מבני נתונים - תרגיל יבש 2

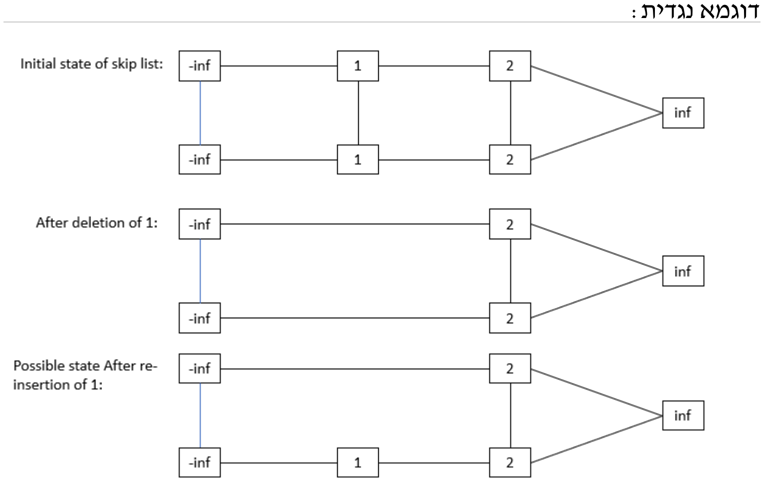
29 בדצמבר 2020

שאלה 1

סעיף א

הפרכה:

דוגמא נגדית:



הרשימה במצב ההתחלתי אפשרית, כאשר גם בהכנסת 2 ו גם בהכנסת 1 נקבעה הוספה של רמה חדשה אחת.

הסרה של 1 תמחק אותו מכל הרמות.

הכנסה של 1 בפעם השניה תוסיף אותו ברמה התחתונה, אך יתכן שבהגרלה האם יתווסף 1 גם ברמה השניה , יוכרע לא להוסיף.

סעיף ב

הוכחה:

תהי S רשימת דילוגים אקראית כך שעבור x מסוים, מתקיים $x \notin S$. ברגע מסוים לS יש n רמות, כך שבאף רמה לא מופיע האיבר x. לאחר הפעולה $insert(x)$, האיבר x מתווסף לרמה התחתונה ולמספר אקראי של רמות מעליה.

לאחר הפעולה $delete(x)$, האיבר x מוסר **מכל** הרמות בהן הוא מופיע, ללא תלות באילו רמות נבחרו אקראית. הסרה של איבר מרשימה מתבצעת על ידי חיבור של האיברים מלפניו ומאחוריו - איברים אילו היו קיימים לפני הוספת x והוא הוסף ביניהם.

לכן, הסרת x מחברת מחדש ביניהם ומחזירה אותם למצב ההתחלתי בלי להשפיע על איברים אחרים בS. בנוסף, האיבר x שלא היה קיים בS לפני הוספתו מוסר ממנה לחלוטין, ולכן S חוזרת למצבה ההתחלתי עם ההנחה כי רמה עליונה שאולי נוצרה על ידי הוספת x תימחק בהסרתו.

כלומר S לא משתנה על ידי רצף הפעולות $insert(x)$ ואז $delete(x)$.

סעיף ג

הפרכה :

נראה דוגמא נגדית בעזרת מקרה הקצה של עץ פיבונאצ'י משמאל ועץ שלם מימין.

נגדיר סדרת עצי AVL באופן הבא :

העץ T_i הוא עץ שמשמאלו עץ פיבונאצי ה $i-1$ ומימינו עץ שלם בגובה $i-1$.

תת העץ השמאלי הוא עץ פיבונאצי,הוא עץ AVL וגובהו $i-1$ כפי שראינו בהרצאה.

תת העץ הימני הוא עץ שלם בגובה $i-1$, ולכן כל העלים נמצאים באותו עומק , מכאן ברור שמדובר גם כן בעץ AVL.

מכאן ש BF של השורש של T_i הוא 0 , והתתי עץ הימני והשמאלי שלו הם עצי AVL ולכן גם הוא עץ AVL.

T_L מקיים $|T_L| \underbrace{=}_{\text{4.6 in the auxiliary matirial}} n_{i+2}-1 \underbrace{=}_{4.7} \Theta(\phi^{i-1}) = \Theta(\phi^i)$ כאשר ϕ הוא יחס הזהב שהוא קטן ממש מ1.7 .

ובפרט $|T_L| = O(\phi^i)$.

T_R מקיים $|T_R| = 2^i - 1$ (ראינו בהרצאה, מתקבל על ידי סכימה של טור הנדסי כי בכל קומה פי 2 צמתים מהקומה הקודמת).

לכן $\Theta(|T_R|) = \Theta(2^i)$,

אבל $|T_L| = O(\phi^i)$ ולכן לא יתכן ש $|T_L| = \Omega(2^i)$ כפי שנדרש כדי שיתקיים $|T_L| = \Theta(2^i)$

.

סעיף ד

הפרכה :

נראה דוגמה נגדית. יהיו A, B שני עצי 2-3 בני n צמתים כל אחד, כך שעבור העץ A, לכל צומת שאינו עלה יש בדיוק 2 בנים, ועבור העץ B לכל צומת שאינו עלה יש בדיוק 3 בנים.

שני עצים אלו הם עצי 2-3 חוקיים העלולים להיווצר.

עבור העץ A, בכל רמה יש מספר כפול של צמתים מאשר ברמה הקודמת, ולכן גובה העץ הוא בדיוק $\lceil log_2(n) \rceil$.

באופן דומה, עבור העץ B בכל רמה יש פי 3 צמתים מאשר ברמה הקודמת ולכן גובה העץ הוא בדיוק $\lceil log_3(n) \rceil$.

לכן אם נסתכל על הפרש הגבהים של השורש של כל עץ, נראה כי הוא שווה ל $\lceil log_2(n) \rceil - \lceil log_3(n) \rceil$.

לכן $D(n)$, הפרש הגבהים המקסימלי בין העצים, מקיים $D(n) \geq \lceil log_2(n) \rceil - \lceil log_3(n) \rceil$.

נניח בשלילה כי $D(n) = O(1)$. אז קיימים $n_0, c > 0$ כך שלכל $n > n_0$:

$$\lceil log_2(n) \rceil - \lceil log_3(n) \rceil \leq D(n) \leq c$$

כלומר, גם עבור $n' = \max\{n_0+1, 2^{10^c}+1\} > n_0$ מתקיים:

$$log_2(2^{10^c}) - log_3(2^{10^c}) \leq log_2(n') - log_3(n') \leq \lceil log_2(n') \rceil - \lceil log_3(n') \rceil \leq D(n') \leq c$$

אבל $log_3(2^{10^c}) = \frac{log_2(2^{10^c})}{log_2(3)}$ כאשר $1.5 > log_2(3)$ ולכן $log_3(2^{10^c}) < \frac{log_2(2^{10^c})}{1.5}$
כלומר:

$$log_2(2^{10^c}) - log_3(2^{10^c}) > log_2(2^{10^c}) - \frac{2}{3}log_2(2^{10^c}) = 10log_2(2^c) - \frac{20}{3}log_2(2^c) = 10c - \frac{20}{3}c = \frac{10}{3}c$$

סה"כ נקבל:

$$c < \frac{10}{3}c < log_2(2^{10^c}) - log_3(2^{10^c}) \leq D(n') \leq c$$

כלומר $c < c$ בסתירה, ולכן $D(n) \neq O(1)$.

סעיף ה

תקציר הוכחה:

נראה שחסם עליון לזמן שלוקח לm פעולות עוקב הוא $O(m)$ ולכן הזמן המשוערך לפעולה אחת הוא $O(1)$.

נראה שמספר הצמתים בהם אנו עוברים במהלך m פעולות העוקב הוא $O(m)$ ובכל צומת שאנו עוברים בה עושים $O(1)$ פעולות בזמן קבוע ומכאן שהזמן לסך הפעולות הוא $O(m)$.

.

תהא הצומת הכי נמוכה שמתחתיה נמצאים m הצמתים הכי קטנים בעץ (הכוונה היא לשורש של העץ , ושהוא גם חלק מהm הצמתים), נסמן את תת העץ שהשורש שלו היא הצומת הזאת בT'.

נראה שגובה העץ T' הוא $O(logm)$:

כיוון ש בתת העץ השמאלי של T' יש לכל היותר m צמתים, גובה תת העץ השמאלי הוא $O(logm)$ וכיוון שהעץ T' הוא עץ AVL מתקיים שתת העץ הימני הוא בגובה של לכל היותר $O(logm)+1$ ומכאן שגובה תת העץ T' הוא $O(logm)+1 = O(logm)$.

במהלך הסיור נעבור על M צמתים כאשר $M = m + d$, כאשר d הוא מספר הצמתים שאנו עוברים דרכם אך הן לא חלק מm הצמתים הקטנים ביותר.

נגדיר "צומת מיותרת" כצומת שאנו עוברים במהלך הסיור אבל היא לא מm הצמתים הקטנים ביותר.

בכל צומת בה אנו עוברים נעשות לכל היותר 3 פעולות (מעבר לצומת שמאלית, ביצוע הפעולה על הצומת, מעבר לצומת ימנית), לכן הזמן שלוקח לביצוע m הפעולות הוא $t = O(m) + O(d)$.

נראה ש d הוא $O(logm) = $O(h), כאשר h הוא גובה העץ T':

בכל צומת מיותרת אנו עושים רק פעולה אחת שהיא פניה שמאלה כדי לעבור על צמתים עם מפתחות קטנים ממנה הנמצאים משמאלה. הצמתים מימינה לא יהיו חלק מהסיור כי היא בעצמה צומת שגדולה מ m ולכן אין בתת העץ הימני שלה צמתים קטנים או שווים מm .

נראה שבכל רמה בעץ T' יש לכל היותר צומת מיותרת אחת:

נניח בשלילה שיש 2 צמתים מיותרות באותה רמה שעוברים עליהן בלי להדפיס אותם, שתי הצמתים באותו עץ ולכן יש להן אב קדמון מינימלי המשותף לשתיהן כאשר אחת הצמתים נמצאת בתת העץ הימני שלו ואחת בתת העץ השמאלי שלו.

אם צומת האב המשותפת היא לא מיותרת, אז הצומת בתת העץ השמאלי היא גם לא מיותרת כי (היא מ m הצמתים הקטנים ביותר) וזאת סתירה לכך שהצומת השמאלית מיותרת.
אם צומת האב המשותפת מיותרת, אין סיבה שנסייר בצומת הנמצאת מימינה ולכן גם זו סתירה.

מכאן ש $d \leq h = O(h) = O(logm)$ ולכן
סיבוכיות הזמן של m הפעולות היא
$t = O(m) + O(d) = O(m) + O(logm) = O(m)$.
ולפי הגדרת סיבוכיות משוערכת , פעולה אחת מm הפעולות היא $O(\frac{m}{m}) = O(1)$.
■

שאלה 2

סעיף א

הסבר בקצרה:
נעביר את תתי העצים של העץ הקטן לעץ הגדול דרך העברת העצים אחד אחד(סה"כ 3 לכל היותר) ל"צומת קיצונית" מתאימה באותו גובה בעץ הגדול.
תשובה:
נתאר את האלגוריתם עבור המקרה בו T_1 הוא הגבוה מבין השניים (המקרים בהם T_2 גבוה או שגבהי העצים שווים עם רעיון דומה).

1)נמצא את גובה T_2 ונסמנו ב h_2 . (עלות $O(log(n_2)$)

2)נמצא את גובה T_1 ונסמנו ב h_1. (עלות $O(log(n_1)$)

3) נמצא את העלה הגדול ביותר בT_1, נסמן ערכו ב $t_{1,max}$. (עלות $O(log(n_1)$)

4) נעלה מ$h_2, t_{1,max}$ פעמים ונסמן את תת העץ של הצומת שהגענו אליה ב K.(עלות $O(log(n_2)$)

5)נעביר את הבן השמאלי של T_2 ל K , תוך הוספה של $t_{1,max}$ לערכי K כערך הימני ביותר (השמורה של סדר תתי העצים בעץ $2-3$ נשמרת כי $t_{1,max}$ גדול או שווה מכל הערכים בתת העץ הימני הקודם שהיה ב K כיוון שהוא הגדול בכל העץ באופן כללי).

6) נבצע פיצול אם נדרש ונתקן את T_1 כמו באלגוריתם הוספה רגיל. בסוף התיקון יתכן שגובה העץ T_1 גדל ב1 .

8) נחזור על שלבים 3-7 עד שנעביר את כל הבנים של T_2 ל T_1 (לכל היותר 3 פעמים, כמספר התתי עצים האפשריים לשורש של T_2 מעצם היותו עץ 2-3).

במהלך התהליך שמרנו שב T_1 כל העלים יהיו באותו גובה, כמו כן ששמורת ערכי הצמתים וסדר תתי העצים של עץ $2-3$ תשמר ולכן T_1 בסוף האלגוריתם נשאר עץ $2-3$. כל העלים של T_2 עברו לT_1 ולכן T_1 הוא עץ 2-3 המכיל את כל הערכים של העצים T_1 ו T_2.

נציין כי גובה העץ המאוחד הוא לכל היותר גבוה ב1 מגובהו המקסימלי מבין שני העצים, זאת כיוון שפיצול של השורש של העץ הנוצר יכול להתבצע בשלב שקורה רק 3 פעמים, ואם התבצע איחוד בשלב כלשהו בשניים הבאים אחריו לא יקרה עוד פיצול(כי אחרי הפיצול ידרשו 2 הוספות לפחות כדי להגדיל את מספר הערכים בשורש מ1 ל2 ,ומצב זה לא דורש פיצול אלה רק אם מספר הערכים בשורש מגיע ל3) .

ביצענו מספר קבוע של פעולות שסיבוכיות כל פעולה היא $O(max\{log(n_1), log(n_2)\})$ ולכן סיבוכים האלגוריתם היא $O(max\{log(n_1), log(n_2)\})$ כנדרש.

סעיף ב

נניח בה"כ כי $h_2 > h_1$. נרצה למזג את T_1 לתוך T_2 על ידי הוספת השורש של T_1 במיקום מתאים ב T_2. נרצה שכל העלים של העץ החדש יהיו באותה רמה, ולכן נרצה שהשורש של T_1 יוצב בעומק $h_2 - h_1$ בעץ.

כל הצמתים בT_1 קטנים מכל הצמתים בT_2 ולכן נלך שמאלה מתי שאפשר כדי להגיע לצומת המינימלי בעץ T_2 בעומק $h_2 - h_1$. מספר הפעולות על מנת להגיע לצומת זה הוא $O(h_2 - h_1)$.

על מנת להציב את השורש של T_1בעומק זה, נוסיף אותו כבן של ההורה של הצומת שמצאנו. בהורה נצטרך להוסיף אינקדס חדש שיהיה אינדקס השמאלי. נתון M הצומת המקסימלי ב T_1ו m הצומת המינימלי בT_2.

ניקח $m < k \leq M$ כלשהו שישמש כאינקדס - הוא גדול ממש מכל צומת בתת העץ השמאלי החדש שאנחנו מוסיפים כיוון שזה T_1 וקטן או שווה לצמתים בשאר הבנים כיוון שהם היו במקור בT_2.

כמו בהוספה רגילה לעץ 2-3 , אם השורש התווסף כאח ל3 בנים, יופר אחד מהתנאים לקיום עץ 2-3 ולכן נידרש לפצל צמתים במסלול מהצומת החדש שהוספנו למעלה אל השורש של T_2, כלומר לכל היותר $h_2 - h_1$ פיצולים שכל אחד מהם מתבצע ב $O(1)$.

נשים לב כי אם $h_2 = h_1$ אין צומת מעל המיקום בו נרצה להציב את השורש של T_1 ולכן ניצור שורש חדש לעץ הממוזג, שיכלול רק את האינדקס k. בנו השמאלי יהיה השורש של T_1 ובנו הימני יהיה השורש המקורי של T_2.

לכן גם כאשר $h_2 - h_1$ מתבצעות פעולות ולכן סיבוכיות המיזוג היא $O(h_2 - h_1 + 1)$.

הפעולה תפעל באופן זהה גם אם $h_1 > h_2$, למעט שT_2 יתמזג לT_1 מצידו הימני, וk יצטרף כאינקדס ימני בצומת אליו הוא מתווסף.

מהתייחסות לשני המקרים נובע כי סיבוכיות המיזוג תהיה $O(|h_2 - h_1| + 1)$.

סעיף ג

רעיון ההוכחה:

נאחד זוגות עצים באופן הבא:

נאחד(בעזרת הפעולה $Join$ מסעיף ב' שהנחותיה מתקיימיות) את T_1 עם T_2 ונסמן את העץ המאוחד ב T_1' , נאחד את T_3 עם T_1' ונסמן את העץ המאוחד בT_2' , נאחד את T_4 עם T_2' ונסמן את העץ המאוחד ב T_3' ,..., נמשיך כך עד שנאחד את העץ הT_k עם העץ T_{k-2}' ונסמן ב T את העץ המאוחד הכולל, עץ $2 - 3$ המכיל את כל המפתחות של כל העצים. תנאי הסיבוכיות יתקיים בגלל שיווצר טור טלסקופי של הפרשי הגבהים שיתכנס ל$h_k - h_1$,הסיבוכיות תהיה $h_k - h_1 + k$ כי בוצעו $k - 1$ פעולות כאלה .

הוכחה:

(1

נאחד את T_1ו T_2 בעזרת הפעולה $Join$ שהנחותיה מתקיימות כי $h_1 \leq h_2$ ובנוסף ערכי המינימום והמקסימום של העצים ידועים.

הסיבוכיות של הפעולה היא $O(h_2 - h_1 + 1)$ כי אם $h_2 - h_1 \geq 1$ הסיבוכיות היא $O(h_2 - h_1)$ אחרת $O(1)$.

נסמן העץ הנוצר מהאיחוד ב T_1' הוא בגובה $h_2 + 1$ לכל היותר (כפי שהוכח בסעיף א) וh_2 לכל הפחות.

(2

נאחד את T_3 עם T_1' .

אם גובה העץ T_3 גדול משל T_1' סיבוכיות האיחוד לפי $Join$ מסעיף ב' היא $O(h_3 - h_2 + 1)$.

אם גובה העץ T_3 הוא קטן או שווה לגובה של T_1', הפרש הגבהים יכול להיות לכל היותר 1 ואז סיבוכיות האיחוד היא $O(1)$.

בשני המקרים גובה העץ הוא לכל הפחות h_3 .

נסמן העץ הנוצר ב T_2' , זהו עץ 2-3 המכיל את כל המפתחות של T_1, T_2, T_3 .

.

.

i) נמשיך כך כשבאופן כללי בצד הi:

נאחד את T_{i+1} עם T_{i-1}' .

אם גובה העץ T_{i+1} גדול מT_{i-1}' סיבוכיות האיחוד היא לפי $Join$ מסעיף ב' היא $O(h_{i+1} - h_i + 1)$.

אחרת הפרש הגבהים הוא לכל היותר 1 ולכן סיבוכיות האיחוד היא $O(1)$.

העץ הנוצר T_i' הוא עץ 2-3 המכיל את כל המפתחות של $i+1$ העצים הקטנים וגובהו לכל הפחות h_{i+1} .

$k - 1$)נמשיך כך עד שנגיע לצעד ה$k - 1$:

נאחד את T_k עם T'_{k-2} בסיבוכיות $O(h_k - h_{k-1} + 1)$.
נסכום את הסיבוכיות בשלבים 1 עד $k-1$:

$$T = O(h_2-h_1+1)+O(h_3-h_2+1)+...O(h_k-h_{k-1}+1) \underbrace{=}_{\text{sum telescopic}} O(h_k-h_1+k)$$

סעיף ד

ניצור שתי רשימות ריקות של מצביעים לצמתים בעץ, כאשר מאחת מהן יווצר העץ הקטן יותר ומהשנייה יווצר העץ הגדול יותר. נחפש את המפתח x בעץ.
עבור כל צומת במסלול החיפוש:

- אם עברנו לבן השמאלי, נוסיף את שני הבנים שמימינו לרשימה של העץ הגדול יותר, כיוון שתתי העצים שהם השורשים שלהם כוללים רק צמתים גדולים יותר מx.
- אם עברנו לבן הימני, נוסיף את שני הבנים שמשמאלו לרשימה של העץ הקטן יותר, כיוון שתתי העצים שהם השורשים שלהם כוללים רק צמתים קטנים יותר מx.
- אם עברנו לבן אמצעי, נוסיף את הבן שמימינו לרשימה של העץ הגדול יותר, ואת הבן שמשמאלו לרשימה של העץ הקטן יותר, מאותם נימוקים.

הסריקה תפסיק כאשר נגיע לרמת העלים.
כעת באחת מהרשימות יש את כל חלקי העץ שגדולים מx, ובשנייה את כל חלקי העץ שקטנים או שווים לx.
נשים לב כי מתקיימים כל התנאים של הפונקציה מסעיף 3:

- ידועים הגבהים של כל אחד מהעצים ברשימה: אם לא ידוע הגובה של העץ המקורי ניתן לעבור עליו פעם אחת ולספור אותו. מעבר על גובה העץ הוא ב $O(log(n))$ בעץ מאוזן ולכן ניתן לעשות זאת בסיבוכיות הנדרשת. לאחר מכן, עבור כל עץ שנכנס לאחת מהרשימות, גובהו הוא הגובה של כל העץ פחות מספר העומק של הצומת שמתווסף לרשימה, אותו ניתן לספור.
- ניתן לסדר את את העצים בכל רשימה כך שלכל עץ, כל הצמתים בו קטנים יותר משל העצים שאחריו וקטנים יותר משל העצים שלפניו או להפך. זאת כיוון שכל הוספה מתבצעת לאחר צעד שעשינו במסלול חיפוש בעץ, כלומר כל תת עץ שמתווסף לרשימה הוא מעומק גבוה יותר מקודמיו. לכן כל עץ שמתווסף לאחת הרשימות הוא "קיצוני" יותר - לרשימת העצים הגדולים יותר מx נוסיף עצים בעלי צמתים גדולים יותר ויותר ולרשימת העצים הקטנים שווים x נוסיף עצים קטנים יותר ויותר.
- ידועים הערכים המקסימלים והמינימלים בכל עץ: מהמימוש לסעיפים 2 ו 3, למעשה נדרש לדעת רק חסם תחתון עבור ערכי העצים בעלי הצמתים הגדולים יותר, וחסם עליון עבור ערכי העצים בעלי הצמתים הקטנים יותר. כיוון שכל עץ שמתווסף לאחת הרשימות הוא "קיצוני" יותר, בין כל שני עצים צמודים ברשימה ניתן לדעת ערך משותף הגדול יותר מכל ערכי העץ הקטן יותר והקטן יותר מכל ערכי העץ הגדול יותר. זהו האינדקס של צומת האב אותו עזבנו כשהוספנו כל עץ לרשימה. במקרה שלאב יש שני אינקדסים, ניתן לקחת את הימני אם עברנו שמאלה, ואת השמאלי אם עברנו ימינה.
- ניתן לסדר את גבהי העצים, ואין שלושה עצים עם אותו גובה: כפי שהוזכר קודם, כל עץ נלקח מעומק גדול יותר כיוון שהוא מתווסף לרשימה לאחר צעד נוסף במורד העץ. לכל היותר, אם עזבנו צומת בעל שלושה בנים ולא עברנו לבן האמצעי, יתווספו לאותה רשימה

שני האחים הנותרים שהם בעלי אותו גובה. לכן לא יהיו באף אחת מהרשימות יותר משני עצים בעלי אותו גובה.

כיוון שמתקיימים התנאים הללו ניתן להשתמש בפונקציה מסעיף 3 עבור כל אחת מהרשימות, שתאחד אותן לעץ - בעץ אחד יהיו כל המפתחות הגדולים מx ובשני את כל המפתחות הקטנים שווים לx, כנדרש.

הפונקציה פועלת ב$O(h_k - h_1 + k)$. לכל עץ ברשימה, מתקיים כי הוא תת עץ של עץ בגובה $O(log(n))$ ולכן גם גובהו הוא $O(log(n))$. k הוא מספר העצים בכל רשימה. כיוון שמתווסף מספר קבוע של עצים לכל אחת מהרשימות בכל צעד, ומספר הצעדים לאורך עץ 2-3 הוא $O(log(n))$, k הוא $O(log(n))$.

בנוסף להפעלת הפונקציה מתקיים גם הסיור עצמו על העץ, אבל כאמור גם הוא $O(log(n))$ ולכן בסך הכל נקבל כי סיבוכיות הזמן של הפעולה היא $O(log(n))$ כנדרש.

שאלה 3

סעיף א

הסבר כללי:

נשתמש בעץ דרגות של הימים, ממוין לפי מספר היום כשכל צומת תכיל :

day(מפתח) - מספר היום

sick_today(ערך) - מספר החולים ביום

n - מספר הצמתים בתת העץ שהצומת היא שורשו (בשביל לאפשר פעולות בסיסיות של rank_tree בהמשך.

worst_under- מספר החולים הגבוה ביותר בתת העץ והיום בו זה קרה כזוג סדור פוינטר לצומת בעץ בה ($sick, day$) כאשר $sick$ הוא מספר החולים הגדול ביותר וday הוא היום בו זה קרה. נגדיר יחס סדר בין זוגות כאלה , קודם כל לפי האנדקס $sick$ ו אז לפי day. יחס הסדר יאפשר תיקון מידע כאשר יבוצעו גלגולים.

ומצביעים לצומת מימינה ומשמאלה, $left$, $right$.

פעולות:

- $SetPositiveInDay(d, x)$

הכנס את היום לעץ עם הערך המתאים ועדכן את הדרגה וworst_under בצמתים שבמסלול החיפוש (אם התבצע גלגול בהכנסה ניתן לתקן את המידע הנוסף בהתאם כפי שנלמד בהרצאה כיוון שהמידע בכל צומת ניתן לשחזור בעזרת המידע הנוסף בעזרת הצמתים מימינו ומשמאלו (שבביצוע גלגול יחיד לא נפגמים) וערך ה $sick$ שלו עצמו.

- $WorseBefore(d)$

אתחל מחסנית ריקה s.

חפש את היום בעץ כשבכל פניה ימינה במסלול החיפוש הכנס את הצומת בה נעשתה הפניה למחסנית.

(המחסנית שומרת צמתים שהם ותת העץ הימני שלהם מכילים ימים קטנים מd , ורק בתתי העצים שלהם נעשה את החיפוש מאוחר יותר. סדר הכנסת האיברים למחסנית מבטיח שהיא ממוינת לפי ימים)

אם d לא נמצא , החזר -1 וסיים.

אם d נמצא, סמן צומת זו ב v, והכנס למחסנית את $v.left$.

אם v.left->worst_under.sick>v.sick_today (יש יום עם מספר חולים גדול יותר בתת העץ השמאלי של v ולכן בעל יום קטן יותר, המחסנית עבור מקרה זה מיותרת) :

החזר $SearchLastDayBiggerThen($v.left,v.sick_today$)$(פונקציית עזר שתוגדר בהמשך*).

אחרת, נעבור לחיפוש היום דרך המחסנית s באופן הבא :

נבצע את הלולאה הבאה(גודל המחסנית הוא לכל היותר $O(logn)$, כעומק העץ ובכל איטרציה אנו מרוקנים ממנו איבר לכן הלולאה תתבצע לכל היותר $O(logn)$ פעמים) :

}

אם המחסנית ריקה : החזר -1 וסיים(לא קיים יום לפני $v.day$ עם מספר חולים גדול מv.sick_today).

$x =$ s.pop()

אם x.sick_today>v.sick_today , החזר את $x.day$ (כיוון שבצמתים מימינו שללנו קיום של יום שקטן מ d עם מספר חולים גדול יותר , הוא בהכרח היום האחרון שבו מספר החולים היה גדול מהיום d).

אחרת, אם x.left->worst_under.sick>v.sick_today (יש יום עם מספר חולים גדול יותר בתת העץ השמאלי של x ולכן בעל יום קטן יותר, מעתה המשך המחסנית מיותר ולא נעבור עליה) :

החזר $SearchLastDayBiggerThen($x.left,v.sick_today$)$(פונקציית עזר שתוגדר בהמשך*, פונקציה שבהכרח תחזיר תשובה כי היום שאותו אנחנו מחפשים נמצא בתת העץ של $x.left$).

(אם לא, המשך הלאה לאיבר הבא במחסנית)

}

<u>**הסבר סיבוכיות ל**$WorseBefore(d)$ **:**</u>

חיפוש הצומת עם המפתח d נעשה ב $O(logn)$ כחיפוש בעץ חיפוש מאוזן.

כדי למצוא את הצומת שמתחתיה יש את היום המבוקש אנו עוברים על מחסנית שגודלה לכל היותר גובה העץ ולכן $O(logn)$, ברגע שנמצאה אנו קוראים לפונקציה $SearchLastDayBiggerThen$ שסיבוכיותה היא $O(logn)$ ובקבלת ערך חזרה ממנה נסיים את התוכנית. (וזה אומר שהיא נקראת רק פעם אחת) .

מכאן ש סיבוכיות $WorseBefore(d)$ היא $O(logn)$.

<u>פונקצית עזר לחיפוש יום גדול ביותר שערכו גדול מערך מסוים בעץ שידוע שיש בו יום כזה :</u>

$SearchLastDayBiggerThen(T, \text{sick\_num})$: (מקבלת עץ שמובטח שיש בו צומת שערכה גדול מ sick_num ומחזירה את) (סיבוכיות כעומק העץ T, בכל רמה אנו עוברים פעם אחת לכל היותר)

נבצע סיור בT מהשורש באופן הבא :

נסמן את הצומת שבה אנו נמצאים כרגע בx

פנה ימינה כל עוד $x.right- > worst\_under.sick > \text{sick\_num}$ (יש בתת העץ הימני ימים עם מספר חולים גדול משל v)

אחרת , אם x.sick_today>sick_num **החזר את** $x.day$**(בהכרח תנאי זה יתקיים עבור צומת כלשהי במהלך הסיור)**.

אחרת, פנה שמאלה.

סעיף ב

<u>הסבר בקצרה :</u>

נשתמש במבנה שהוגדר בסעיף א' , עצם היותו עץ דרגות יאפשר לנו לדעת אם הוכנסו כל הימים בין שתי ימים בעזרת הפונקציה $rank$ שנלמדה בהרצאות. נעזר בפונקציה $WorseBefore$ מסעיף קודם כדי לבדוק אם d_1 הוא $WorseBefore$ של d_2 , ובהנתן המידע מ$rank$ של כמה ימים הוכנסו ביניהם נוכל לדעת איזה ערך להחזיר לפי דרישות הפונקציה.

<u>תשובה :</u>

נשתמש באותו מבנה מסעיף א ונוסיף את הפעולה $IsWorstSince(d_1, d_2)$:

-$IsWorstSince(d_1, d_2)$

res_d=WorseBefore(d2) (נעשה ב$O(logn)$)

אם res_d!=d1 : החזר $False$ (res_d הוא יום גדול מ d_1 שבו החולים מאשר בd_2 ומספרו קטן מd_2 ║ ש d_1 הוא בכלל לא יום שמספר החולים בו גדול ממספר החולים בd_2)

אחרת,(res_d $== d1$ ולכן הוא היום האחרון לפני d שבו מספר החולים גדול מd_2 מבין הימים שהוזנו)

נבדוק האם יש ימים שעוד לא הוזמנו ביניהם : העץ שמחזיק את המידע הוא עץ דרגות ולכן ניתן לממש עבורו פונקצית $rank(d)$ כפי שנלמד בהרצאה ובתרגול , שתחזיר במקרה שלנו את מספר הימים עד d כולל שהוכנסו.

אם $rank(d_2) - rank(d_1) == d_2 - d_1$ (זה אומר שהוכנסו כל הימים בין שני הימים) :

החזר $True$.

אחרת :

החזר $Unknown$ (כי יש ימים בין שני הימים שלא הוכנסו עדיין למערכת).

<u>סיבוכיות :</u>

הפעולות הכבדות שמתבצעות ה WorseBefore, ו $rank$, שתיהן נעשות ב $O(logn)$ ולכן הסיבוכיות היא $O(logn)$.

שאלה 4

חלק 1.

הסבר בקצרה:
נשתמש בעץ AVL כדי לאפשר הכנסה למבנה ב$O(logn)$ וברשימה מקושרת כדי להגיע לאיבר הבא ב$O(1)$ תוך שמירה על שמורותיהם בכל שלב.
תשובה:
שדות המבנה יהיו:
$list$ -רשימה מקושרת דו כיוונית ממוינת של תחנות העצירה. התחנות מיוצגות על ידי מספרים שלמים.
$tree$-עץ AVL של תחנות העצירה. איבר בעץ יכיל מפתח שהוא מספר הקומה , והערך שלו יהיה פוינטר לאיבר ברשימה המקושרת.
$level$ - פוינטר לאיבר ברשימה המציין את הקומה הנוכחית.
פעולות:
- $Init$
אתחול רשימה מקושרת עם איבר אחד שערכו 0 וקביעת ערך $level$ להיות כתובת של האיבר.
-addStop(k)
חפש את k בעץ $.tree$ סיבוכיות: חיפוש ב$AVL-$ בעל n איברים $O(logn)$.
אם נמצא סיים.
אם לא נמצא,
זכור את הצומת האחרונה במסלול החיפוש, הוסף איבר בעל ערך k לעץ ובצע גלגול מתאים.
אם הצומת האחרונה במסלול החיפוש היא בעלת מפתח גדול מk , הוסף את k לרשימה לפני איבר זה .
אחרת(הצומת האחרונה במסלול החיפוש היא בעלת מפתח קטן מk), הוסף את k לרשימה $list$ אחרי האיבר.
(נציין שפוינטר לאיבר נמצא בצומת ולכן ניתן להוסיף את k לרשימה ב$O(1)$ ולשמור עליה ממוינת).
סה"כ סיבוכיות זמן נקבע לפי הפעולה הקבצה שהיא פעולת החיפוש של הצומת בעץ: $O(logn)$
.

-NextStop()
אם $level$ מצביע על איבר אחרון ברשימה סיים(השאר בקומה הנוכחית).
אחרת, קדם את $level$ להצביע על האיבר הבא ברשימה $list$.
הדפס את הערך של האיבר ברשימה עליו $level$ מצביע.
מספר קבוע של פעולות $O(1)$ ולכן סיבוכיות זמן $O(1)$.

חלק 2.

הסבר בקצרה:
נשתמש במערך דינאמי שייצג את הקומות בהן יש עצירות, מההנחה של הקומה הגבוה ביותר שהמעלית עתידה לעצור נובע (נוכיח זאת) שההפרש בין שתי תחנות עוקבות "לא רחוק" בדר"כ ובגלל תכונה זו נעמוד בסיבוכיות המשוערכת.
הגדרת המבנה
שדות המבנה:
$level$ - מספר הקומה בה כרגע אנחנו נמצאים
$highest\_level$ - מספר הקומה הגדולה ביותר שיש בה עצירה ברגע נתון

num_of_stops - מספר הפעמים שהפונקציה AddStop נקראה
stop_level_arr - מערך דינמי של בוליאנים , במקומות בהן יש תחנת עצירה יסומן 1 , אחרת 0. באופן דיפולטיבי הערכים יהיו 0.
arr_size - גודל המערך stop_level_arr.

<u>פעולות :</u>

- $Init$

אתחול המערך stop_level_arr למערך בגודל 2 .

$level = 0$

highest_level=0

num_of_stops=0

-AddStop(k)

num_of_stops++

אם 2*num_of_stops>=arr_size : הגדל את המערך stop_level_arr פי 2 .(נשמור על גודל המערך תמיד גדול או שווה ל$2 * num_of_stops$.

אם highest_level<k : highest_level=k

stop_level_arr[k]=1

-NextStop()

אם highest_level==level : סיים והדפס את הקומה הנוכחית.

אחרת :

התקדם במערך stop_level_arr מהאיבר ה level עד האיבר הראשון במערך שערכו הוא 1 ומסמל תחנת עצירה וסמנו ב found_level.

הדפס את תחנת העצירה שנמצאה , בצע השמה level=found_level וסיים.

(סיבוכיות הפעולה היא $O(n_i)$ כאשר n_i הוא ההפרש בין הקומה הקודמת לבאה)

ניתוח סיבוכיות זמן משוערכת :

יהו m פעולות $AddStop(k_i)$ ו $NextStop()$. נסמן את הזמן שלוקח לביצוע כולן ב T .
נראה $T = O(m)$ ומכך ינבע ש $\frac{T}{m} = O(1)$ ולכן הסיבוכיות המשוערכת לפעולה תהיה $O(1)$.
נסמן מספר פעולות $AddStop$ ב a ו מספר פעולות $NextStop$ ב n כך ש :

$m = n + a$

העלות הכוללת של פעולות n היא $\sum_{i=1}^{n} n_i \leq 2a = O(a)$, כיוון שסכום ההפרשים בין שתי קומות הוא לכל היותר $2a$ (מהנתון שהקומה הבאה שנקראת תמיד קטנה ממספר הפעמים שהוכנסו קומות חדשות) וn_i הוא ההפרש בין שתי קומות עוקבות .

העלות המשוערכת של פעולות a היא $O(1)$ כפי שנלמד בהרצאה (הכנסה למערך דינמי, ההגדלה של המערך מחושבת במחיר) ולכן עלות סך פעולות מסוג a הוא $\sum_{i=1}^{a} a_i = O(a)$ גם כן.

מכאן ש $T = \sum_{i=1}^{n} n_i + \sum_{i=1}^{a} a_i = O(a) + O(a) \underbrace{=}_{a \leq m} O(m)$

שאלה 5

נפתור באמצעות שימוש במבני $trie$, שכן הם יאפשרו לנו להשוות בין מחרוזות ארוכות בסיבוכיות התלויה רק באורכיהן.

בנוסף לפונקציונליות הרגילה של $trie$, כל צומת במבנה יכיל גם את מספר המילים הקיימות בתת ה$trie$ שמתחיל ממנו.

$trie$ אחד ידאג למילים שלפני המקף , והשני ידאג למילים שאחרי המקף.

לשם כך נגדיר מבנה $trie$, המכיל את השדות הבאים:

$children$: מערך של n מצביעים מסוג $trie$ המייצג עבור כל צומת את המשכי המילים שהרישא שלהן היא המסלול שעברנו עד עכשיו, כאשר n הוא גודל הא"ב Σ.

$subCount$: מספר המילים שהרישא שלהן היא המסלול שעברנו עד עכשיו, על פני כל אותיות ההמשך האפשריות.

$endCount$: מספר הפעמים שהוכנסה מחרוזת שמסתיימת בצומת הנוכחי. הכרחי על מנת לספור את המילים שהן רישא ממש של המילה הרצויה ולכן קטנות לקסיקוגרפית ממנה.

ואת המתודות הבאות:

$Init()$: יוצרת צומת $trie$ חדש, בעל מערך ריק של ילדים ומאתחלת את $subCount$ להיות 1 ואת $endCount$ להיות 0. מהנלמד על יצירת מערך, ניתן לאתחל את הצומת ב $O(1)$.

$getIndex(ch)$: מנרמלת אות ch בΣ לערך בין 0 ל$n-1$, שהוא האינדקס המזוהה איתה במערך $children$. מניחים כי קיים סידור לא"ב, ושומרים על סידור זה גם באינקדסים. לדוגמה, אם Σ הוא האותיות הקטנות בא"ב האנגלי, a תקבל את הערך 0, b את הערך 1, עד לz שתקבל את הערך 25.

סיבוכיות הזמן היא $O(1)$, שכן ניתן אפילו להשוות את האות המתקבלת לכל ערך אפשרי שלה מתוך גודל סופי וידוע מראש של הא"ב, ולהחזיר את האינדקס המתאים.

$Insert(w)$: עובדת כמו הכנסה רגילה של $trie$ - מתחילים מהשורש, ועבור כל אות הבאה במילה שרוצים להכניס מנסים לעבור לצומת שלה במערך $children$ לפי האינדקס שמוחזר על ידי $getIndex$ עבור האות. אם הצומת לא קיים, יוצרים אותו.

בכל פעם שעוברים בצומת במהלך ההכנסה, מקדמים את $subCount$ שלו באחד. זאת כיוון שלאחר ההכנסה תהיה מילה אחת נוספת שמתחילה בכל רישא שעליה אנחנו עוברים. כל עוד אנחנו עוברים על מסלול קיים, ה$subCount$ של כל צומת יגדל באחד, וברגע שנתפצל ונתחיל ליצור צמתים, ה$subCount$ שלהם יוגדר להיות 1.

עבור הסמל האחרון האחרון במחרוזת, כלומר הצומת האחרון אליו אנחנו מגיעים, נגדיל את $endCount$ באחד.

במבנה הראשי, נשתמש במתודה זו רק מתוך השורש.

השינוי לא משפיע על הסיבוכיות כיוון שבכל צומת מבצעים לכל היותר שתי פעולות מעבר להכנסה רגילה, ולכן כמו עבור $trie$ רגיל סיבוכיות הזמן תהיה $O(m)$ כאשר m הוא אורך המחרוזת המוכנסת.

$getSubCount()$: מחזירה את $subCount$ של הצומת הנוכחי ב$O(1)$.

עבור הבעיה הנתונה אין צורך לממש הסרה מה$trie$.

נגדיר את המבנה הראשי בבעיה, שמכיל את השדות הבאים:

m - הפרמטר המתואר בשאלה.

$prefix$ **-** $trie$ המכיל את כל המחרוזות שהוכנסו והופיעו לפני המקף בצמד שלהן.

$suffix$- $trie$ המכיל את כל המחרוזות שהוכנסו והופיעו אחרי המקף בצמד שלהן.

ואת המתודות הבאות:

$Init(m)$: מאתחלת את המבנה עם שני צמתי $trie$ ריקים, ומאתחלת את הפרמטר m להיות הערך המתקבל. אתחולים אלה מתבצעים ב$O(1)$.

$Insert(w_1, w_2)$: מכניסה את המילה w_1 ל$prefix$, ואת המילה w_2 ל$suffix$ באמצעות פונקציית $Insert$ של $trie$. לפי הידוע על $trie$, ההכנסה ל$prefix$ תתבצע בסיבוכיות $O(|w_1|)$ וההכנסה ל$suffix$ תתבצע בסיבוכיות $O(|w_2|)$. שתי פעולות אלה יתבצעו בכל קריאה ל$Insert$, ולכן סיבוכיות הפעולה היא $O(|w_1| + |w_2|)$

$Magic(w_1, w_2)$: מתחילה בסריקה של $prefix$.

עבור כל צומת במסלול החיפוש למעט האחרון של w_1 ואות נוכחית של המילה ch, נוסיף למשתנה סכימה את $subCount$ של כל אחד מהצמתים ב$children$ באינדקסים $\{0, \ldots, getIndex(ch) - 1\}$ כאשר אם $getIndex(ch) == 0$ לא נעבור על המערך.

זה יספור בכל שלב בחיפוש את כל המילים במבנה שמתחילות באותה רישא עליה עברנו עד עכשיו, אבל שהמשכן קטן ממש לקסיקוגרפית מהמשך המילה.

אם אנחנו לא בצומת האחרון בחיפוש, נרצה לספור גם את כל המילים שהן רישא ממש של המילה אותה אנחנו מחפשים ולכן נוסיף למשתנה הסכימה את $endCount$ של הצומת הנוכחי לפני שנעבור לצומת הבא.

אם בשלב מסוים ננסה לעבור לצומת שאינו קיים, סימן שהמילה לא מופיעה במבנה ולכן נחזיר מיידית $false$.

לאחר שנסיים את הסריקה של המילה משתנה הסכימה יחזיק את מספר המילים במבנה שקטן ממש לקסיקוגרפית מw_1. אם מספר זה קטן ממש מm, נחזיר $false$. אחרת, נאפס את משתנה הסכימה ונעבור לסרוק את $suffix$.

עבור כל צומת במסלול החיפוש של w_2 ואות נוכחית של המילה ch, נוסיף למשתנה הסכימה את $subCount$ של כל אחד מהצמתים ב$children$ באינדקסים $\{n-1, \ldots, getIndex(ch) + 1\}$ כאשר אם $getIndex(ch) == n - 1$ לא נעבור על המערך.

זה יספור בכל שלב בחיפוש את כל המילים במבנה שמתחילות באותה רישא עליה עברנו עד עכשיו, אבל שהמשכן גדול ממש לקסיקוגרפית מהמשך המילה.

כאשר הגענו לצומת האחרון במסלול החיפוש של w_2, נוסיף את ה$subCount$ של כל אחד מהאיברים הקיימים ב$children$ כיוון שכל המילים שw_2 היא רישא ממש שלהן גדולות ממש ממנה לקסיקוגרפית.

כמו קודם, אם בשלב כלשהו בחיפוש ננסה לעבור לצומת שאינו קיים נחזיר $false$.

לאחר הסריקה של w_2, משתנה הסכימה יחזיק את מספר המילים במבנה שגדולות ממש לקסיקוגרפית ממנה. אם ערך זה קטן מm נחזיר $false$, אחרת נחזיר $true$.

כידוע על $trie$, סיבוכיות מעבר שלם על מילה w היא $O(|w|)$. בכל שלב בחיפוש אנחנו מבצעים $O(1)$ פעולות - המערך $children$ הוא בגודל של מספר המילים בא"ב הסופי שנחשב גודל קבוע ולכן מעבר על חלק ממנו וביצוע פעולות כמו עדכון משתנה הסכימה הוא $O(1)$.

אנחנו מבצעים לכל היותר מעבר על כל w_1 וכל w_2 ולכן סיבוכיות הפעולה היא $O(|w_1|+|w_2|)$.