מבני נתונים – משימה 3

מאור אסייג 318550746

רפאל שטרית 204654891

המחלקה להנדסת חשמל ומחשבים, התכנית להנדסת מחשבים

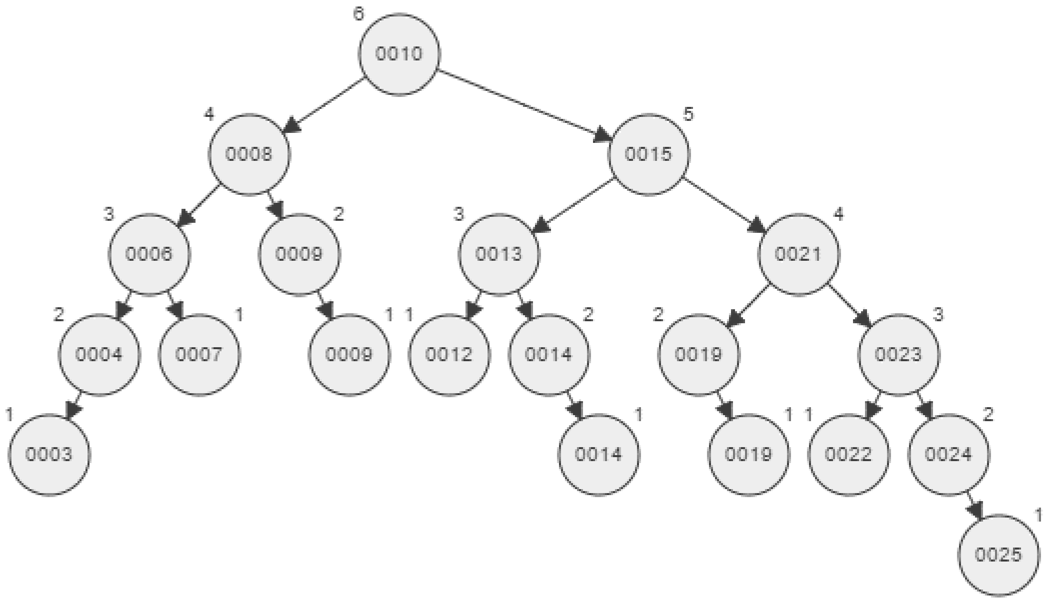
מבני נתונים 202.1.1011

תשובות

1. הוכיחו או הפריכו

א. אם T עץ AVL בגובה h , אז כל הרמות עד הרמה ה- $h-2$ (כולל) מלאות.

לא נכון, דוגמה נגדית :



העץ הינו עץ AVL וגובהו 5, אולם הרמה ה 5-2=3 בעלת 7 צמתים, בסתירה לטענה שהרמה צריכה להיות מלאה ($2^3 = 8 > 7$).

ב. פעולת המחיקה מעץ חיפוש בינארי היא חלופית.

נכון, נוכיח על ידי פירוט המקרים האפשריים במחיקת הצמתים x ו y,:

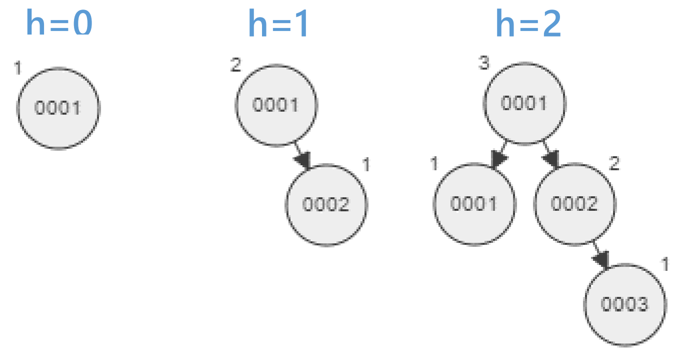
1. נניח ש **x בתת העץ הימני של השורש ו y בתת העץ השמאלי של השורש.** לפי אלגוריתם המחיקה סדר מחיקתם לא תשפיע על התוצאה הסופית כיוון שמחיקת האחד לא תשנה את העוקב של האחר(במקרה ויש לאחת הצמתים 2 בנים, אחרת סדר המחיקה גם לא ישנה את מחליפי הצמתים) אזי מחיקת צמתים בתתי עצים שונים בלתי תלויה, כלומר חלופית.
2. בלי הגבלת הכלליות נניח ש_**x בתת העץ השמאלי של y.** אם ל y 2 בנים : נחליף את y בעוקב מתת העץ הימני, כלומר מחיקת x לא תשפיע על העוקב בתת העץ הימני לכן סדר המחיקה לא ישנה.
אם ל y בן שמאלי בלבד : נחליף את y בבן שלה, נסמן את המחליף של x במחיקתו כ z. נשים לב שגם אם x הוא הבן של y, בסוף במקום y יהיה z, אחרת אין כלל תלות בין הצמתים הנמחקים.
3. נניח ש **x בתת העץ הימני של y,** תלות בין הצמתים יהיה רק תחת ההנחה ש x הינו העוקב של y, או ש x הינו העוקב של העוקב של y.
   - x הינו העוקב של y : נסמן את העוקב של x ב z. במחיקת y נחליף אותו ב x, ובמחיקת x נחליף אותו ב z. ובמקרה ההפוך במחיקת x נחליף אותו ב z, ובמחיקת y נחליף אותו ב z.
   - x הינו העוקב של העוקב של y : תפקיד זהה למקרה לעיל, רק האותיות x ו z מוחלפות.

ובאופן זהה עבור המקרים 1,2,3 אם לצומת אין 2 בנים נחליף אותה בבנה היחיד, כך שהטענות נשארות עדיין תקפות.
בסה"כ סדר המחיקה אינו משפיע על התוצאה הסופית, ולכן פעולת המחיקה בעץ חיפוש הינה **חלופית**.

ג. המספר המינימאלי של צמתים בעץ AVL בגובה h=4 הוא 12.

נכון, נסמן n(x) כמספר הצמתים המינימאלי בעץ AVL עבור גובה x.

- $n(0)=1$
- $n(1)=2$
- $n(2)=4$

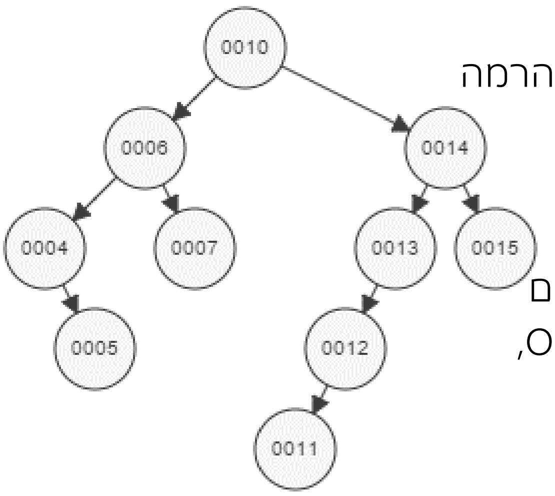


$$n(h) = 1 + n(h-1) + n(h-2)$$
$$total\ nodes : current\ + left\ sub\ tree + right\ sub\ tree$$
$$n(4) = 1 + n(3) + n(2) = 1 + [1 + n(2) + n(1)] + n(2)$$
$$= 2 + 4 + 4 + 2 = 12.$$

ד. רוצים לבדוק האם BST מקיים את תכונת האיזון של עץ AVL. אזי זמן הריצה של האלגוריתם היעיל ביותר לבדיקה הנ"ל במקרה הגרוע הוא $\Theta(log\ n)$.

- הנחנו שלכל צומת ישנן התכונות של BST, ללא שדה המצביע על איזון הגבהים.

לא נכון, דוגמה נגדית :
נשים לב שתכונת האיזון מופרת לקראת הרמה האחרונה. כל אלגוריתם חיפוש יצטרך לחשב עבור כל צומת נוכחית את הגובה של 2 הבנים ולבדוק את ההפרש, על כן נצטרך במקרה הגרוע לבקר בכל הצמתים בעץ, כלומר בדוגמא הזו סדר גודל של O(n), בסתירה להנחה שהזמן ריצה $\Theta(log\ n)$.



2. עץ חיפוש מקולקל קלות

א. מציאת צומת מקולקל

הפונקציה $Finder(node)$ מחזירה צומת מקולקל :

1. אם השורש ריק(=העץ ריק), החזר שגיאה.
2. אחרת, קרא לפונקציה הרקורסיבית שמחזירה צומת $Find(node, min, max)$ עם הערכים $Find(T.root, -\infty, +\infty)$ והחזר את התוצאה.

הפונקציה $Find(node, min, max)$ מחזירה צומת מקולקל :

1. אם $node = null$ החזר $null$.
2. אחרת, בדוק אם $min \leq node.value \leq max$
   - 2.1 אם לא מתקיים החזר את $node$.
   - 2.2 אם מתקיים החזר את מה שלא יהיה $null$ מבין : $Find(node.left, min, node.value)$ or $Find(node.right, node.value, max)$.
     - 2.2.1 אם שתיהן $null$, החזר $null$.

מההנחה שבעץ יש לפחות צומת מקולקל אחד, תוחזר לפונקציה הראשית הצומת המקולקלת.

זמן ריצה : במקרה הגרוע נבקר בכל הצמתים עד העלים (כמובן שניתן לייעל מעט את האלגוריתם, אך זה לא ישפיע על זמן הריצה) – כלומר נבקר בכל צמתי העץ תוך כדי ביצוע O(1) פעולות ולכן : $\mathrm{T}(n) = nO(1) = O(n),\ num\ of\ nodes = n$.

ב. תיקון עץ חיפוש מקולקל קלות

תחת ההנחה שאנו מקבלים עץ ובו צומת מקולקל קלות

Pseudo code :

$global\ array\ A;$

$global\ int\ i = 0$

$mainFunc\ (node\ root)$

$\quad v = Finder(root);$

$\quad left = inOrder(v.left);$

$\quad right = inOrder(v.right);$

$\quad A = mergeSort1(left, right, v.value);$

$\quad inOrder1(v);$

$$inOrder1(node\ current) \qquad //Aid - Function$$
$$\quad if\ (current = !\,null)$$
$$\qquad inOrder1(current.left);$$
$$\qquad current.value = A[i];$$
$$\qquad i++;$$
$$\qquad inOrder1(current.right);$$

$//Aid - Function$

$algorihtem\ for\ mergeSort1(arr\ \ left, arr\ \ right, int\ \ node)$

1. צור מערך חדש לתוך A בגודל $left.length + right.length + 1$
2. צור משתנים המסמלים אינדקס נוכחי במערכים left, right ומשתנה בוליאני המסמן אם הכנסנו כבר את node.
3. **רוץ** מ $j = 0$ עד $j < \mathrm{A.length}$
   - **3.1** הכנס ל A[j] את המינימום מבין הערכים המותרים שנותרו,במידה וזה אחד מאיברי המערכים קדם את האינדקס של אותו מערך המכיל את המינימום הנוכחי.

     $currentMin = \min(left[j1], right[j2], node)$

     מינימום נוכחי : left באינדקס הנוכחי (או שכבר הכנסנו את כל left) או **right** באופן זהה או הערך **node** (כל עוד לא נבחר כבר, כלומר כל עוד הוא לא יצא המינימום הנוכחי, הכוונה לא למספר שהוא מייצג בכל השוואה).

הסבר מילולי על האלגוריתם בשלמותו : ראשית נסדר את כלל הערכים בעץ החל מהצומת המקולקל במערך A, זאת נעשה על ידי קריאה ל inOrder עבור כל תת עץ של הצומת המקולקל v וקריאה לפונקציה המאחדת את אותם מערכים והערך של v לתוך A. מכיוון שהמערכים כבר ממוינים, זמן הריצה שלה יהיה בהתאם למספר האיברים הכולל כלומר O(n).

שנית מכיוון שאנו רוצים לשמור על מבנה העץ, נרוץ על מבנה העץ הנוכחי בעזרת inOrder1 – אופן הפעולה שלו זהה לחלוטין ל inOrder

הרגיל, אך כעת נחליף את הערך בצומת אליה נגיע לערך היחסי מהמערך A. לדוגמה במקום הדפסה של הערך הקטן ביותר בתת העץ השמאלי הנוכחי של v אנו נשים לתוך אותה צומת את הערך המינימלי שנמצא מ v ומטה, כלומר המיקום הנכון שיקיים את תכונות BST, וכן הלאה.

ניתוח זמן ריצה : הפונקציה inOrder מהכיתה *O(n)*. מכיוון שאנו קוראים ל *mergeSort1* עם 2 מערכים ממיונים וערך נלווה, שילוב כלל הערכים למערך אחד ממוין ייערך כסדר גודל של מספר האיברים הכולל, כלומר סדר גודל של *O(n)* במקרה הגרוע. *InOrder1* במקרה הגרוע יחל מהשורש, כלומר נעבור בסופו של דבר בכל האיברים ובאופן זהה נקבל גם *O(n)*. בסה"כ :

$$T(n) = 2O(n) \qquad + O(n) \qquad + O(n) \quad = O(n).$$
$$\{2\ inOrder\ \ + mergeSort1\ \ + inOrder1\}$$

3. מיון תור בעזרת תור עזר

אלגוריתם :

1. כל עוד *Q1* לא ממוין :
1.1 הוצא את האיבר הראשון ב *Q1*.
1.2 כל עוד *Q1* מלא :
1.2.1 נוציא את האיבר הבא ב*Q1* לשם השוואה .
1.2.2 נשווה בין 2 האיברים, את הקטן מבניהם נכניס ל *Q2*.
1.3 כל עוד *Q2* מלא :
1.3.1 נוציא את האיבר הבא ב*Q2* לשם השוואה.
1.3.2 נשווה בין 2 האיברים, את הקטן מבניהם נכניס ל *Q2*.
1.3.3 אם האיברים לא מסודרים בסדר עולה, *Q1* לא ממוין.

המשך ->

Pseudo code :

$Find\ (queue\ Q1)$

```
    first, second, isSorted = false;
    while (! = isSorted){
        second = Q1.dequeue();
        while (!Q1.isEmpty( )){
            first = second;
            second = Q1.dequeue( );
            if(first > second)
                Q2.enqueue(second);   \\ לשם ההשוואה הבאה
                second = first;
            else
                Q2.enqueue(first);
        }
        isSorted = true;
        second = Q2.deuqeue;
        while (!Q2.isEmpty( )){
            first = second;
            second = Q2.dequeue( );
            if(first > second)
                isSorted = false;   \\ כרגע התור לא ממוין, חזור על התהליך
                Q1.enqueue(second);
                second = first;   \\ לשם ההשוואה הבאה
            else
                Q1.enqueue(first);
        }
    }
    return Q1;
```

בחירת האלגוריתם : הרעיון הטריוויאלי הינו brute force – מציאת המינימום הנוכחי בריצה על כל איברי Q1 הנותרים, למחוק אותו מ Q1 ולהכניס אותו ל Q2. גישה זו תענה על הנדרש בזמן ריצה של $\Theta(n^2)$, ואמנם רצינו להציע אלגוריתם שזמן הריצה שלו הינו O(n^2) במקרה הגרוע (אותו מקבלים בסבירות נמוכה, בהתאם למספר האיברים).

המשך ->

הסבר מילולי : המיון הינו מיון מבוסס השוואת המזכיר את פעולת **bubble sort**. באיטרציה הראשונה (ה while הראשון) אנו בודקים יחסי סדר גודל בין כל 2 עוקבים שנותרו בתהליך (ראשית האיבר הראשון והשני, בהגב"כ השני והשלישי, השלישי והרביעי וכן הלאה) ובצורה דומה כך גם באיטרציה השנייה (תוך כדי נבדוק אם המערך ממוין לשם הפסקת התהליך). חוזרים על 2 האיטרציות עד שהתור Q1 יכיל את האיברים בצורה ממוינת.

4. הראו שניתן להפוך עץ חיפוש בינארי נתון כלשהו, בעל *n* צמתים לעץ חיפוש בינארי אחר שצורתו נתונה, תוך שימוש ב *O(n)* רוטציות.

אלגוריתם יצירת שרשרת ימנית :

1. קרא לאלגוריתם עם השורש של העץ.
2. הגדר שורש זמני שתפקידו להצביע על החוליה שמבצעים עליה את הסיבוב הימני (כפי שהוגדרת בקורס *Rotate Right*)
3. בדוק האם קיים **בן שמאלי** לשורש הזמני

3.1 אם כן :

3.1.1 בצע סיבוב ימינה, הגדר את הבן השמאלי (של השורש הזמני) כשורש הזמני החדש.

3.1.2 חזור על שלב 3 עבור השורש הזמני החדש.

3.2 אחרת בדוק האם קיים **בן ימני** :

3.2.1 אם לא קיים - סיים את התהליך, אזי נוצרה שרשרת ימנית.

3.2.2 אם כן – הגדר אותו כשורש זמני ובצע את שלב 2 עליו.

- במקרה הגרוע נגדיר *n-1* פעמים שורש זמני חדש ונבצע עליו סיבוב ימני, לכן נצטרך *n-1* רוטציות.

המשך ->

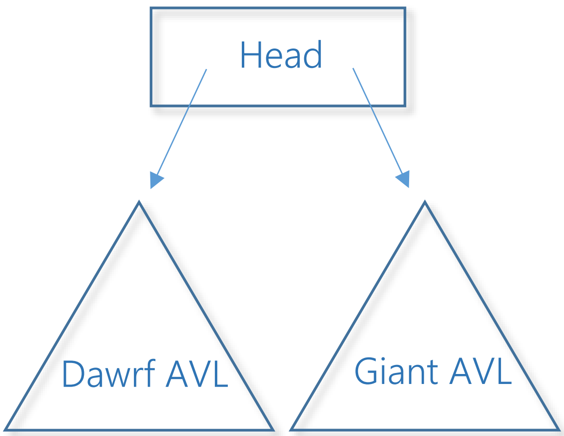
אלגוריתם המרת עץ חיפוש 1 לעץ חיפוש 2 בעזרת סיבובים :

1. צור שרשרת ימנית של עץ 1 בעזרת האלגוריתם לעיל.
2. צור שרשרת ימנית של עץ 2 בעזרת אלגוריתם זהה ששומר את הפעולות שנעשו **במערך עם 2 שדות מידע** – פעולת הסיבוב (*right/left*) והמידע השייך לצומת שהוגדר כשורש הזמני.
3. גש לאיבר האחרון בשרשרת ה 1 ובשרשרת 2.
4. נעבור על המערך החל מהאיבר האחרון, נחפש את האיבר הזה בשרשרת ה 2 – כל התקדמות בשרשרת 2 נתקדם גם בשרשרת 1. אם הגענו לחוליה בשרשרת 2 שהמידע שלה שמור בתא הנוכחי במערך נבצע את הסיבוב הנגדי המופיע במערך על החוליה בשרשרת 1.

- בצורה זו נוכל לקבל את המבנה של עץ 2 עם המידע של עץ 1 בעזרת ***O(n)=O(n)+O(n)*** **רוטציות**.

5. גמדים וענקים

תיאור מבנה הנתונים : בחרנו להשתמש ב 2 עצי AVL, עץ AVL לגמדים ועץ AVL לענקים. גישה לעצים תינתן על ידי חוליה מיוחדת שתחזיק 2 מצביעים לשורשי העצים.



נגדיר $Head.left = Dawrf.root,\ \ Head.right = Giant.root$.

אלגוריתמי הפונקציות :

$Init(\quad)$

1. ניצור חוליה עם 2 מצביעים.
2. ניצור 2 עצי AVL ריקים.
3. נקשר את 2 המצביעים לשורשי העצים הריקים.

ניתוח זמן ריצה : כל שלב הינו מספר סופי של פעולות, לכן **בסה"כ O(1)**.

$InsertDwarf\ (location)$

1. קרא לפונקציה $Head.left.Insert(location)$ (עבור תת העץ השמאלי שמייצג את עץ הגמדים).

ניתוח זמן ריצה : מההרצאה פעולת $Insert$ בעץ AVL לוקחת $O(h)=O(log\,n)$ ועבור המקרה הגרוע נקבל בסה"כ $O(log\,n)$.

$InsertGiant\ (location)$

1. קרא לפונקציה $Head.right.Insert(location)$ (עבור תת העץ הימני שמייצג את עץ הענקים).

ניתוח זמן ריצה : מההרצאה פעולת $Insert$ בעץ AVL לוקחת $\mathrm{O}(h)=O(log\,n)$ ועבור המקרה הגרוע נקבל **בסה"כ** $O(log\,n)$.

$IsTalking(L1, L2)$

1. קרא לפונקציה $Head.left.Search(L1)$ (עבור תת העץ השמאלי שמייצג את עץ הגמדים) ושמור את הערך במשתנה $first$.
2. קרא לפונקציה $Head.left.Search(L2)$ (עבור תת העץ השמאלי שמייצג את עץ הגמדים) ושמור את הערך במשתנה $second$.
3. מצא את $\min(\text{first}, \text{second})$ והגדר אותו כ low, ואת הנותר כ $high$, במידה ומישהו מהם $null$ הוצא הודעת שגיאה וצא מהפונקציה.
4. קרא ל $Find(Head.right, low, high)$ המחזירה משתנה בוליאני שבודק האם ישנו ערך בתת העץ של הענקים שנמצא בתחום $[\text{low}, \text{high}]$.

ניתוח זמן ריצה : מההרצאה פעולת $Search$ בעץ AVL לוקחת $\theta(h) = \theta(log\, n)$, פעולת $\min(\text{first}, \text{second})$ תיקח $O(1)$ ופעולת $Find(Head.right, low, high)$ במקרה הגרוע תבקר בכל רמות העץ, מכיוון שהעץ הינו עץ AVL גובהו לכל היותר $log(n)$ ולכן **בסה"כ** :

$$T(n) = 2\theta(log\, n) + O(1) + O(\log n) = O(log\, n).$$

$Find(node\ currentNode, low, high)$ $//aid\ function$

- $Find$ תחזיר משתנה בוליאני שיסמל האם נמצא ערך בעץ בתחום $[\text{low}, \text{high}]$.

1. אם $currentNode.value$ בתחום החזר $false$.
2. אם $currentNode.value > high$:
   - **3.1** אם קיים $currentNode.right$ החזר $Find(currentNode.right, low, high)\&\&true$.
   - **3.2** אחרת החזר $true$.
4. אם $currentNode.value < low$:
   - **4.1** אם קיים $currentNode.left$ החזר $Find(currentNode.left, low, high)\&\&true$.
   - **4.2** אחרת החזר $true$.

ניתוח זמן ריצה : עקרון הפעולה הינו זהה ל$Search$ בעץ AVL, אומנם כעת אנו מתנים תנאי עצירה ולכן במקרה הגרוע נצטרך לבקר בכל רמות העץ, **ובסה"כ** $T(n) = O(log\ n)$.

$Remove\ (location)$

1. צור משתנה בוליאני $Giants = Head.right.Search(location)$ (עבור תת העץ הימני שמייצג את עץ הענקים).
2. צור משתנה בוליאני $Dwarfs = Head.left.Search(location)$ (עבור תת העץ השמאלי שמייצג את עץ הגמדים).
3. אם $Giants = false\&\&Dawrfs = false$: החזר הודעת שגיאה.
4. אם $\text{Giants} = \text{true}$:
   4.1 קרא ל $Head.right.Delete(location)$ (מחיקה ב AVL).
5. אם $Dwarfs = true$:
   5.1 קרא ל $Head.left.Delete(location)$ (מחיקה ב AVL).

ניתוח זמן ריצה : מההרצאה פעולת $Search$ בעץ AVL לוקחת $O(h) = O(log\ n)$, פעולת $Delete$ $O(log\ n)$ ועבור המקרה הגרוע נקבל **בסה"כ**

$$T(n) = 2O(log\ n) + 2O(log\ n) = O(log\ n).$$

$WhomTalking\ (location)$

1. בכדי לגלות את 2 הענקים הקרובים ביותר לגמד, נכניס את הגמד לעץ הענקים $Head.right.Insert(location)$.
2. נצור משתנה $high = Head.right.Successor(location)$.
3. נצור משתנה $low = Head.right.Predecessor(location)$.
4. נמחק את הגמד מעץ הענקים $Head.right.Delete(location)$.
5. קרא לפונקציה שתדפיס את הגמדים בתחום שמצאנו $Print(low, high,\ Head.left.Search(location))$

- אם בשלב **2,3** לא קיים עוקב\קודם נעניק ערך $\infty\backslash-\infty$.

ניתוח זמן ריצה : פעולת $Insert$ $O(log\ n)$ וכן גם $Successor$, , $Delete, Predecessor$.
פעולת $Print$ לוקחת $O(k)$ כאשר k יהיה מספר האיברים בתחום
ולכן במקרה הגרוע נקבל **בסה"כ**

$$T(n) = 5O(log\ n) + O(k)$$
$$= O(\max\{\log n, k\}) = O(\log(n) + k).$$

$Print(low, high, currentNode)$

1. אם $\text{currentNode} = null$ צא מהפונקציה.
2. אחרת אם $low \leq \text{currentNode.value} \leq high$
   - **2.1** הדפס $currentNode.value$
3. אם $\text{currentNode.value} < low$ וגם יש לו בן שמאלי
   - **3.1** קרא ל $Print(low, high, currentNode.left)$
4. אם $\text{currentNode.value} > high$ וגם יש לו בן ימני
   - **4.1** קרא ל $Print(low, high, currentNode.right)$

ניתוח זמן ריצה : במקרה הגרוע נקבל את השורש של עץ AVL,
לכן הפונקציה תבקר בכל הערכים הנמצאים בתחום נסמנם k, לכן
בסה"כ $T(n\ node, k\ in\ range) = O(k)$.

Until next time, thank you.