מבני נתונים 2021 סמסטר א' מועד א'

שאלה 1

א. נשתמש בטבלת hash בגודל n שמאותחלת לאפסים (ניתן להניח שאפשר לעשות זאת בזמן קבוע). בכל פעם שמגיע מפתח חדש a_i מחשבים ב$O(1)$ במקרה הגרוע את המספר $\frac{\sqrt[3]{T}-3a_i}{2}$ (אפשר כי המספרים מתחום גדול וידוע מראש) ובודקים האם הוא נמצא בטבלת hash ב$O(1)$ בתוחלת (נשים לב שתמיד $\alpha \leq 1$ ולכן זה באמת $O(1)$). אם הוא נמצא עונים "כן" ואחרת עונים "לא" ומוסיפים את המפתח a_i לטבלה ב$O(1)$ בתוחלת.

ב. נשתמש בperfect hash בגודל $O(n)$ ונחפש את המפתח $\frac{\sqrt[3]{T}-3a_i}{2}$.

שאלה 2

א. האלגוריתם יהיה רקורסיבי ויקבל 4 פרמטרים $T(A,l,r,num)$ ויעבוד באופן הבא:

בכל פעם נעבוד בטווח $l \le \cdots \le r$ של המערך (בהתחלה $l=0, r=|A|-1$) ונחפש את המספר הראשון כך שאם נסכום את המשקלים של האיברים מסודרים לפי הסדר נקבל $\le num$ (בהתחלה $num=\frac{1}{2}$). כלומר הקריאה הרקורסיבית הראשונה היא $T(A,0,|A|-1,\frac{1}{2})$. בכל קריאה רקורסיבית נמצא את החציון ב$A[l,\dots,r]$ ב$O(r-l)$ ונביא אותו למקום הנכון ב$A[l,\dots,r]$ ב$O(r-l)$ (כמו בpartition שבquicksort). כעת נסכום את המשקלים עד אותו חציון כולל ב$O(r-l)$ ואם $sum < num$ נמשיך לחצי העליון של $A[l,\dots,r]$ (לא כולל החציון עצמו) עם $num_{new} = num - sum$. אחרת אם $sum = num$ נחזיר את החציון ונעצור. אחרת נמשיך לחצי התחתון של $A[l,\dots,r]$ (כולל החציון עצמו) עם $num_{new} = num$. מקרה הבסיס יהיה כשהאורך של הטווח הוא 1 ואז נחזיר פשוט את המספר היחיד שבטווח.

נוסחת הנסיגה היא $T(n) = O(n) + T(\frac{n}{2})$ שפתרונה הוא $T(n) = O(n)$.

ב. האלגוריתם דומה לאלגוריתם מסעיף א' ויקבל 3 פרמטרים $T(A,l,r,M)$:

בכל פעם נעבוד בטווח $l \le \cdots \le r$ של המערך (בהתחלה $l=0, r=|A|-1$) ונחפש את המספר הראשון כך שאם נסכום את האיברים לפי הסדר עם M נקבל מספר חיובי. כלומר הקריאה הרקורסיבית הראשונה היא $T(A,0,|A|-1,0)$. בכל קריאה רקורסיבית נמצא את החציון ב$A[l,\dots,r]$ ב$O(r-l)$ ונביא אותו למקום הנכון ב$A[l,\dots,r]$ ב-$O(r-l)$ (כמו בpartition שבquicksort). כעת נסכום את האיברים עד אותו חציון כולל בזמן שהוא $O(r-l)$ ואם $0 < sum + M$ נמשיך לחצי התחתון של $A[l,\dots,r]$ (כולל החציון עצמו) עם $M = sum$. אחרת אם $sum + M \le 0$ נמשיך לחצי העליון של $A[l,\dots,r]$ (לא כולל החציון עצמו) עם $M = sum$. מקרה הבסיס יהיה כשהאורך של הטווח הוא 1 ואז נחזיר פשוט את המספר היחיד שבטווח.

נוסחת הנסיגה היא $T(n) = O(n) + T(\frac{n}{2})$ שפתרונה הוא $T(n) = O(n)$.

שאלה 3

א. האלגוריתם יעשה שימוש בערימת מינימום ויעבוד באופן הבא:

נאתחל ערימת מינימום מk האיברים הראשונים במערך בזמן $O(k)$. כעת המינימום של המערך הוא בהכרח המינימום של הערימה אז נוציא ונמחק אותו ב$O(logk)$ ונשמור אותו במערך חדש במקום הראשון. כעת נכניס את האיבר ה$k+1$ לערימה ב$O(logk)$ ונבחין שכעת האיבר השני קטן ביותר חייב להיות המינימום של הערימה. נוכל להמשיך כך במשך $n-k+1$ איטרציות (בכל פעם נשלוף מינימום מערימה עם לכל היותר k איברים ונוסיף לערימה את האיבר הבא). כל איטרציה היא $O(logk)$ ואם נוסיף את העלות של להעתיק את האיברים מהמערך עזר שלנו (אפשר לפעול גם בלעדיו) אז צריך להוסיף בסוף עוד $O(n)$ פעולות. סה"כ מקבלים $O(k)+O(nlogk)+O(n)=O(nlogk)$ כדרוש.

ב. נשיג חסם תחתון על מספר העלים בעץ ההשוואות של האלגוריתם. נניח שמגיעים n איברים ואנחנו מחלקים אותם ל$\sqrt{n}$ מערכים באורך $\sqrt{n}$. נשים לב שכל פרמוטציה של מערך כזה היא חוקית (כי המרחק של כל איבר בתוך מערך כזה מכל איבר אחר קטן מ$\sqrt{n}$) בתור סידור סופי של האיברים. לכן יש לפחות $\left(\sqrt{n}\right)!^{\sqrt{n}}$ עלים בעץ ההשוואות. כלומר הגובה של עץ ההשוואות הוא לכל הפחות $\log\left(\left(\sqrt{n}\right)!^{\sqrt{n}}\right)$ (כי ראינו בכיתה שלעץ בגובה h יש לכל היותר 2^h עלים ולכן אם יש לעץ k עלים לא ייתכן שהגובה שלו פחות מ$logk$ כי אז 2 בחזקת הגובה שלו זה חסם עליון על מס' העלים שלו אבל זה יהיה קטן מk בסתירה לכך שיש לו k עלים). אבל נשים לב ש:

$$\log\left(\left(\sqrt{n}\right)!^{\sqrt{n}}\right)=\sqrt{n}\log\left(\left(\sqrt{n}\right)!\right)\overset{*}{=}\sqrt{n}\cdot\Omega\left(\sqrt{n}log\sqrt{n}\right)=\Omega\left(\mathrm{nlog}\sqrt{n}\right)=\Omega(\mathrm{nlogn})$$

* כי ידוע ש$\log(n!)=\Theta(nlogn)$

שאלה 4

נשתמש בעץ AVL שבו המפתחות הם הזוויות, וב$counter$ שהוא סכום הזוויות של כל פעולות $Rotate$ עד ההווה. בנוסף כל צומת תחזיק שדה נוסף שהוא $offset$. בכל פעם שנצטרך להתייחס למערך של מפתח בצומת x נתייחס במקום לערך הבא :
$x.key + counter - offset$ נראה איך לממש את כל הפעולות בזמן הדרוש:

<u>$Insert(info, \theta)$:</u>

ניצור צומת חדשה עם הזווית המתאימה והערך המתאים, ו$offset = counter$ הנוכחי.
מעבר לזה ההכנסה היא הכנסה רגילה לעץ AVL.

<u>$Delete(\theta)$:</u>

נמצא את האיבר ונמחק אותו כמו בעץ AVL רגיל.

<u>$Search(\theta)$:</u>

שוב חיפוש רגיל בעץ AVL .

<u>$Rotate(\theta)$:</u>

נעשה $counter += \theta$.

<u>$DeleteArc(\theta_1, \theta_2)$:</u>

נוסיף את θ_1, θ_2 אם הם לא קיימים כבר, ונעשה את הפעולות הבאות:

$T_1, T_2 \leftarrow Split(T, \theta_1)$ (Note that: $T_1 < \theta_1 < T_2$)

$T_3, T_4 \leftarrow Split(T_2, \theta_2)$ (Note that: $T_3 < \theta_2 < T_4$)

$return\ Join(T_1, T_4)$

שאלה 5

א.

קודם אוכיח שאם מערך A באורך N מייצג ערימה בינארית כאשר האינדקס הראשון הוא 1 אז העלים הם באינדקסים $\left\lfloor\frac{N}{2}\right\rfloor+1,\dots,N$. כזכור הבן השמאלי והבן הימני של צומת באינדקס i יהיו באינדקסים $2i,2i+1$ בהתאמה (מהגדרת אותו מערך). אם כן j הוא אינדקס של עלה אם ורק אם $N<2j$ כלומר אם ורק אם $j\in\{\left\lfloor\frac{N}{2}\right\rfloor+1,\dots,N\}$.

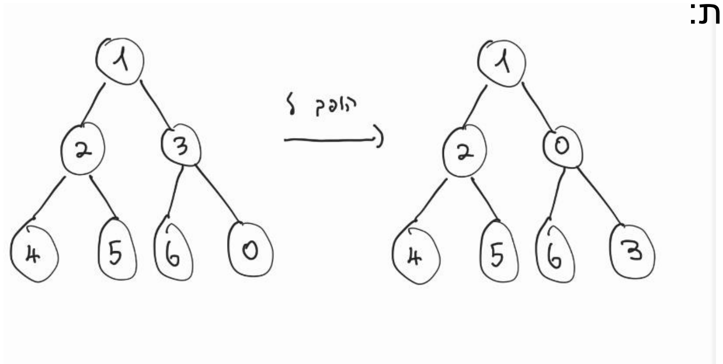
כעת נובע ישירות שמימוש Build_heap(A) עושה Heapify_down מכל צומת שאינו עלה בדיוק פעם אחת כי בדיוק עוברים על שאר האינדקסים בתוך לולאה. כעת ניזכר שאם אנחנו בצומת באינדקס i אז האינדקס של האב של אותה צומת הוא $\left\lfloor\frac{i}{2}\right\rfloor$, היות ו$N=$ 2^h-1 הוא אי זוגי ולכן $\left\lfloor\frac{N}{2}\right\rfloor+1=\left\lceil\frac{N}{2}\right\rceil\neq\left\lfloor\frac{N}{2}\right\rfloor$ כלומר בשני המימושים האחרים מתחילים מצומת שהיא עלה וכל עוד האינדקס שלה בעל זוגיות מסוימת ויש לה אב עולים מעלה ועושים עליו Heapify_down . הסיבה שלא יתכן כי עושים Heapify_down על אותה צומת פעמיים היא שלכל צומת בעץ יש מסלול ייחודי לכל אב קדמון שלה (כלומר לצמתים שונים יהיו מסלולים שונים לאותו אב קדמון) ולכן אם נדמיין שיורדים מהאב הקדמון למטה עד שהמסלולים מתפצלים (כי הם חייבים להיות שונים) אחד יעבור בצומת שהיא בן ימני ואחד יעבור בצומת שהיא בן שמאלי והזוגיות שלהם בהכרח שונה ($2k,2k+1$ עבור k כלשהו) ולכן באלגוריתם ספציפי שבו נדרשת זוגיות ספציפית לא ייתכן ששני המסלולים הללו יתקבלו אלא רק אחד מהם. לכן גם בBuild_heap1(A) וBuild_heap2(A) עוברים מכל צומת שאינו עלה בדיוק פעם אחת. עבור הזמן הריצה נשים לב שמההסבר בכיתה יתקיים –

$$time=1\cdot\frac{n}{2}+2\cdot\frac{n}{4}+\cdots+h\cdot 1\leq n\sum_{h=1}^{\infty}\frac{h}{2^h}=2n=O(n)$$

מנגד היות ויש $\left\lfloor\frac{N}{2}\right\rfloor$ צמתים שאינם עלים וHeapify_down $=\Omega(1)$ ועושים זאת על כל אחד מהם נקבל גם $\Omega(n)$ ולכן סה"כ מקבלים $time=\Theta(n)$.

ב.

דוגמה נגדית:



ג.

הוכחה:

נוכיח שאם הפעלנו את Heapify_down על צומת כלשהי אז הפעלנו אותה גם על כל צומת שהיא לא עלה בתת העץ שהיא השורש שלה. היות והשורש במקום אי זוגי זה אומר בפרט שהפעלנו את Heapify_down על כל צומת שאינה עלה ולכן תתקבל ערימה תקינה. באינדוקציה על גובה הצומת, אם הגובה הוא 0 היא עלה ולכן הפעלנו את Heapify_down על כל צומת שהיא לא עלה בתת העץ שהיא השורש שלה (באופן ריק). נניח נכונות לגובה h ונוכיח לגובה $h+1$. תהי צומת בגובה $h+1$ שהפעלנו עליה את השגרה. השגרה Build_heap2(A) בהכרח ביקרה בבן הימנית שלה ולכן תת העץ הימני תקין. אבל לבן השמאלי שלה יש בעצמו בן ימני ואותו בהכרח ביקרנו ולכן ביצענו Heapify_downעל אבא שלו (הבן השמאלי) כלומר מהנחת האינדוקציה גם תת העץ הימני תקין ולכן כל תת העץ תקין כדרוש.