מבני נתונים - עבודה מספר 4 – שי בן-ששון (061203410) ויטב אליהו (042828855)

חלק א

מבנה נתונים שיכול להתאים ל ADT הנתון הוא עץ AVL בתוספת פונקציית range שנציג את מימושה בהמשך.

1. **Insert(x)** :

תחילת ההכנסה כמו בעץ חיפוש בינארי רגיל,
עוברים על העץ מהשורש ,
אם המפתח x.key שווה למפתח של הצומת עליה עוברים לא עושים כלום (האיבר כבר קיים בעץ).
אם המפתח x.key קטן מהמפתח של הצומת עליה עוברים נפנה לבן השמאלי.
אם המפתח x.key גדול מהמפתח של הצומת עליה עוברים נפנה לבן הימני.
נמשיך כך עד שנגיע לעלה ושם נוסיף את x בתור בן ימני או שמאלי כך ששומרים על תכונת עץ חיפוש בינארי.
כעת מעדכנים את שדה הגובה של העלה אליו הוספנו את x ובודקים האם הוא מאוזן לפי תכונת עץ AVL.
אם כן ממשיכים לעלות להורה של הצומת הנוכחי מעדכנים את הגובה ובודקים האם הוא מאוזן או לא.
אם הגענו לצומת לא מאוזן מבצעים תיקון :
נסמן z - הצומת הלא מאוזן
נסמן y – הבן של z עם הגובה היותר גדול.
נסמן x – הבן של y עם הגובה היותר גדול.
בהתאם למיקום היחסי בין x,y,z מבצעים את התיקון המתאים.
ישנם 4 מקרים :
y בן שמאלי של z , x בן שמאלי של y - מבצעים רוטציה ימנית על z כפי שנלמד בכיתה.
y בן ימני של z , x בן ימני של y - מבצעים רוטציה ימנית על z כפי שנלמד בכיתה.
y בן שמאלי של z , x בן ימני של y – מבצעים תחילה רוטציה שמאלית על y ואחר כך רוטציה ימנית על z .
y בן ימני של z , x בן שמאלי של y – מבצעים תחילה רוטציה ימנית על y ואחר כך רוטציה שמאלית על z .

<u>סיבוכיות:</u>

הכנסה רגילה של BST $O(logn)$

עדכון במידת הצורך שדה height $h \bullet O(1)$

כאשר נלמד בכיתה שגובה של עץ AVL בעל n צמתים הוא $O(logn)$

לכן $O(\log n) \bullet O(1) = O(\log n)$

עולים מהצומת החדש אל השורש כדי לבדוק האם צומת מסוים לא מקיים את תכונת האיזון

$O(logn)$

רוטציה $O(1)$

סה"כ זמן ריצה של הכנסה הוא $O(logn)$

2. **Find(x)**

חיפוש מתבצע כמו חיפוש בעץ חיפוש בינארי רגיל ,
עוברים על העץ מהשורש ,
אם המפתח x.key שווה למפתח של הצומת עליה עוברים מחזירים את הצומת .
אם המפתח x.key קטן מהמפתח של הצומת עליה עוברים נפנה לבן השמאלי.
אם המפתח x.key גדול מהמפתח של הצומת עליה עוברים נפנה לבן הימני.
אם הגענו לצומת null ולא החזרנו תשובה נחזיר null .

<u>**סיבוכיות:**</u>
במקרה הגרוע ביותר חיפשנו לאורך כל העץ , כלומר בכל הגובה h .

ע״פ מה שנלמד בכיתה גובה של עץ AVL בעל n צמתים הוא $O(logn)$,

לכן זמן הריצה הוא $O(logn)$

3. **Delete(x)**

תחילת המחיקה תתבצע כמו מחיקה של עץ חיפוש בינארי רגיל ,
ישנם 3 מקרים :

א. הצומת שרוצים למחוק היא עלה בעץ - אז שמים מצביע ל null בצומת האבא במקום המצביע לצומת שרוצים למחוק.
ב. לצומת שרוצים למחוק יש בן אחד בדיוק – שמים את המצביע לתת העץ המושרש בצומת במקום המצביע לצומת שרוצים למחוק בצומת האבא (ומעדכנים את האבא של תת העץ להיות האבא של הצומת שרוצים למחוק).
ג. לצומת שרוצים למחוק יש שני בנים – מעתיקים את ה key והsatellite data של העוקב של x לתוך הצומת x .
מפני שלעוקב יש בן אחד לכל היותר(אחרת הוא לא היה העוקב) ,
מבצעים מחיקה לפי א . או ב. בהתאם למקרה.

כעת מעדכנים את תכונת הגובה של צומת האבא של מי שמחקנו לפי א. או ב. , ובודקים האם הוא מאוזן לפי תכונת עץ AVL. אם כן ממשיכים לעלות להורה של הצומת הנוכחי ובודקים האם הוא מאוזן או לא.

אם הגענו לצומת לא מאוזן מבצעים תיקון :
נסמן z - הצומת הלא מאוזן
נסמן y – הבן של z עם הגובה היותר גדול.

נסמן x – הבן של y עם הגובה היותר גדול.
בהתאם למיקום היחסי בין x,y,z מבצעים את התיקון המתאים.
ישנם 4 מקרים :
y בן שמאלי של z , x בן שמאלי של y - מבצעים רוטציה ימנית על z כפי שנלמד בכיתה.
y בן ימני של z , x בן ימני של y - מבצעים רוטציה ימנית על z כפי שנלמד בכיתה.
y בן שמאלי של z , x בן ימני של y – מבצעים תחילה רוטציה שמאלית על y ואחר כך רוטציה ימנית על z .
y בן ימני של z , x בן שמאלי של y – מבצעים תחילה רוטציה ימנית על y ואחר כך רוטציה שמאלית על z .
ממשיכים לעלות ולבצע רוטציות במידת הצורך עד שנגיע לשורש.

<u>סיבוכיות:</u>

מחיקה רגילה של BST $O(logn)$

עדכון במידת הצורך שדה height $h \bullet O(1)$

כאשר נלמד בכיתה שגובה של עץ AVL בעל n צמתים הוא $O(logn)$

לכן $O(\log n) \bullet O(1) = O(\log n)$

עולים מהאבא של הצומת שנמחק אל השורש כדי לבדוק האם צומת מסוים לא מקיים את תכונת האיזון $O(logn)$

ממשיכים לעלות ולבצע רוטציות במידת הצורך עד שנגיע לשורש $O(logn)$

סה״כ זמן ריצה של מחיקה הוא $O(logn)$

4. **Range(a , b)**:

1) נאתחל רשימה מקושרת חדשה.
2) נמצא את הצמתים המכילים את a ו b בעץ או את הצומת האחרון בחיפוש עץ חיפוש בינארי של a ו b אם הראשונים לא נמצאו , יהיו P_a ו P_b מסלולי החיפוש מן השורש לצמתים הללו בהתאמה.

3) נמצא את הצומת x אשר ממנה המסלולים P_a ו P_b מתפצלים.

4) לכל צומת v במסלול החיפוש מ a (או מהצומת האחרון בחיפוש עץ חיפוש בינארי של a) אל x צריך לעשות :
אם (a<v.key)
(i) הוסף את v בסוף הרשימה המקושרת
(ii) בצע סריקת (inorder(v.right והוסף את הצמתים לפי סדר הסריקה לסוף הרשימה

5) לכל צומת v במסלול החיפוש מ x אל b (או אל הצומת האחרון בחיפוש עץ חיפוש בינארי של b)צריך לעשות :
אם (b>v.key)
(i) בצע סריקת (inorder(v.left והוסף את הצמתים לפי סדר הסריקה לסוף הרשימה
(ii) הוסף את v בסוף הרשימה המקושרת

6) אתחל מערך חדש באורך מספר האיברים שברשימה המקושרת.

7) עבור על הרשימה המקושרת מההתחלה ועד לסוף והעתק את האיברים אל המערך מתחילתו ועד לסופו.

8) החזר את המערך.

<u>סיבוכיות:</u>

1) $O(1)$

2) $O(\log n)+O(\log n)$

3) $O\left(logn\right)$

4 + 5) – סריקת inorder על m איברים והכנסה לרשימה מקושרת ב m $O(1)$ פעמים :

$$O(m)+O(m)$$

6) $O(m)$

7) $O(m)$

סה"כ : $O(\log n+m)$

חלק ב

מימוש החיפוש וההכנסה של תשתית AVL.java זהה מבחינת האלגוריתם וזמני הריצה לחיפוש וההכנסה בחלק א.

השוני בין עץ AVL רגיל לבין המימוש הוא בפונקציית range(a,b) שתפורט להלן :

1) נאתחל רשימה מקושרת חדשה.
2) נמצא את הצמתים המכילים את a ו b בעץ או את הצומת האחרון בחיפוש עץ חיפוש בינארי של a ו b אם הראשונים לא נמצאו , יהיו P_a ו P_b מסלולי החיפוש מן השורש לצמתים הללו בהתאמה.

3) נמצא את הצומת x אשר ממנה המסלולים P_a ו P_b מתפצלים.

4) לכל צומת v במסלול החיפוש מ a (או מהצומת האחרון בחיפוש עץ חיפוש בינארי של a) אל x צריך לעשות :
אם (a<v.key)
(i) הוסף את v בסוף הרשימה המקושרת
(ii) בצע סריקת inorder(v.right) והוסף את הצמתים לפי סדר הסריקה לסוף הרשימה

5) לכל צומת v במסלול החיפוש מ x אל b (או אל הצומת האחרון בחיפוש עץ חיפוש בינארי של b)צריך לעשות :
אם (b>v.key)
(i) בצע סריקת inorder(v.left) והוסף את הצמתים לפי סדר הסריקה לסוף הרשימה
(ii) הוסף את v בסוף הרשימה המקושרת

6) החזר את הרשימה המקושרת.

<u>סיבוכיות:</u>

1) $O(1)$

2) $O(\log n)+O(\log n)$

3) $O\left(logn\right)$

4) + 5) – סריקת inorder על m איברים והכנסה לרשימה מקושרת ב $O(1)$ m פעמים:

$$O(m)+O(m)$$

סה"כ : $O(\log n+m)$

חלק ג

בונוס

נניח שפונקציית הגיבוב הכניסה את כל הערכים לאותו מקום בטבלה כלומר תוצאת הפונקצייה היא c .

אזי

$$\forall\, x, y : (x+y) \bmod m = c$$
$$0 \le c \le m-1$$
$$\Leftrightarrow x + y = k \bullet m + c$$
$$k \in \mathbb{Z}$$
$$\Leftrightarrow y = -x + k \bullet m + c$$

ניתן לראות שהתקבלה משוואת ישר מהצורה $y = a \bullet x + b$

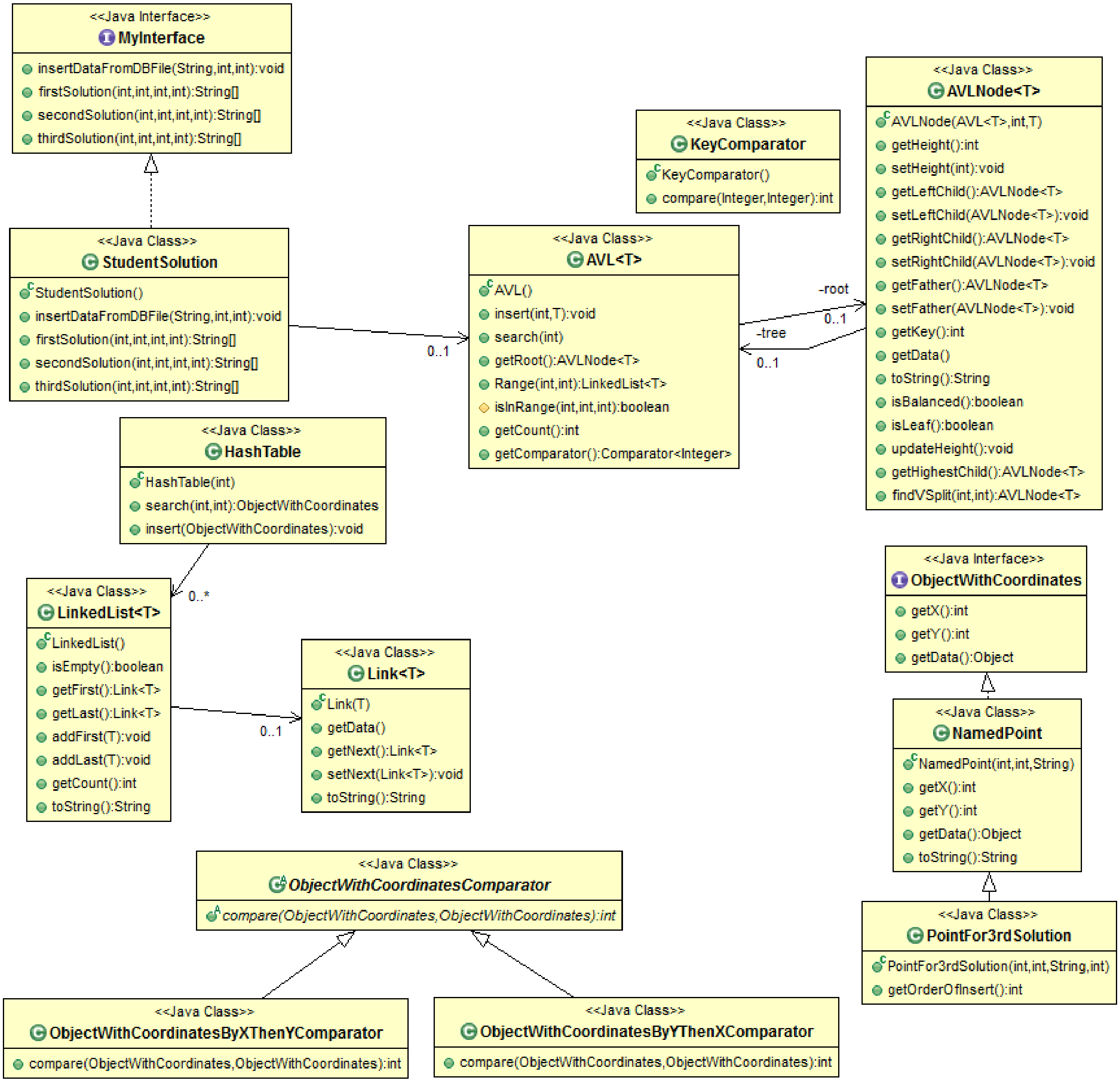
$$a = -1$$

כאשר $b = k \bullet m + c$

כלומר עבור m קבוע כל תתי הקבוצות של הנקודות שמקיימות את משוואת הישרים מהצורה הנ״ל הם פריסה אפשרית של נקודות תחת האילוץ הנ״ל.

חלק ד

להלן סכמה כללית של המחלקות שמימשנו והיררכיה ביניהן :



הערות בנוגע למחלקות:

- KeyComparator הוא Comparator<Integer> המשווה בין שני מפתחות מסוג Integer.
- המחלקות היורשות מObjectWithCoordinatesComparator מאפשר לנו השוואה בין מופעים ObjectWithCoordinates בחלק ד' מימוש 2.
- NamedPoint היא מחלקה הממשת את ObjectWithCoordinates.
- PointFor3rdSolution היא מחלקה המרחיבה את NamedPoint ומוסיפה לה שדה המאחסן את המיקום של האיבר לפי סדר הכנסתו למבנה (נשתמש בה מאוחר יותר במימוש 3 כמזהה ייחודי למופע של ObjectWithCoordinates).
- המחלקה LinkedList הנה מחלקה המממשת רשימה מקושרת שניתן להוסיף לה איברים (אך לא למחוק ממנה). בחרנו לממש אותה מכיוון שרצינו להיות בטוחים לגבי זמני הריצה של השיטות שלה. כחלק מהמימוש שלנו מימשנו שדה count אשר מוסיף 1 כל פעם שנוספת חוליה לרשימה.

מימוש 1:

:Preprocessing

שימוש בעץ הגנרי AVL.java על מנת לייצר אובייקט עץ AVL שה data שלו מסוג NamedPoint.java implments ObjectWithCoordinates .

לצורך השאילתות range אנו יוצרים שני עצי AVL אשר ה מהם ממוין ע"פ קואורדינאטת ה x והשני לפי קואורדינאטת ה y .

מימוש:

1. הפעלת שאילתת range בעץ הראשון על ציר x וקבלת m_1 נקודות.
2. הפעלת שאילתת range בעץ השני על ציר y וקבלת m_2 נקודות.
3. מציאת החיתוך :

א. מעבר על כל m_1 הנקודות כל נקודה פעם אחת בדיוק והכנסתן לטבלת Hash מסעיף ג.

ב. מציאת הקטן מבין m_1 ו m_2 ע"י שימוש בשיטת הgetCount של הרשימה המקושרת LinkedList.

ג. אתחול של מערך בגודל הקטן מבין m_1 ו m_2 (הגודל המקסימלי של החיתוך הוא הקטן מבין שניהם).

ד. מעבר על כל m_2 הנקודות ובדיקה של כל נקודה פעם אחת בדיוק האם היא נמצאת בטבלת ה Hash ,

ה. אם נקודה נמצאת מכניסים אותה למערך.

ו. העברת הנקודות שנמצאו בחיתוך למערך חדש בגודל החיתוך , והחזרת המערך החדש בתור תשובה.

סיבוכיות:

זמן ריצה :

1.) $O(\log n+m_1)$ מחלק א זמן הריצה של שאילתת range הוא $O(\log n+m)$).

2.) $O(\log n+m_2)$ (מחלק א זמן הריצה של שאילתת range הוא $O(\log n+m)$).

3.

א. $m_1 \bullet O(1)=O(m_1)$

ב. מציאת המינימום בO(1) מכיוון שgetCount() הוא שדה שמתעדכן רק בהכנסה.

ג. $O(\min\{m_1, m_2\})$

ד. $m_2 \bullet O(1)=O(m_2)$: Average (נלמד בכיתה שחיפוש בטבלת hash בשיטת גיבוב עם שרשור הוא בממוצע) $O(1)$.

$m_2 \bullet O(m_1)=O(m_2 \bullet m_1)$: Worst Case (נלמד בכיתה שבשיטת גיבוב עם שרשור במקרה הגרוע כל איברי m_1 ייכנסו לאותו תא ולכן זמן ריצה של חיפוש יהיה $O(m_1)$).

ה. $O(\min\{m_1, m_2\})$

ו. $O(\min\{m_1, m_2\})$

סה"כ :

במקרה הממוצע : $O(m_1+m_2+\log n)$

במקרה הגרוע : $O(m_1 \bullet m_2+\log n)$

<u>זיכרון :</u>

1. $O(m_1)$

2. $O(m_2)$

3.

א. $O(m_1)$

ב. לא דורש זיכרון נוסף ממה שכבר הוקצה

ב. $O(\min\{\mathrm{m}_1,\mathrm{m}_2\})$

ג. לא דורש זיכרון נוסף ממה שכבר הוקצה

ד. לא דורש זיכרון נוסף ממה שכבר הוקצה

ה. $O(\min\{\mathrm{m}_1,\mathrm{m}_2\})$

סה״כ :

$$O(m_1+m_2)$$

מימוש 2:

:Preprocessing

כמו במימוש הראשון.

מימוש :

1. הפעלת שאילתת range בעץ הראשון על ציר x וקבלת m_1 נקודות.

2. הפעלת שאילתת range בעץ השני על ציר y וקבלת m_2 נקודות.

3. מציאת החיתוך :

א. מציאת $\max\{m_1, m_2\}$ ע״י שימוש בשיטת הgetCount של הרשימה המקושרת LinkedList.

ב. מציאת הקטן מבין m_1 ו m_2 ע״י שימוש בשיטת הgetCount של הרשימה המקושרת LinkedList.

ג. אתחול של מערך merged[] בגודל הקטן מבין m_1 ו m_2 (הגודל המקסימלי של החיתוך הוא הקטן מבין שניהם).

ד. העתקת הנקודות מהרשימה המקושרת של $\max\{m_1, m_2\}$ לפי הסדר באופן שעוברים על כל נקודה פעם אחת בדיוק (כך שיישארו ממוינות) למערך חדש maxArr[].

ה. לכל אחד מהנקודות $\min\{m_1, m_2\}$ נבצע חיפוש בינארי פעם אחת בדיוק במערך maxArr[] ,

ו. אם נקודה נמצאת מכניסים אותה למערך merged[] .

ז. העברת הנקודות שנמצאו בחיתוך למערך חדש בגודל החיתוך ,

והחזרת המערך החדש בתור תשובה.

<u>סיבוכיות:</u>

<u>זמן ריצה :</u>

1.($O(\log n+m_1)$ מחלק א זמן הריצה של שאילתת range הוא $O(\log n+m)$).

2. ($O(\log n+m_2)$ מחלק א זמן הריצה של שאילתת range הוא $O(\log n+m)$).

3.

א. מציאת המקסימום בO(1) מכיוון שgetCount() הוא שדה שמתעדכן רק בהכנסה.

ב. מציאת המינימום בO(1) מכיוון שgetCount() הוא שדה שמתעדכן רק בהכנסה.

ג. $O(\min\{m_1,m_2\})$

ד. $O(\max\{m_1,m_2\})$

ה.

$$O(\min\{m_1,m_2\})\bullet O(\log(\max\{m_1,m_2\}))=O(\min\{m_1,m_2\}\bullet\log(\max\{m_1,m_2\}))$$

(חיפוש בינארי הוא $O(\log n)$ הוכחה מצורפת בהמשך)

ו. $O(\min\{m_1,m_2\})$

ז. $O(\min\{m_1,m_2\})$

סה"כ : $O(\min\{m_1,m_2\}\bullet\log(\max\{m_1,m_2\})+\max\{m_1,m_2\}+\log n)$

<u>הוכחת זמן ריצה חיפוש בינארי :</u>

נתייחס למקרה הגרוע ביותר בו האיבר לא נמצא במערך .

נסמן ב $T(n)$ את זמן הריצה של הפונקציה על Arr[a…b] שמכיל n נקודות.

קיים $n=b-a+1$

כדי להקל על החשבון נניח בשלב ראשון $n=2^k-1$

משתי המשוואות

$$
\begin{aligned}
&2^k-1=n=b-a+1\\
&b=2^k+a-2\\
&middle=\left\lfloor \frac{a+b}{2} \right\rfloor\\
&=\left\lfloor \frac{a+2^k+a-2}{2} \right\rfloor\\
&=a+2^{k-1}-1
\end{aligned}
$$

נחשב את גודלי 2 תתי המערכים שעליהם מבצעים את הפיצול בחיפוש.

תת מערך 1 Arr[a…middle-1] :

$$middle-1-a+1=middle-a=(a+2^{k-1}-1)-a=2^{k-1}-1$$

תת מערך 2 Arr[middle+1…b] :

$$
\begin{aligned}
&b-(middle+1)+1=b-middle=\\
&=(2^k+a-2)-(a+2^{k-1}-1)=\\
&=2^k-2^{k-1}-1\\
&=2^{k-1}-1
\end{aligned}
$$

אם כן משוואת הנסיגה המתאימה לתאר את הבעיה היא :

$$
\begin{aligned}
&T(2^k-1)=T(2^{k-1}-1)+c\\
&T(0)=1
\end{aligned}
$$

נפתור בשיטת האיטרציה :

$$
\begin{aligned}
&T(2^k-1)=T(2^{k-1}-1)+c=T(2^{k-2}-1)+2c=\\
&\vdots\\
&=T(2^{k-i}-1)+ic\\
&\begin{bmatrix}
2^{k-i}-1=0\\
2^{k-i}=1\\
k-i=0\\
i=k\\
\downarrow\left[n=2^{k-1}\Rightarrow \log n=k-1\Rightarrow \log n+1=k\right]\\
\downarrow\\
i=\log n+1
\end{bmatrix}\\
&=T(0)+(\log n+1)\bullet c\\
&=O(\log n)
\end{aligned}
$$

לכל m טבעי נסמן ב $u(m)$ את השלם הגדול ביותר שצורתו 2^k-1 ,

ושמקיים את אי השוויון : $u(m)=2^k-1<2m$

מהתכונה האקסטרמלית של $u(m)$ נובע :

$$
\begin{aligned}
&2m\le 2^{k+1}-1\\
&\Leftrightarrow m\le 2^k-\frac{1}{2}\\
&\Rightarrow m<2^k
\end{aligned}
$$

זהו אי שוויון חריף בין שני מספרים שלמים לכן :

$$
\begin{aligned}
&m\le 2^k-1\\
&m\le u(m)
\end{aligned}
$$

מסקנה : לכל m טבעי קיים מספר $u(m)$ שצורתו 2^k-1 ושמקיים : $m\le u(m)<2m$

נשתמש במסקנה : $n\le u(n)<2n$

מן המונוטוניות של $T()$ (פונקציית זמן ריצה ולכן לקלט יותר גדול תיתן תוצאה יותר גדולה או שווה) נובע

$$T(n)\le T(u(n))=c\bullet\log(\mathrm{u}(n))$$

מן המונוטוניות של פונקציית log :

$$c \bullet \log(\mathrm{u}(n)) < c \log 2n$$

קיבלנו אם כן :

$$T(n) \le T(u(n)) = c \bullet \log(\mathrm{u}(n)) < c \log 2n$$
$$T(n) < c \log 2n$$

כלומר

$$T(n) = O(\log n)$$

<u>זיכרון :</u>

1. $O(m_1)$
2. $O(m_2)$
3.

א. לא דורש זיכרון נוסף

ב. לא דורש זיכרון נוסף

ג. $O(\min\{\mathrm{m}_1, \mathrm{m}_2\})$

ד. $O(\max\{\mathrm{m}_1, \mathrm{m}_2\})$

ה. לא דורש זיכרון נוסף ממה שכבר הוקצה

ו. לא דורש זיכרון נוסף ממה שכבר הוקצה

ז. $O(\min\{\mathrm{m}_1, \mathrm{m}_2\})$

סה"כ : $O(m_1 + m_2)$

מימוש 3:

:Preprocessing

שימוש בעץ הגנרי AVL.java על מנת לייצר אובייקט עץ AVL שה data שלו מסוג PointFor3rdSolution.java extends NamedPoint.java

השוני בין המחלקה היורשת למחלקת האבא הוא ש PointFor3rdSolution.java מכילה שדה נוסף מסוג int בשם orderOfInsert . השדה מקבל ערך מ 0 עד n-1 לפי סדר ההכנסה של כל נקודה.

לצורך השאילתות range אנו יוצרים שני עצי AVL אשר ה מהם ממוין ע״פ קואורדינאטת ה x והשני לפי קואורדינאטת ה y .

אתחול משתנה mergerTimeStamp מסוג int והשמה של ערך תחילי 0.

:מימוש

1. הפעלת שאילתת range בעץ הראשון על ציר x וקבלת m_1 נקודות.

2. הפעלת שאילתת range בעץ השני על ציר y וקבלת m_2 נקודות.

3. מציאת החיתוך :

א. אם זו הקריאה הראשונה לפונקציה מאתחלים מערך חדש []merger בגודל n ומאפסים אותו.

ב. הגדלת mergerTimeStamp ב 1 .

ג. מעבר על כל אחת מ m_1 הנקודות כל נקודה פעם אחת בדיוק , והשמה של mergerTimeStamp במערך []merger בתא שהאינדקס שלו הוא ה orderOfInsert של כל נקודה.

ד. מציאת הקטן מבין m_1 ו m_2 ע״י שימוש בשיטת הgetCount של הרשימה המקושרת LinkedList.

ה. אתחול של מערך []merged בגודל הקטן מבין m_1 ו m_2 (הגודל המקסימלי של החיתוך הוא הקטן מבין שניהם).

ו. מעבר על כל אחת מ m_2 הנקודות כל נקודה פעם אחת בדיוק , ובדיקה במערך []merger בתא שהאינדקס שלו הוא ה orderOfInsert של כל נקודה , האם הערך של התא שווה ל mergerTimeStamp ,

ז. אם כן (כלומר התא עם האינדקס לפי סדר ההכנסה ״סומן״ בשלב ג.) ,

נכניס את הנקודה למערך []merged .

ח. העברת הנקודות שנמצאו בחיתוך למערך חדש בגודל החיתוך ,

והחזרת המערך החדש בתור תשובה.

סיבוכיות:

זמן ריצה :

1.) $O(\log n+\mathrm{m}_1)$ מחלק א זמן הריצה של שאילתת range הוא $O(\log n+m)$).

2.) $O(\log n+m_2)$ (מחלק א זמן הריצה של שאילתת range הוא $O(\log n+m)$).

3.

א. אם זו קריאה ראשונה $O(n)$ אך לפי הוראות העבודה מותר.

ב. $O(1)$

ג. $O(\mathrm{m}_1)$

ד. מציאת המינימום בO(1) מכיוון שgetCount() הוא שדה שמתעדכן רק בהכנסה.

ה. $O(\min\{\mathrm{m}_1,\mathrm{m}_2\})$

ו. $O(\mathrm{m}_2)$

ז. $O(\min\{\mathrm{m}_1,\mathrm{m}_2\})$

ח. $O(\min\{\mathrm{m}_1,\mathrm{m}_2\})$

סה"כ : $O(m_1+m_2+\log n)$

זיכרון :

1. $O(m_1)$

2. $O(m_2)$

3.

א. $O(n)$

ב. לא דורש זיכרון נוסף ממה שכבר הוקצה

ג. לא דורש זיכרון נוסף ממה שכבר הוקצה

ד. לא דורש זיכרון נוסף ממה שכבר הוקצה

ה. $O(\min\{\mathrm{m}_1, \mathrm{m}_2\})$

ו. לא דורש זיכרון נוסף ממה שכבר הוקצה

ז. לא דורש זיכרון נוסף ממה שכבר הוקצה

ח. $O(\min\{\mathrm{m}_1, \mathrm{m}_2\})$

סה״כ : $O(n)$

חלק ה'

הסבר קצר על הסטים שצירפנו

האתר The Million Dollar Homepage:

מתוך [Wikipedia](https://he.wikipedia.org):

The Million Dollar Homepage (בתרגום חופשי: "דף הבית השווה מיליון דולר") הוא אתר אינטרנט שנוסד בשנת 2005 על ידי הסטודנט אלכס טו בן ה-21 מווילטשייר שבאנגליה, במטרה לממן את הלימודים האקדמיים שלו. דף הבית של האתר כולל מיליון פיקסלים שאורגנו במערך תמונות בגודל 1,000 × 1,000 פיקסלים. כל פיקסל נמכר תמורת דולר אמריקאי אחד בחבילה של 10×10 פיקסלים. הרוכשים סיפקו קובץ תמונה בגודל השטח שנרכש, היפר קישור (URL) וסיסמת פרסום שתוצג כאשר סמן העכבר מרחף מעל לתמונה. מטרתו של האתר הייתה למכור את כל מיליון הפיקסלים וליצור הכנסה של מיליון דולרים.

האתר הושק ב-26 באוגוסט 2005 והפך במהרה לתופעת אינטרנט מוכרת (*היה באז מטורף באותה תקופה למי שזוכר*). ב-1 בינואר 2006 הוצעו 1,000 הפיקסלים האחרונים למכירה פומבית, וזו נסגרה ב-11 בינואר עם הצעת המחיר הגבוהה ביותר - $38,100. ההכנסה ממכירת כל הפיקסלים הייתה $1,037,100.

אנחנו פילטרנו את 90~ הבאנרים הגדולים ביותר (לאחר שהסרנו קורדינטות x ו/או y זהות), מתוכם חילצנו את הנקודה האמצעית ואז המרנו אותה לפורמט שתוכנית הבדיקה GUI מקבלת. כמו כן החבאנו מספר פרסומות מגניבות, מקווים שתמצאו ☺ (לדוגמא: הלב).