תרגול יצירת מבנה נתונים

בכל השאלות הבאות יש לתאר את מבנה הנתונים, לתאר את הפעולות השונות ולהסביר את זמן הריצה.

תרגיל 1

כיצד תממשו מבנה נתונים שתומך ב:

- הכנסת זוגות מספרים (a, b) ב$O(\log n)$
- הוצאת זוגות מספרים (a, b) ב$O(\log n)$
- חיפוש לפי הקוואורדינטה הראשונה ב$O(\log n)$
- חיפוש לפי הקוואורדינטה השניה ב$O(\log n)$
- מעבר על הזוגות ממוינים לפי הקוא' הראשונה\השניה ב$O(n)$

תרגיל 2

הציעו מבנה נתונים המאפשר ביצוע הפעולות הבאות בזמן הריצה הנדרש:

Init()	אתחול מבנה הנתונים.	$O(1)$
Insert(x)	הכנסת הערך x.	$O(\log n)$
Search(x)	בדיקה האם האיבר x נמצא במבנה הנתונים.	$O(\log n)$
Delete(x)	מחיקת האיבר x.	$O(\log n)$
Max_Gap()	הפונקציה מחזירה את ההפרש המקסימאלי בין שני מספרים כלשהם במבנה הנתונים, ז"א את הערך $\max_{x,y\in S}\{\|x-y\|\}$.	$O(1)$
Min_Gap()	הפונקציה מחזירה את ההפרש המינימאלי בין שני מספרים כלשהם במבנה הנתונים, ז"א את הערך $\min_{x,y\in S}\{\|x-y\|\}$.	$O(1)$

תרגיל 3

הציעו מבנה נתונים S עבור קבוצה דינאמית של איברים התומך בפעולות הבאות בזמנים הנדרשים (n מציין את מספר האיברים במבנה, כל המפתחות שונים):

פעולה	תיאור	זמן
Build(S,x)	בניית המבנה S מתוך סדרה של n מפתחות. **כאשר $x \in S$ הינו החציון של קבוצה S.**	$O(n)$ **במקרה הגרוע**
Insert(S,k)	הכנסת המפתח k למבנה S.	$O(\log n)$ **במקרה הגרוע**
Del-Median(S)	מחיקת החציון של המפתחות ב - S. תזכורת: חציון הוא האיבר אשר מחצית האיברים קטנים ממנו, ומחציתם גדולים ממנו.	$O(\log n)$ **במקרה הגרוע**
Del-New(S)	מחיקת האיבר שנכנס אחרון לתוך המבנה S.	$O(\log n)$ **במקרה הגרוע**

הערה: ניתן להניח כי כל האיברים שמהם בונים את המבנה נתונים S בפונקציה Build(S) נכנסים באותו הזמן.

בנוסף, ניתן להניח כי אם יש כמה איברים עם אותו זמן הכנסה, אזי בפונקציה Del_New(S) נמחק שרירותית אחד מהם.

תרגיל 4

הציעו מבנה נתונים המכיל מספרים שלמים ומאפשר ביצוע הפעולות הבאות בזמן הריצה הנדרש:

Insert(x)	הכנסת הערך x. (אם x קיים כבר ההכנסה נכשלת.)	$O(\log n)$	n- מספר האיברים במבנה נתונים
Search(x)	בדיקה האם האיבר x נמצא במבנה הנתונים.	$O(\log n)$	
Delete(x)	מחיקת האיבר x. (אם x אינו קיים המחיקה נכשלת.)	$O(\log n)$	
ComplementPair()	הפונקציה מחזירה זוג מספרים (כלשהו) במבנה הנתונים שסכומם הוא 1. אם לא קיים זוג כזה, הפונקציה מחזירה null	$O(1)$	
HasComplement (x)	האם קיים איבר y במבנה כך ש- $x+y=1$. אם כן, החזר True אחרת, החזר False.	$O(\log k)$	k - מספר הזוגות במבנה שסכומם 1. x- איבר שנמצא במבנה.

תרגיל 5

הציעו מבנה נתונים עבור קבוצת איברים בעלי מפתחות מספרים שלמים. על מבנה הנתונים לתמוך בפעולות הבאות:

פעולה	תיאור	זמן (במקרה הגרוע)
Init ()	אתחול מבנה נתונים. בהתחלה המבנה ריק (לא מכיל איברים).	$O(1)$
Insert (k)	הכנסת איבר חדש בעל מפתח k.	$O(\log n)$
Delete(k)	מחיקת האיבר בעל מפתח k מהמבנה.	$O(\log n)$
PairSum(k)	מציאת שני איברים (שונים) במבנה כך שסכום המפתחות שלהם הינו k. אם לא קיים זוג איברים כזה יש להחזיר null.	$O(n)$
Sum(k)	החזרת סכום כל המפתחות במבנה שערכם קטן שווה מ k	$O(\log n)$

אין הגבלת זכרון.

תרגיל 6

תארו מבנה נתונים המאפשר ביצוע הפעולות הבאות בסיבוכיות הנדרשת:

- Init(S) – אתחול המבנה, בהינתן סדרת איברים S באורך m, בזמן $O(m)$.
- Insert(x) – הוספת x למבנה, בזמן $O(\log n)$ (n הוא מספר האיברים הנוכחי).
- Find-min – החזרת המינימום, בזמן $O(1)$.
- Find-max – החזרת המקסימום, בזמן $O(1)$.
- Del-min – הוצאת המינימום מהמבנה, בזמן $O(\log n)$.
- Del-max – הוצאת המקסימום מהמבנה, בזמן $O(\log n)$.

<u>הערה :</u> מס' האיברים המקסימלי במבנה נתונים הוא m.

תרגיל 7

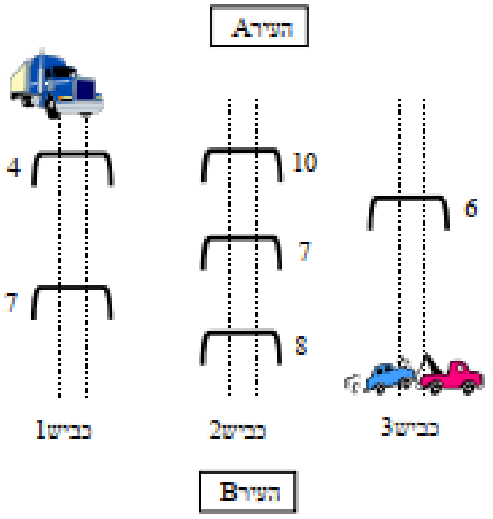
הציעו מימוש למבנה נתונים התומך בפעולות הבאות (n הוא מספר האיברים ברגע נתון):

- $\text{Insert}(x)$ – הוספת האיבר x למבנה, בזמן $O(\log n)$
- $\text{Delete}(x)$ – מחיקת האיבר x מהמבנה, בזמן $O(\log n)$
- $\text{Find}(k)$ – מציאת איבר בעל מפתח k במבנה, בזמן $O(\log n)$
- $\text{Min}()$ – מציאת האיבר בעל מפתח מינימלי במבנה, בזמן $O(1)$
- $\text{Between}(k_1, k_2)$ – החזרת מספר המפתחות בין k_1 ל- k_2 (כולל), בזמן $O(\log n)$

תרגיל 8

נתונות שתי ערים A,B וביניהן יש m כבישים מקבילים (לא נחתכים), הממוספרים מ- 1 ועד m. במהלך הזמן בונים על הכבישים גשרים, כאשר על כל כביש יכול להיות מספר כלשהו (לא מוגבל) של גשרים, והגובה של כל גשר יכול להיות מספר ממשי חיובי כלשהו.

דוגמה: במקרה זה יש m = 3 כבישים בין A ל- B, כאשר על כל כביש מצוירים הגשרים שנבנו עליו, וליד כל גשר רשום גובה הגשר. מספר הגשרים הכולל במקרה זה הוא n = 6.



עליכם לתכנן מבנה נתונים שתומך בפעולות הבאות:

- Init() – אתחול מבנה הנתונים כך שיכיל m כבישים ללא גשרים עליהם.
  יעילות הפעולה צריכה להיות O(m).
- AddBridge(float h, int r) – הוספת גשר שגובהו h על כביש מספר r.
  היעילות צריכה להיות O(log m) במקרה הגרוע.
- WhichRoad(float heigt) – הפונקציה תחזיר מספר של כביש שבו יכולה לנסוע משאית שגובהה height (המשאית תיסע מתחת לגשרים ואסור לה כמובן להיתקל באף אחד מהגשרים שעל הכביש הזה. לכן גובה המשאית צריך להיות קטן ממש מגובה כל הגשרים שעל הכביש). אם יש כמה כבישים שבהם המשאית יכולה לעבור, יוחזר מספרו של אחד מהם. אם המשאית אינה יכולה לעבור באף כביש תחזיר הפונקציה 0.
  יעילות הפעולה צריכה להיות Θ(1) במקרה הגרוע.
- Print(int r) – הפונקציה מדפיסה את הגבהים של כל הגשרים שנמצאים על כביש מספר r.
  יעילות הפונקציה צריכה להיות ליניארית במספר הגשרים שעל הכביש.

תארו את מבנה הנתונים וכתבו אלגוריתם לכל אחת מהפעולות. הסבירו מדוע יעילות כל אחת מהפעולות היא כנדרש.

תשובות

תרגיל 1

נחזיק שני עצי חיפוש (AVL או $2-3$).
עץ אחד יהיה ממוין לפי הקואורדינטה השמאלית והשני לפי הימנית.

- הכנסה = הכנסה לשני העצים
- הסרה = הסרה משני העצים
- חיפוש - חיפוש בעץ המתאים לפי הקואורדינטה המבוקשת
- מעבר על האיברים ממויינים לפי קואו' שמאלית\ימנית - מעבר סדרתי על העץ המתאים.

תרגיל 2

א. תארו את מבנה הנתונים:

עץ avl אשר כל קודקוד v בו מחזיק את השדות הנוספים הבאים:

Max – הערך המקסימלי בתת העץ המושרש מ v

Min – הערך המינימלי בתת העץ המושרש מ v

MinGap - ההפרש המינימאלי בין שני מספרים כלשהם בתת העץ המושרש מ v

ב. תארו את הפעולות והסבירו את זמן הריצה:

Insert(x) – הכנסה לעץ avl ב O(log n) ועדכון השדות כלפי מעלה ב O(log n):

אם בוצעה פעולת heapify – החל מאחיו של הקודקוד x

אחרת – החל מאביו של x

v.Max = max(v, v.right.max)

v.Min = min(v, v.left.min)

v.MinGap = min(v-v.left.max, v-v.right.min, v.left.MinGap, v.left.MinGap)

סה"כ: O(log n)

Search(x) – חיפוש רגיל בעץ avl: O(log n)

Delete(x) – מחיקה מעץ avl ב O(log n) ועדכון השדות החל מאביו של הקודקוד כלפי מעלה

ב O(log n) כפי שמתואר בפונק' Insert(x)

סה"כ: O(log n)

Max_Gap() – החזר root. MinGap

Max_Gap() – החזר root.Max – root.Min

תשפ"ב

תרגול יצירת מבנה נתונים

התקבלו פתרונות נוספים אפשריים. למשל: שימוש בערימה/עץ AVL של הפרשי הערכים עם מצביעים הדדים לעץ שמופיע בפתרון לעיל עבור לחישוב ה- Min_Gap(). בפתרון זה אין צורך בכל השדות בעץ המתוארים לעיל. כמו-כן, ישנן כמה וריאציות אפשריות שהסתמכו על פונקציות שנלמדו בכיתה כמו minimum(),maximum(),successor(),predecessor().

סעיפי ניקוד עיקריים:

- פתרון שלא ענה על Max_Gap() נכון (ובכלל זה בזמן הריצה המבוקש, בין אם השגיאות נובעות מהמבנה או מהאלגוריתם של הפונקציות השונות) – הורדה של עד 13 נקודות.
- פתרון שלא ענה על Max_Gap() נכון (ובכלל זה בזמן הריצה המבוקש, בין אם השגיאות נובעות מהמבנה או מהאלגוריתם של הפונקציות השונות) – הורדה של עד 13 נקודות.
- על פתרון לא עקבי, ברור, לא מפורט ירדו נקודות.
- טעויות עקרוניות שסותרות את הנלמד בקורס הורידו נקודות.
- חישובי זמן ריצה או קביעות ספציפיות שאינן נכונות הורידו נקודות.

תרגיל 3

תארו את מבנה הנתונים:

מבנה הנתונים יהיה מורכב משתי ערימות: ערימת מקסימום עבור כל הערכים שקטנים מהחציון וערימת מינימום עבור כל הערכים שגדולים מהחציון. בנוסף, נחזיר שדה med לחציון.

בנוסף, נחזיק רשימה מקושרת (שנוציא ונכניס אליה ערכים בהתאם לסדר הכנסה והוצאה במחסנית).

יהיו מצביעים הדדיים בין הערכים הזהים ברשימה ובערימה המתאימה.

תארו את הפעולות השונות והסבירו את זמן הריצה:

Build(S,x) – נחלק את קבוצת המספרים הנתונה למספרים שקטנים מהחציון ולמספרים שגדולים מהחציון. מהמספרים שקטנים מהחציון נבנה ערימה מקסימום H_{max}, מהמספרים שגדולים מהחציון נבנה ערימת מינימום H_{min}. נשמור את החציון ב – med.

בנוסף, נוסיף את המספרים לרשימה.

זמן ריצה:

סריקת המספרים + בניית שתי ערימות: $O(n) + 2 \cdot O\left(\frac{n}{2}\right) = O(n)$.

הכנסת המספרים לרשימה: $O(n)$.

בסה"כ: $O(n)$.

תשפ"ב
תרגול יצירת מבנה נתונים

Insert(S,k) – נשווה את k מול החציון, אם k קטן, נכניס את k ל - H_{max}, אחרת ל - H_{min}. נשווה את הגדלים של שתי הערימות. אם הפרש בגדלי הערימות גדול מ – 1, נניח בלי הגבלת הכלליות, כי H_{max} מכילה כרגע בשני ערכים יותר מ - H_{min}, אזי נוציא את המקסימום מ - H_{max}, נשים אותו בתור החציון, ונכניס את החציון הישן ל - H_{min}, במקרה ההפוך, נעשה את הפעולות הסימטריות.

בנוסף, נכניס את הערך החדש לתחילת הרשימה.

זמן ריצה:

מספר קבוע של פעולות הכנסה לערימה ומחיקת שורש הערימה + הכנסה לרשימה:
$O(1)+O(\log n)=O(\log n)$.

Del-Median(S) – נמחק את החציון, בעזרת המצביע ההדדי, נמחק אותו גם מהרשימה.

נבחר כחציון החדש, את האיבר שנמצא בראש הערימה עם יותר ערכים. במקרה ושתי הערימות בגודל זהה, ניקח את שורש הערימה H_{max} (בחירה של הערימה שרירותית).

זמן ריצה:

מחיקה מרשימה + מחיקת שורש ערימה: $O(1)+O(\log n)=O(\log n)$.

Del-New(S) – נמחק את האיבר בתחילת הרשימה, בעזרת המצביע ההדדי, נמחק אותו גם מהערימה המתאימה, נעשה "פעולות איזון" לערימות בהתאם למתואר בפעולת ההכנסה.

זמן ריצה:

מחיקה מרשימה + מחיקה מערימה מאינדקס נתון + מספר קבוע של פעולות הכנסה לערימה ומחיקת שורש הערימה : $O(1)+O(\log n)+O(\log n)=O(\log n)$.

<u>טעויות נפוצות:</u>

הפתרונות הלא נכונים הבאים קיבל בין 6-12 נקודות:

פתרונות שהציעו להשתמש בעצי AVL, לעדכן לכל איבר עוקב וקודם או לתחזק מערך ממוין, ההצעות הללו יכולות לדרוש זמן ריצה של $\Omega(n\log n)$ באתחול.

פתרונות בעזרת ערימות שכוללות את כל n האיברים, ללא התייחסות (או טעויות) כיצד החציון יתוחזק ויעודכן עם כל פעולה.

עוד טעויות:

פתרונות כמעט נכונים, למשל שהציעו שתי ערימות שכ"א מהן מחזיקה מחצית מהערכים, אך עם טעויות בעדכון החציון או ללא התייחסות אליו בכלל, איבדו בין 2-5 נקודות.

בניית ערימה ע"י פעולות הכנסה, בניית ערימה ע"י כך שאנחנו מכניסים את כל אחד מהערכים לערימה, וכך מקבלים ערימה חוקית. בניה שכזו יכולה לקחת $\Omega(n\log n)$, ולא $O(n)$ כמו הבניה שלמדנו. לא הורדו נקודות.

פתרונות אשר הציעו להחזיק מצביע לאחרון שנכנס (ולא מבנה נתונים נוסף כמו רשימה או מחסנית) איבדו בין 3-4 נקודות.
הבעייתיות בלהחזיק רק מצביע היא שלא ברור מה קורה בשתי קריאות רצופות ל – del_new, לא ברור מי יהיה האיבר השני שנצטרך למחוק.

תרגיל 4

א. תארו את מבנה הנתונים:

נשתמש בשני עצי AVL. את הראשון נסמן ב- A ובו נאחסן את כל האיברים, ואת השני נסמן ב- B ובו נאחסן את כל הזוגות במבנה שסכומם 1. (המפתח ב- B הוא לפי הערך הגדול מבין הזוג)

ב. תארו את הפעולות והסבירו את זמן הריצה:

הכנסה: נכניס ל- A כרגיל (הכנסה רגילה לAVL). אם x כבר קיים ב- A האלגוריתם יחזיר כישלון. אחרת נחפש ב- A את המספר $y = 1 - x$. אם איבר זה קיים, אז נכניס לעץ B את הזוג x ו-y. מפתח ההכנסה לפי הערך הגדול מבין השניים.
הוצאה: נוציא מ- A כרגיל(הוצאה רגילה מ-AVL). אם לא קיים נחזיר כשלון. אחרת, נבדוק האם קיים הזוג $(x,1-x)$ ב-B. (לפי הגדול מבין השניים). אם נמצא איבר כזה, נוציא אותו מ-B.
חיפוש: חיפוש רגיל בעץ A.
החזרת זוג: נחזיר את שורש העץ B. (ללא הוצאה) זמן ריצה $O(1)$
מציאת משלים: נחפש בעץ B לפי הגדול מבין x ו- $1-x$. ב- B יש k איברים ולכן זמן הריצה $O(\log k)$.

תרגיל 5

מבנה הנתונים שנחזיק יהיה עץ AVL, שבו כל קודקוד v,
יחזיק שדה נוסף sum, שיכיל את סכום המפתחות הנמצאים בתת-העץ ששורשו הוא v.

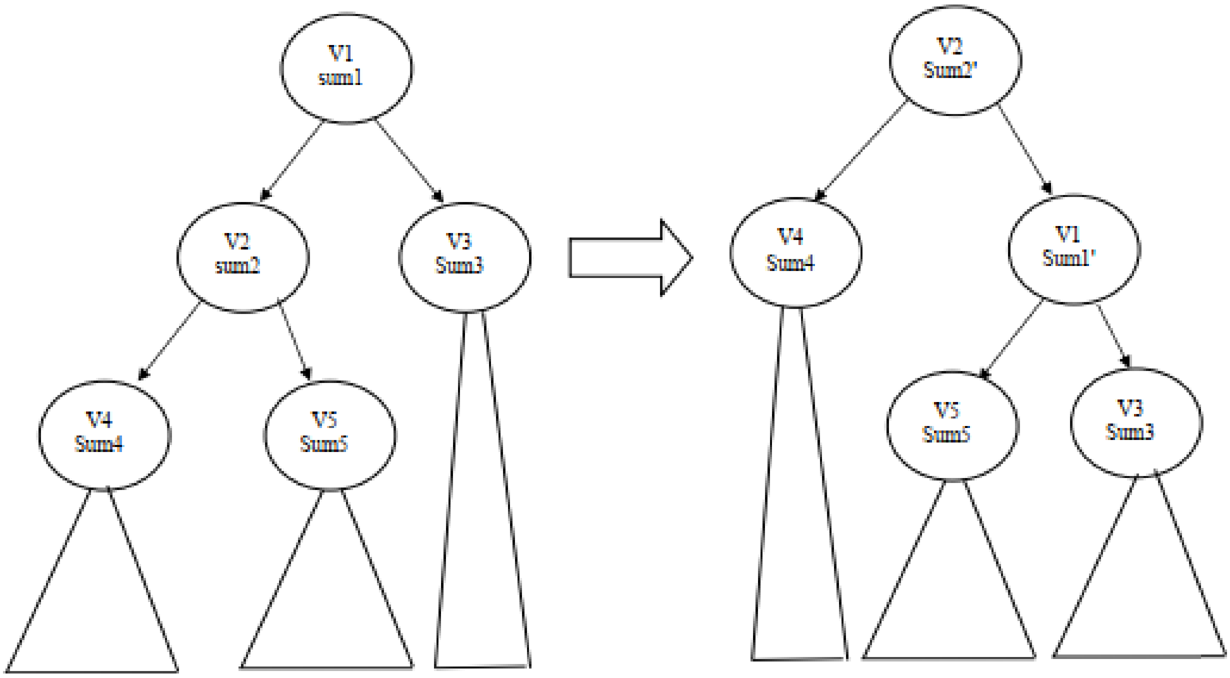
תפקידנו בפעולות ההכנסה ומחיקה לשמר תכונה זו של העץ (אינווריאנטה).

<u>הכנסה:</u>

תזכורת, הכנסת קודקוד חדש לעץ AVL מתבצעת בשני שלבים.

1. הכנסה כעץ בינארי רגיל, הקודקוד החדש ייכנס כעלה בעץ.
2. ביצוע רוטציות במקרה הצורך.

כל פעם שנכניס איבר חדש לעץ, בשלב 1 נוסיף לשדה sum של כל האבות (ancestors) במסלול את ערך המפתח של v. בשלב 2 (רוטציות) נעדכן את ערכי ה sum בהתאם: לדוגמא: עבור רוטציה ימנית יתבצע העדכון הבא:



מחיקה:

כאשר נסיר איבר מן העץ, תחילה נעבור על המסלול מהקודקוד ועד השורש, ונחסר את ערכו משדה ה-sum של האיברים. לאחר מכן נבצע מחיקה של קודקוד מעץ AVL, ונעדכן את ערכי ה-sum של הקודקודים הרלוונטיים.

פעולת Pair Sum:

נבצע סריקת in-order של העץ לתוך מערך A, וקיבלנו מערך ממויין. נריץ את הקוד הבא

```
For (i=0, j=A.length-1;)
{
If (A[i]+A[j] ==k)
        Print("found it, leaving now")
        Exit;
else  If (A[i]+A[j] < k)
        i++
else if (A[i]+A[j]>k)
        j—
}
Print("not found");
```

פעולת Sum:

רעיון הפתרון, נסרוק את העץ באותו אופן שאנו סורקים את העץ כשאנו רוצים להכניס קודקוד חדש שערכו k. בכל פעם נשווה את השורש הנוכחי עם k, אם השורש קטן או שווה ל –k, נוסיף לפתרון את ערך השדה sum של תת העץ השמאלי והשורש. אם השורש גדול מ –k, נמשיך בסריקה מבלי לעדכן דבר.

```
Sum(T, k)
  cnt = 0
  while (T != Null )
    if (k > T.key)
      cnt += T.left.sum+T.key
      T = T.right
    else if (k <  T.key)
          T = T.left
        else    // k = T.key
          cnt +=  T.left.sum + T.key
          break;
  return keys
```

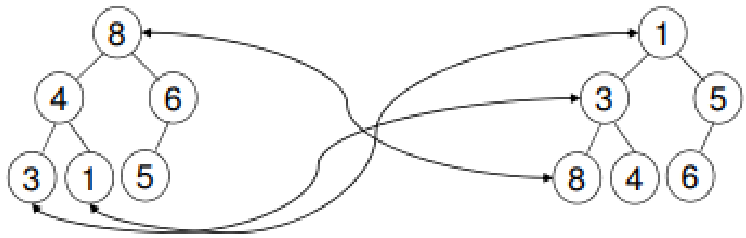
תשפ"ב
תרגול יצירת מבנה נתונים

תרגיל 6

נשמור את האיברים בשתי ערמות בו זמנית: ערימת מינימום וערימת מקסימום.

לכל איבר נשמור גם את האינדקס של ה"תאום" שלו בערמה השנייה.

ערימת מקסימום ערימת מינימום



	1	2	3	4	5	6	7	8
element	8	4	6	3	1	5		
index of "twin"	4	5	6	2	1	3		

1	2	3	4	5	6	7	8
1	3	5	8	4	6		
5	4	6	1	2	3		

<u>מימוש הפעולות</u>

Init(S): נעתיק את איברי S לשני מערכים נפרדים. בשלב זה "תאומים" נמצאים באותו אינדקס, לכן לכל איבר נאתחל את אינדקס ה"תאום" שלו כאינדקס שלו עצמו.

נריץ Build-heap פעם לערמת מינימום ופעם לערמת מקסימום. כל תזוזה של איבר תגרור עדכון של האינדקס ששמור אצל אחיו "התאום" (תוספת של קבוע לסיבוכיות).

זמן: $2\cdot\Theta(m)$ עבור ההעתקה ועוד $2\cdot\Theta(m)$ עבור בניית הערמות. סה"כ $\Theta(m)$.

Find-max / Find-min: החזרת שורש הערמה המתאימה. זמן: $\Theta(1)$.

Insert(x): נכניס את x לכל אחת משתי הערמות. שוב, בכל תזוזה של איבר בערמה אחת, נעדכן את האינדקס ששמור אצל אחיו "התאום" בערמה השנייה בהתאם.

זמן: כל הכנסה $\Theta(\log n)$, סה"כ $\Theta(\log n)$.

Del-min: נמחק את שורש ערימת המינימום ובעזרת האינדקסים נגיע ל"תאום" ונמחק גם אותו מערימת המקסימום. מחיקת שורש תתבצע כרגיל, ואילו מחיקת ה"תאום" תתבצע כפי שראינו בתרגיל קודם. מימוש Del-max סימטרי ל- Del-min.

זמן: כל מחיקה $\Theta(\log n)$, סה"כ $\Theta(\log n)$.

<u>הערה</u>: הצומת התאום של שורש ערימה אחת הוא עלה בערימה השנייה.

תרגיל 7

נשתמש בעץ דרגות (עץ AVL עם שדה נוסף *size*).

בנוסף, נשמור מצביע לאיבר המינימלי בעץ (שיעודכן בעת הכנסה והוצאה, כפי שיוסבר).

- $\text{Insert}(x)$ – נכניס לעץ דרגות כפי שראינו בשקפים – $O(\log n)$, ואם מפתח האיבר שהוכנס קטן מהמינימום נעדכן את המצביע למינימום – $O(1)$.
- $\text{Delete}(x)$ – מציא מעץ דרגות כפי שראינו בשקפים – $O(\log n)$, ואם יש צורך נמצא את המינימום החדש ע"י קריאה ל- AVL-Minimum – $O(\log n)$.
- $\text{Find}(k)$ – כרגיל בעץ AVL – $O(\log n)$.
- $\text{Min}()$ – נחזיר את האיבר המינימלי בעזרת המצביע, ב- $O(1)$.
- $\text{Between}(k_1, k_2)$ – נשתמש בפעולה Tree-Rank שמחזירה את דירוגו של איבר נתון.
  1. איננו יודעים אם קיימים במבנה איברים בעלי המפתחות הנתונים, לכן תחילה נבדוק זאת, ואם לא – נכניס אותם (נקרא להם x_1 ו- x_2).
  2. נחשב את $\text{Tree-Rank}(x_2) - \text{Tree-Rank}(x_1) + 1$.
  3. נפחית מהערך שקיבלנו את כמות האיברים (בין 0 ל- 2) שהכנסנו, וזוהי התוצאה המבוקשת.
  4. לבסוף, אם יש צורך, נמחק את האיברים שהכנסנו.

כל הפעולות הנ"ל רצות בזמן $O(\log n)$.

תרגיל 8

מבנה הנתונים: אנו נשמור לכל כביש את כל הגבהים של הגשרים שנבנו עליו, וכן את גובהו של הגשר המינימלי עליו. בנוסף, נחזיק ערימת מקסימום שבה נשמור לכל כביש את גובהו של הגשר המינימלי עליו. הדבר יאפשר לנו בקלות למצוא מיהו הכביש שגובה הגשר המינימלי עליו גבוה ככל האפשר. ביתר דיוק מבנה הנתונים יכיל את המרכיבים הבאים:

1. ערימת מקסימום שתישמר במערך Heap[m], כאשר בכל איבר בערימה יישמרו השדות הבאים:
   א. mum – מספר כביש.
   ב. min – גובה גשר מינימלי על כביש מספר mum.

   שדה המפתח בערימה יהיה השדה min ואילו השדה mum יהיה שדה נתונים.
2. מערך Roads[m], כאשר בתא i במערך יישמרו השדות הבאים:
   א. index - מספר התא במערך Heap שבו נשמר כביש מספר i.
   ב. List – רשימה מקושרת חד-כיוונית של הגשרים על כביש i.

הנחה: נניח שהתאים של המערכים ממוספרים מ- 1 עד m ולא מ- 0 עד m–1 כמקובל ב- C.

:Init() אתחול

אלגוריתם: עבור $i = 1,\ldots,m$:

.$Heap[i].num = i$

.$Heap[i].min = \infty$

.$Roads[i].List = NULL$

.$Roads[i].index = i$

יעילות האלגוריתם: היא כמובן $\Theta(m)$ כנדרש.

הערה: אתחלנו את השדה min להכיל מספר גדול מאוד ∞, שפירושו שכל משאית יכולה לעבור על הכביש כי אין עליו גשרים. לחילופין, אפשר לאתחל את השדה min להכיל ערך 0 שאינו ערך חוקי של גובה של גשר, אבל אז יש לדאוג לעדכן את הפעולות הרגילות המוגדרות על ערימת מקסימום כך שיתייחסו אל הערך 0 כאל ערך הגדול מכל ערך אחר.

:AddBridge(float hb, int r)

הרעיון: נוסיף גשר חדש שגובהו hb לכביש מספר r בערימה ונתקן כמובן את הערימה.

אלגוריתם:

1. נוסיף את hb כאיבר חדש לראש הרשימה $Roads[r].List$.
2. יהי $j = Roads[r].index$ מספר התא שבו נמצא כביש מספר r בערימת המקסימום.
3. אם $hb < Heap[j].min$ אז:
   א. נעדכן $Heap[j].min = hb$.
   ב. נתקן את הערימה על ידי קריאה ל- $FixHeap(j)$.

הערה: הפונקציה FixHeap(int j) תתקן את תת-הערימה שהשורש שלה נמצא בתא מספר j במערך Heap בדיוק כפי שלמדנו בכיתה, אולם כל פעם שיש החלפה בין שני תאים בערימה, היא תיגש גם למערך Roads ותחליף בין הערכים המתאימים. כלומר, אם מחליפים בערימה את התאים שמכילים את כבישים מספר s ו- t, אז נחליף בין $Roads[s].index$ ל- $Roads[t].index$.

יעילות: הכנסת איבר לראש רשימה לוקחת $\Theta(1)$ פעולות. לכן, יעילות האלגוריתם נשלטת על ידי היעילות של FixHeap. מכיוון שהערימה שלנו מכילה m איברים, הרי גובהה $\Theta(\log m)$, ולכן היעילות תהיה $O(\log m)$.

:WhichRoad(float hr)

אלגוריתם:

1. יהי $max = Heap[1].min$.
2. אם $hr < max$ אז נחזיר את $Heap[1].num$.
3. אחרת נחזיר 0.

יעילות: הפעולה היא כמובן $\Theta(1)$.

:Print(int r)

אלגוריתם: נדפיס את כל איברי הרשימה $Roads[r].List$.

יעילות: הפעולה היא כמובן ליניארית במספר הכבישים על כביש מספר r.

הערה:

a) שימו לב שאחרי בנית הערמה בשלב ה- Init, הפעולה היחידה המעדכנת אותה היא הקטנת ערך של איבר הנמצא בה (אין שימוש בפעולות העדכון הרגילות של ערמה ו- Insert ו- DeleteMax).

b) נקרא לפעולה המוזכרת בסעיף הקודם DecreaseValue. נסו להראות שאם בנית הערמה לוקחת $O(n)$ זמן, אז אחת הפעולות Max או DecreaseValue חייבות לקחת $\Omega(\log n)$ זמן.