Return K-th element (x.c[i], k, 0)
Else
  Return x.keys[i]
```

ניתוח זמן הריצה:

במקרה הגרוע נקבל שזמן הריצה הוא:
$(2t-1) * \theta(\log_t(n)\ ) = \theta(t * \log_t(n)) = \theta\left(\frac{t}{\log(t)}\log(n)\right)$.

כיוון שהאיבר ה-k בגודלו וכל שאר הקדקודים במסלול גודלם יהיה $2 * t - 1$, אז עוברים על כל גובה העץ ובכל קדקוד שעוברים עליו עוברים על כל המפתחות בו.