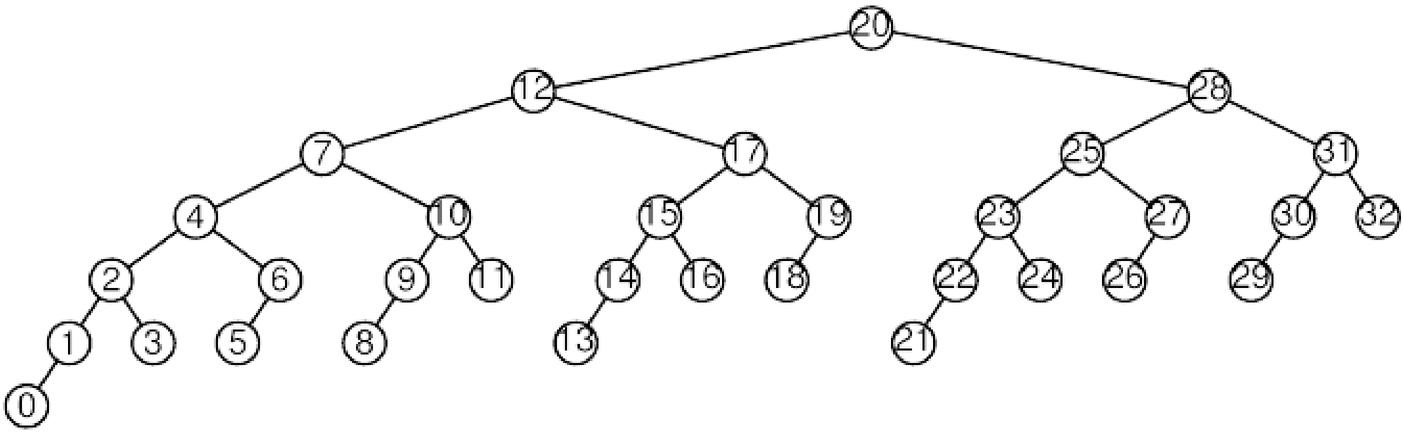
<u>עבודה במבני נתונים</u>

ניב לוי- 204416846 ודין אברג'ל- 308365881

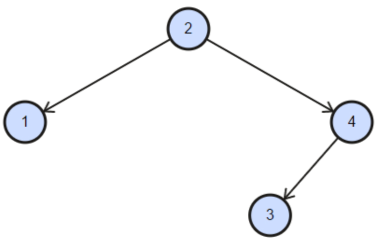
<u>שאלה 1:</u>

סעיף א': הטענה אינה נכונה, דוגמה נגדית:

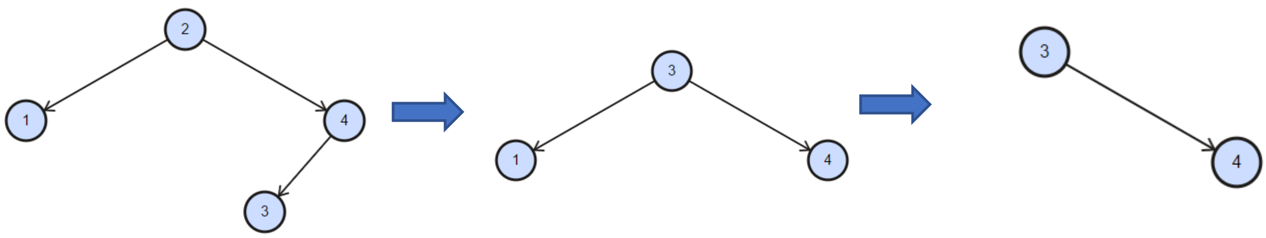


סעיף ב': הטענה אינה נכונה, דוגמה נגדית:

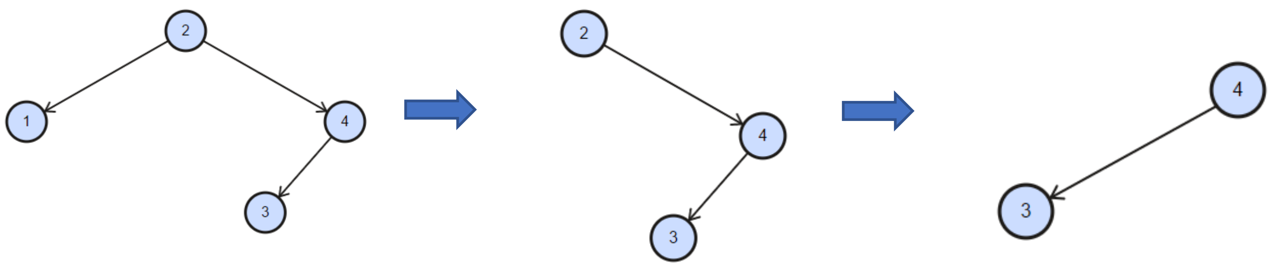
נתבונן בעץ הבא:



נרצה למחוק את קודקוד 2 ואחריו את קודקוד 1:



כעת נמחוק בסדר הפוך, קודם את קודקוד 1 ואחריו את קודקוד 2:



קיבלנו שני עצים שונים שבאחד מהם 3 הוא השורש ובשני 4 הוא השורש.

סעיף ג': הטענה נכונה

נסמן Wh – מס' הצמתים המינימלי של עץ AVL בגובה h.

$W0 = 1$ – בעץ קיים רק השורש.

$W1 = 2$ – עץ שבנוי משורש ובן.

ובאופן כללי: מס' הצמתים המינימלי בעץ בגובה h שווה ל מס' הצמתים המינימלי של עץ בגובה h-1 + מס' הצמתים המינימלי של עץ בגובה h-2 + השורש.

כלומר: $W(h) = W(h-1) + W(h-2) + 1$

נחשב את מספר הצמתים המינימלי של עץ AVL בגובה 4 ונקבל:

$$W4 = W3 + W2 + 1 = W2 + W1 + 1 + W2 + 1 = 2W2 + W1 + 2 = 8 + 2 + 2 = 12$$

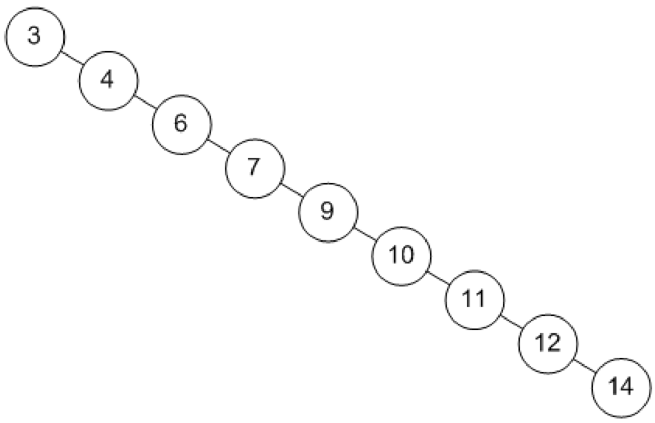
קיבלנו שמספר הצמתים המינימלי עבור עץ AVL בגובה 4 הוא 12.

סעיף ד': הטענה אינה נכונה

בהינתן עץ חיפוש בינארי כללי כדי לדעת אם הוא מקיים את תכונת האיזון נצטרך לבדוק:

1. בדיקת הגובה של תת העץ הימני.
2. בדיקת הגובה של תת העץ השמאלי.
3. בדיקה האם הפרש הגבהים בין תתי העצים הוא לכל היותר 1.

נשים לב שבמקרה הגרוע הדרך יכול לקחת O(n), לפי הדוגמה הבאה:



במקרה זה כל צמתי העץ נמצאים בתת העץ הימני ולכן בבדיקת הגובה של תת העץ הימני בהכרח נעבור על כל צמתי העץ, כלומר זמן הריצה יהיה O(n).

שאלה 2:

סעיף א':

השיטה תקבל את שורש העץ unfixedNode(node root), תחפש את הצומת המינימלי בעץ (יסומן min) ואת הצומת המקסימלי (יסומן max) לאחר מכן נשלח לפונקציה רקורסיבית המקבלת את שורש העץ, min ו-max. השיטה הרקורסיבית מתוארת ע"י פסואדו קוד:

unfixedNode (Node root, Node min, Node max)

```
If (root = NULL)
        return NULL
If (root.key < min.key)
        return min
If (root.key > max.key)
        return max
Else nodeLeft = unfixedNode(root.left, min, root)
       If (nodeLeft != NULL)
           return nodeLeft
       Else return unfixedNode(root.right, root, max)
```

ניתוח זמן ריצה: השיטה תחפש בתחילה את הצומת המינימלי והצומת המקסימלי בעץ, במקרה הגרוע נעבור על כל העץ, כלומר O(n). לאחר מכן נשלח לפונקציה הרקורסיבית unfixedNode(node root, node min, node max) את השורש והצמתים המינימלי המקסימלי. השיטה תבדוק עבור צומת בעץ האם המפתח קטן מערך המינימום האפשרי לצומת או המפתח גדול מערך המקסימום האפשרי לצומת ותעצור כאשר נגיע לעלה או כאשר נמצא צומת מקולקל כזה. כלומר במקרה הגרוע נעבור על כל צמתי העץ עד שנגיע לצומת המקולקל כלומר O(n). לכן בסך הכל זמן הריצה הוא O(n).

סעיף ב':

יהי T "עץ חיפוש מקולקל קלות" כלשהו, נתייחס לאלגוריתם שהוצע בסעיף א' בתור unfixedNode(Node node).

על מנת לתקן עץ מקולקל נבצע כמה פעולות:

1. נפעיל את השיטה unfixedNode(Node T.root) על שורש העץ T.
2. נקבל חזרה את צומת העץ המקולקל, נקרא לו unfixedNode.
3. נסרוק בסריקת inOrder את תת העץ המושרש של הבן הימני, ונמנה את כמות הצמתים (נניח m).
4. ניצור מערך בגודל המתאים.
5. נסרוק בסריקת inOrder את תת העץ המושרש של הבן הימני, הפעם נכניס בהתאם לסריקה כל צומת למקום המתאים במערך.
6. באופן סימטרי נעשה זאת עבור תת העץ המושרש של הבן השמאלי (נניח k).
7. כעת, קיבלנו שני מערכים ממוינים. ניצור מערך חדש באורך כגודל שני המערכים והשורש המקולקל (m+k+1).

8. נבצע merge על שני המערכים והשורש, כלומר נבצע השוואה על כל איבר במערך הראשון עם השורש ועם האיבר המתאים במערך השני. לבסוף נקבל מערך ממוזג וממויין המכיל את כל איברי תת העץ המקולקל.
9. נבצע סריקת inOrder, הפעם נכניס לכל קודקוד את המפתח המתאים לו לאחר המיון.

השמת המפתחות לאחר המיון מבטיחה לשמור על מבנה העץ (אנחנו לא מבצעים מחיקה ולא נדרשים לאזן את העץ מכל סיבה שהיא) אלא אנחנו דורסים את המפתחות הקיימים בעץ ומכניסים אליהם את המפתחות המתאימים אילו היה עץ חיפוש בינארי.

<u>ניתוח זמן ריצה:</u>

השיטה unfixedNode(root) מסעיף 2א' רצה בזמן O(n).

לאחר מכן כשנקבל את הצומת המקולקל ונבצע סריקות inOrder כל סריקה כזאת תקח O(n) במקרה הגרוע בו שורש העץ הוא הצומת המקולקל.

הפעולה merge רצה בזמן O(n) כיוון שעבור כל איבר במערך הסורק את תת העץ של הבן השמאלי נבצע השוואה עם השורש, עם האיבר המתאים במערך השני ובמידת הצורך גם בין השורש לאיבר המתאים במערך השני. ניתן לומר שעבור כל איבר נבצע מספר קבוע של השוואות, כלומר O(n) במקרה הגרוע.

לבסוף השמת המפתחות במערך (גם היא בסריקת inOrder) תיקח O(n).

סה"כ זמן הריצה של האלגוריתם הוא O(n).

<u>שאלה 3</u>

אלגוריתם הממיין את התור Q1:

Q2 = new queue()

i=0 ,x=null ,size=1 ,min=null

min=Q2.dequeue()

כל עוד Q2 לא ריק בצע:

X= Q2.dequeue()

אם x<min אז:

Q1.enqueue(min)

min=x

אחרת:

Q1.enqueue(x)

size++

בצע size-1 פעמים:

אם Q1 ריק:

בצע i פעמים:

x=Q2.dequeue()

Q1.enqueue(x)

Q1.enqueue(min)

min=Q2.dequeue()

כל עוד Q2 לא ריק בצע:

x= Q2.dequeue()

אם x<min אז:

Q1.enqueue(min)

min=x

אחרת:

Q1.enqueue(x)

אחרת:

בצע i פעמים:

x=Q1.dequeue()

Q2.enqueue(x)

Q2.enqueue(min)

min=Q1.dequeue()

כל עוד Q1 לא ריק בצע:

x= Q1.dequeue()

אם x<min אז:

Q2.enqueue(min)

min=x

אחרת:

Q2.enqueue(x)

i++

אם Q1 ריק

Q2.enqueue(min)

החזר את Q2

אחרת

Q1.enqueue(min)

החזר את Q1

<u>ניתוח זמן ריצה</u>

ניתן לראות בתחילת האלגוריתם לולאה שמתבצעת n פעמים ובתוכה מספר פעולות קבוע.

לאחר מכן ישנה לולאה שמתבצעת n-1 פעמים ובתוכה מספר פעולות קבועות ועוד 2 לולאות שמתבצעות ביחד סה"כ n-1 פעמים כאשר יש בתוכן מספר פעולות קבוע.

בסה"כ קיבלנו שבמקרה הגרוע ביותר מתבצעות

$$c+n-1+(n-1)*\left(c+(n-1)\right)=c+nc-c+n^2-n$$

כלומר זמן הריצה הוא $O(n^2)$

<u>שאלה 4</u>

צ"ל: ניתן להפוך עץ חיפוש בינארי נתון כלשהו, בעל n צמתים לעץ חיפוש בינארי אחר שצורתו נתונה, תוך שימוש ב- $O(n)$ רוטציות.

הוכחה:

תחילה נראה כי נחוצות m<n רוטציות כדי להפוך עץ בעל n צמתים לשרשרת ימנית. נוכיח באינדוקציה על n.

מקרה בסיס n=1, כמובן שהעץ הוא שרשרת ימנית ולכן דרוש m=0 רוטציות.

מקרה בסיס n=2, או שקיבלנו עץ שהוא שרשרת ימנית ואז דרוש m=0 רוטציות או שהעץ הוא שרשרת שמאלית ואז דרוש m=1 רוטציות ובכל מקרה m<n.

הנחת האינדוקציה, נניח כי עבור עץ בעל n-1 צמתים נחוצות m<n-1 רוטציות כדי להפוך את העץ לשרשרת ימנית ונוכיח עבור עץ בגודל n.

יהי עץ חיפוש בינארי כלשהו בעל n צמתים. נתקדם מהשורש לבן הימני עד אשר נגיע לצומת ללא בן ימני והוא יהיה הצומת בעל הערך המקסימלי בעץ(פעולה שלוקחת $O(n)$) נשמיט את הצומת הזו(פעולה שלוקחת $O(1)$ – הסבר ב-*). קיבלנו עץ בעל n-1 צמתים, לפי הנחת האינדוקציה נחוצות m<n-1 רוטציות על מנת להפוך אותו לשרשרת ימנית. נהפוך אותו לשרשרת ימנית ב-m רוטציות ולאחר מכן נוסיף את הצומת בעל הערך המקסימלי שהשמטנו לסוף השרשרת(פעולה שלוקחת $O(n)$).

עד כה הוכחנו כי ניתן להפוך עץ חיפוש בינארי בעל n צמתים לעץ חיפוש בינארי שהוא שרשרת ימנית בזמן ריצה של $O(n)$ רוטציות ($O(n)$ כי כל רוטציה היא מספר קבוע של פעולות קבועות). כעת נוכיח כי ניתן להפוך עץ כלשהו בעל n צמתים שהוא שרשרת ימנית לעץ חיפוש בינארי שצורתו נתונה תוך שימוש ב m<n רוטציות. נוכיח באינדוקציה על n.

מקרה בסיס n=1, ישנה רק צורה אחת אפשרית ולכן נחוצות m=0 רוטציות.

מקרה בסיס n=2, ישנן 2 צורות אפשריות לעץ, או שרשרת ימנית ואז נחוצות m=0 רוטציות או שרשרת שמאלית ואז נחוצות m=1 רוטציות ובכל מקרה m<n.

הנחת האינדוקציה, נניח כי ניתן להפוך עץ כלשהו בעל n-1 צמתים שהוא שרשרת ימנית לעץ חיפוש בינארי שצורתו נתונה תוך שימוש ב m<n-1 רוטציות ונוכיח עבור עץ בגודל n.

יהי עץ חיפוש בינארי שהוא שרשרת ימנית בגודל n וצורה של עץ חיפוש בינארי.

נשמיט את הצומת הימנית ביותר בשרשרת, שהוא גם הערך המקסימלי בשרשרת(פעולה שלוקחת $O(n)$) נשמיט מהצורה את הצומת הימני ביותר(שאמור להיות בעל הערך המקסימלי בעץ חיפוש בינארי). כמו במקרה לעיל, אם הצומת בצורה הוא עלה אז פשוט נשמיטו ואם יש לו בן שמאלי, נשמיטו ונציב את הבן השמאלי במקומו.

כעת עבור השרשרת בעלת n-1 צמתים והצורה החדשה, ניתן להפוך את השרשרת לעץ בצורה החדשה תוך m<n רוטציות. כעת נחזיר את הצומת המקסימלי שהשמטנו מהשרשרת למקום שהשמטנו מהצורה המקורית(פעולה שלוקחת $O(n)$ להגעה למקום המבוקש ועוד מספר פעולות קבועות להצבתו במקום המיועד).

כעת הוכחנו כי ניתן להפוך עץ חיפוש בינארי בעל n צמתים שהוא שרשרת ימנית לעץ חיפוש בינארי עם צורה נתונה בזמן ריצה של $O(n)$ ($O(n)$ רוטציות כי כל רוטציה היא מספר קבוע של פעולות קבועות).

בסה"כ קבלנו שניתן להפוך עץ חיפוש בינארי נתון לעץ חיפוש בינארי שצורתו נתונה ב - $O(n) = O(n) + O(n)$ ($O(n)$ רוטציות כי כל רוטציה היא מספר קבוע של פעולות קבועות).

מש"ל.

*ישנן 2 אפשרויות, או שהצומת עלה ואז פשוט תהפוך ל-null או שלצומת יש בן שמאלי ואז נציב את הבן השמאלי להיות במקום הצומת.

<u>שאלה 5</u>

מבנה הנתונים יורכב באמצעות 2 עצי AVL:

- עץ AVLG המכיל את מיקומי הענקים.
- עץ AVLD המכיל את מיקומי הגמדים.

כאשר בעץ AVLD בכל צומת יהיו גם מצביעים לעוקב ולקודם.

- Init() – ניצור שני עצי AVL ריקים. זמן ריצה O(1).
- InsertDwarf(location) –

נמצא מצביע לעוקב: נחפש בעץ AVLD את הצומת שערכו הכי קטן וגם ערכו גדול מ- location of new dwarf (אם קיים) באופן הבא:
נתחיל לסרוק את העץ מהשורש עד אשר נגיע לעלה כאשר ההתקדמות היא:
אם location of new dwarf >location נתקדם לבן הימני.
אם location of new dwarf <location נתקדם לבן השמאלי ונציב במשתנה עזר x את הערך של location .
כאשר נגיע לעלה, גם בו נבצע את הבדיקה של
אם location of new dwarf <location נציב במשתנה עזר x את הערך של location

ולאחר מכן x יהיה העוקב. באותו אופן נמצא את הקודם (נחפש בעץ AVLD את הצומת שערכו הכי גדול וגם ערכו קטן מ-location of new dwarf (אם קיים)). כמו כן, כשנגיע לעוקב/קודם, נעדכן את המצביע לקודם/עוקב בהתאמה.

*זמן הריצה של מציאת עוקב/קודם הוא כגובה העץ * מספר פעולות קבועות, כלומר גובה העץ ומכיוון שמדובר בעץ AVL אז זמן הריצה הוא O(log n).*

נכניס את הערך location לתוך AVLD. כפי שלמדנו בכתה, זמן הכנסה של ערך לתוך עץ AVL הוא לכל היותר O(log n) מכיוון שגובה העץ הוא log n לכל היותר.

בסה"כ קבלנו זמן ריצה של O(log n).

- InsertGiant(location) - נכניס את הערך location לתוך AVLG. כפי שלמדנו בכתה, זמן הכנסה של ערך לתוך עץ AVL הוא לכל היותר O(log n) מכיוון שגובה העץ הוא log n לכל היותר.

- *IsTalking(L1,L2) – תחילה נבצע 2 חיפושים בעץ AVLD כדי לבדוק את קיום 2 הגמדים, אם אחד מהם לא קיים נחזיר שגיאה, זמן ריצה של חיפוש בעץ AVL הוא כפי שלמדנו O(log n).*
כעת נכניס למשתנה min את המינימאלי מבין L1 ו L2 ונכניס למשתנה max את השני.
נחפש בעץ AVLG את הצומת שערכו הכי קטן וגם ערכו גדול מ-min (אם קיים) באופן הבא:
נתחיל לסרוק את העץ מהשורש עד אשר נגיע לעלה כאשר ההתקדמות היא:
אם location<min נתקדם לבן הימני.
אם location>min נתקדם לבן השמאלי ונציב במשתנה עזר x את הערך של location .

כאשר נגיע לעלה, גם בו נבצע את הבדיקה של אם location>min נציב במשתנה עזר x את הערך של location ולאחר מכן x יהיה הצומת הכי קטן שערכו גדול מ-min (אלא אם x=null ואז אין בעץ ערך גדול מ-min , במקרה זה בהכרח הגמדים יכולים לדבר ונחזיר true). כעת אם x<max נחזיר false ואחרת נחזיר true.
*זמן הריצה של פעולה זו הוא כגובה העץ * מספר פעולות קבועות, כלומר גובה העץ ומכיוון שמדובר בעץ AVL אז זמן הריצה הוא O(log n).*
*בסה"כ קבלנו O(log n) * 3 שזה זמן ריצה של O(log n).*

- *Remove(location) – נבצע מחיקה בעץ AVLG כפי שלמדנו על מחיקה בעץ AVL. אם לא קיים יצור במייקום זה, נבצע מחיקה בעץ AVLD, כמו כן נעדכן את המצביעים של העוקב והקודם(O(1)). אם גם בעץ זה לא קיים יצור, נחזיר הודעת שגיאה. כפי שלמדנו, מחיקה של צומת בעץ AVL לוקחת זמן ריצה של O(log n).*

- *WhomTalking(location) –*
  א. נמצא בעץ AVLD את הצומת העוקב של location, נסמנו min.
  ב. נמצא בעץ AVLD את הצומת הקודם של location, נסמנו max.
  ג. נמצא בעץ AVLG את הצומת שערכו הכי קטן וגם שערכו גדול מ- min.
  ד. נבצע סריקת בעזרת המצביעים לקודם ועוקב החל מאיבר זה(ג') ועד האיבר שערכו הכי גדול וגם קטן מ- max ונדפיס כל איבר כזה.

<u>א –</u>

בדיוק כמו בפונקציה *IsTalking, זמן ריצה של O(log n).*

<u>ב -</u>

בדיוק כמו בפונקציה *IsTalking, עם הבדל אחד והוא שהפעם בזמן הסריקה אם location<min נציב במשתנה עזר x את הערך של location . זמן ריצה של O(log n).*

ג -

בדיוק כמו בא', זמן ריצה של $O(\log n)$.

ד -

זמן ריצה של $O(k)$.

בסה"כ קבלנו זמן ריצה של $O((\log n) + k)$.