איתמר קרייטמן ת.ז 208925578

מטלה 3

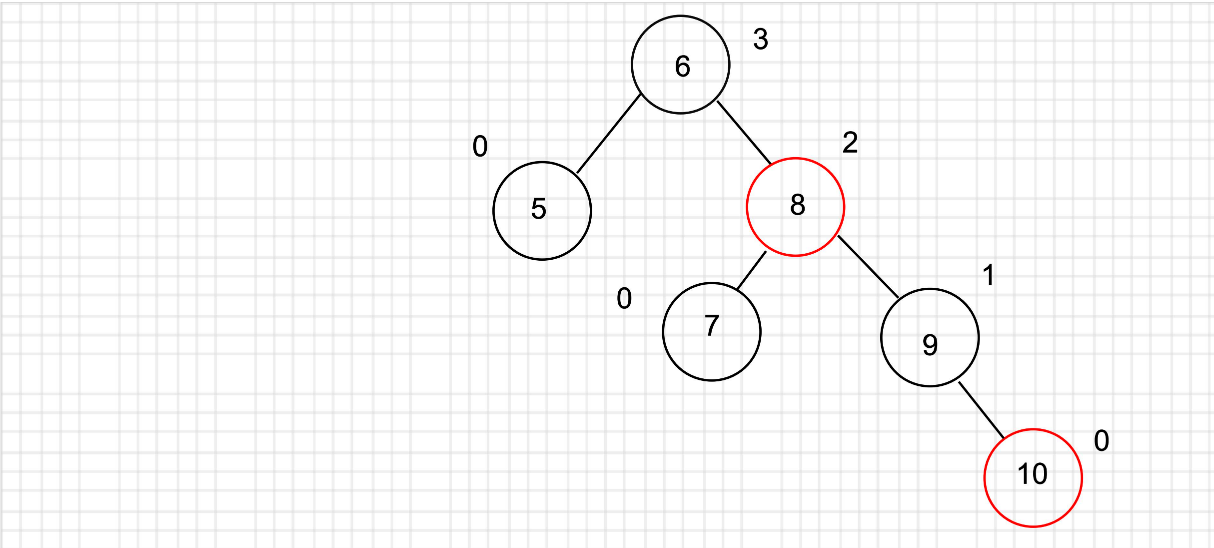
איתמר קרייטמן ת.ז 208925578

שאלה 1:

עץ AVL מקיים את התכונה שלכל צומת בעץ, ההפרש בין גבהי ילדיו הוא לכל היותר 1. האם עץ אדום-שחור מקיים את אותה תכונה? אם כן, הוכיחו זאת, ואם לא תנו דוגמה נגדית.

תשובה:

נפריך באמצעות דוגמא נגדית:



נשים לב כי הילדים של השורש, שהם הצומת עם הערך 8 ועם הערך 5, אינם באותו גובה. כאשר הילד הימני עם הערך 8 בגובה 2 והילד השמאלי עם הערך 5 בגובה 0.

$2 - 0 = 2 > 1$.

מכאן מובן כי התשובה לשאלה האם עץ אדום-שחור מקיים את אותה תכונה של עץ AVL היא שלא.

תשובה סופית: עץ אדום שחור אינו מקיים את תכונה שעבור כל צומת בעץ ההפרש בין גבהי ילדיו הוא לכל היותר 1.

איתמר קרייטמן ת.ז 208925578

שאלה 2:

שאלה 2:

לכל אחד מהסעיפים, ענו נכון או לא נכון, ונמקו את תשובתכם. (בשאלות אלו הנחנו שגובה עלה הוא 0.)

א. תת העץ השמאלי של שורש של עץ אדום-שחור תמיד מקיים את כל התכונות של עץ אדום שחור.

ב. אם יש לצומת בעץ אדום-שחור בדיוק ילד אחד (שאינו NULL), הילד תמיד אדום.

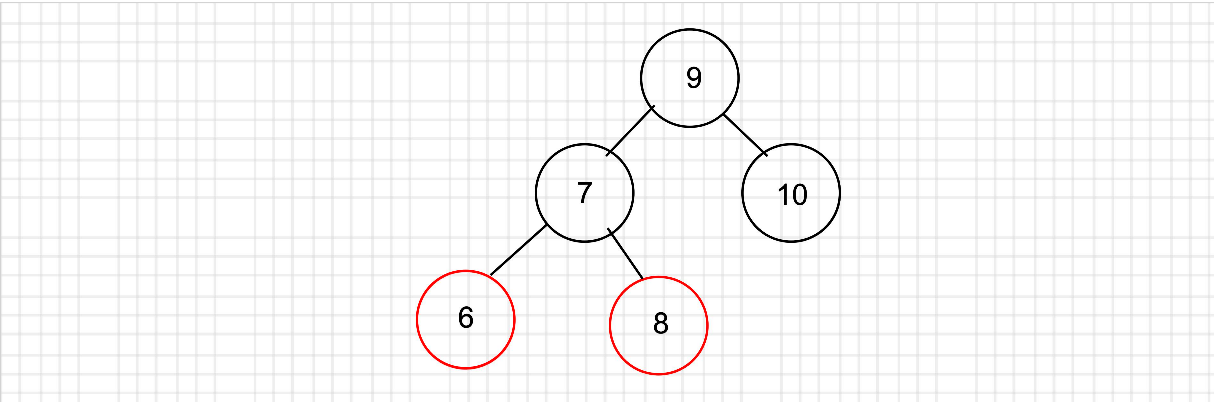
ג. לכל צומת שחור בעל גובה k יש לפחות $\lfloor k/2 \rfloor$ צאצאים שחורים.

ד. תת העץ השמאלי של שורש של עץ AVL תמיד מקיים את כל התכונות של עץ AVL.

ה. לכל צומת בגובה k בעץ אדום שחור יש לפחות $\lfloor k/2 \rfloor$ צאצאים אדומים.

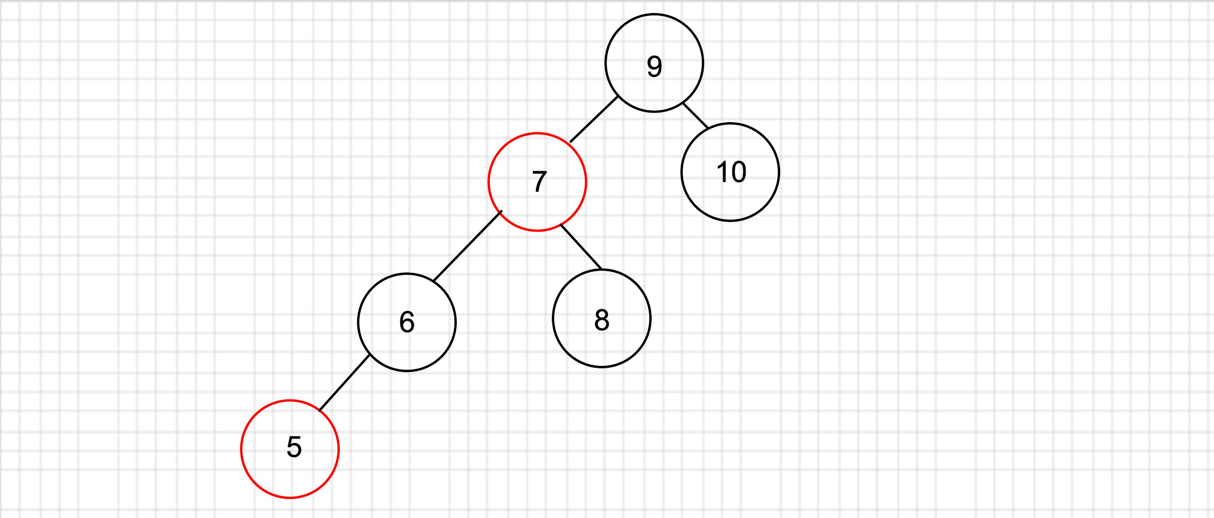
סעיף א':

לא נכון. ניקח מקרה נגדי בו הוכנס צומת חדש (עלה). הוא מוכנס כאדום, על פי כללי עץ אדום שחור, צבעו של אביו משתנה לשחור ושל דודו משתנה לשחור גם כן, ושל סבו לאדום. אך מכיוון שהסבא אינו השורש של העץ, צבעו אינו משתנה חזרה לשחור, ולכן התנאי של עץ אדום שחור- שורש העץ תמיד שחור אינו מתקיים.
דוגמא למתואר:



כאשר נרצה להכניס צומת חדש עם האיבר 5. 6 ו-8 יצבעו בשחור על פי כללי recoloring ו-7 שהוא הסבא יצבע באדום, ולא יחזור להיות שחור מכיוון שאינו השורש של העץ ויתקבל העץ:

איתמר קרייטמן ת.ז 208925578



נשים לב כי אכן תת העץ השמאלי אינו מקיים את התכונה של עץ אדום שחור שהשורש תמיד שחור מכיוון שהצומת עם הערך 7 הוא אדום והוא השורש של תת העץ השמאלי.

<u>סעיף ב:</u>

נכון. לפי חוקי עץ אדום שחור, כל צומת שמוכנס תמיד אדום. ואז מתחלקים לכמה מקרים:

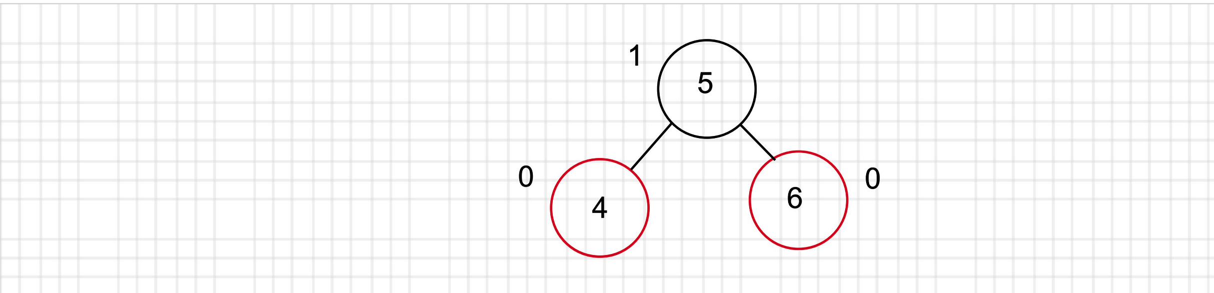
* במקרה וקיים דוד, והאבא והדוד אדומים שניהם, הם הופכים לשחורים והבן נשאר אדום.
* אם הדוד והאבא אינם אדומים, לא יתבצע כלום וכמו כן- הילד ישאר אדום (כמו כן אם האב שחור והדוד אדום).
* אם לא קיים דוד, תתבצע רוטציה, האבא יהפך לשחור אך הילד ישאר אדום.

לכן ניתן לשים לב כי בכל מקרה, הילד ישאר אדום.

<u>סעיף ג</u>

<u>הערה: לפי מה שלמדנו, צמתים null נחשבים כשחורים, אך הגובה שלהם הוא 1- ולכן אינם נחשבים בחישוב הגובה ואין להם השפעה על התשובה.</u>

לא נכון. דוגמא נגדית:



נשים לב כי העץ הוא עץ אדום שחור תקין מכיוון שמקיים את כל התכונות אך לצומת השחור (היחיד בעץ זה) שהוא השורש אין לו אף לא צאצא אחד.

איתמר קרייטמן ת.ז 208925578

נשים לב כי גובה השורש הוא 1, ולכן צריך להיות $1 = \left\lceil \frac{1}{2} \right\rceil$ על פי הטענה אך לא קיים כלל צאצא שחור.

<u>סעיף ד:</u>
נכון. יהי A שורש של עץ AVL, נניח בשלילה כי תת העץ השמאלי של A אינו מקיים את כל תכונות עץ AVL, נובע מכך כי תת העץ השמאלי אינו עץ AVL, ולכן כל A אינו עץ AVL מכיוון שאינו מקיים את תכונות עץ AVL, וזוהי סתירה כי נתון שA הוא שורש של עץ AVL.

<u>סעיף ה:</u>
לא נכון. נוכל להגיע על ידי הוצאה והכנסה של צמתים לעץ אדום שחור שכולו צמתים שחורים, וכמו כן עץ כזה שכל צמתיו שחורים מקיים את כל התכונות של עץ אדום שחור: השורש שחור, לכל צומת אדום יש אך ורק בנים שחורים, ובכל מסלול שיש לו ילד אחד או שהוא עלה יש אותו מספר של צמתים שחורים.

איתמר קרייטמן ת.ז 208925578

<u>שאלה 3:</u>

<u>שאלה 3:</u>

א. הוכיחו שהדפסת in-order על עץ חיפוש בינארי ידפיס את אברי העץ בסדר ממוין.

ב. נתון עץ בינארי T, ונתון שהדפסת in-order על T מדפיסה את איבריו בסדר ממוין. הוכיחו ש-T הוא עץ חיפוש בינארי.

ג. האם עץ מלא הוא תמיד מאוזן? נמקו את תשובתכם.

ד. האם עץ שלם הוא תמיד מאוזן? נמקו את תשובתכם.

<u>סעיף א:</u>

נוכיח באינדוקציה על K כאשר K הוא גובה העץ.

<u>בסיס:</u> עבור $K=0$ יש רק צומת אחד ולכן הדפסת in-order תדפיס בצורה ממוינת.

<u>צעד:</u> נניח כי הטענה מתקיימת עבור כל K מ1 ועד $K-1$, כלומר הדפסת in-order מדפיסה בצורה ממויינת ונוכיח עבור עץ בגובה K.

על מנת להגיע לעץ בגובה K נוסיף צומת אחד, שיהיה בן לאחד העלים, נניח בלי הגבלת הכלליות כי הערך של הצומת קטן מהשורש ולכן הצומת החדש יוצב כבן של אחד העלים בתת העץ השמאלי. כעת, הדפסת in-order תדפיס את הבן השמאלי, לאחר מכן את האבא ולאחר מכן את הבן הימני. וזה מתקיים לפי הנחת האינדוקציה, לכן כאשר יגיע תורו של הצומת החדש להיות מודפס, אם הוא בן שמאלי, הוא קטן מאביו ולכן יודפס לפניו, ואם הוא בן ימני הוא גדול מאביו ויודפס אחריו. אך בכל מקרה, לפני סבו ולפני כל אב קדמון שלו, וכל זאת על פי הנחת האינדוקציה.

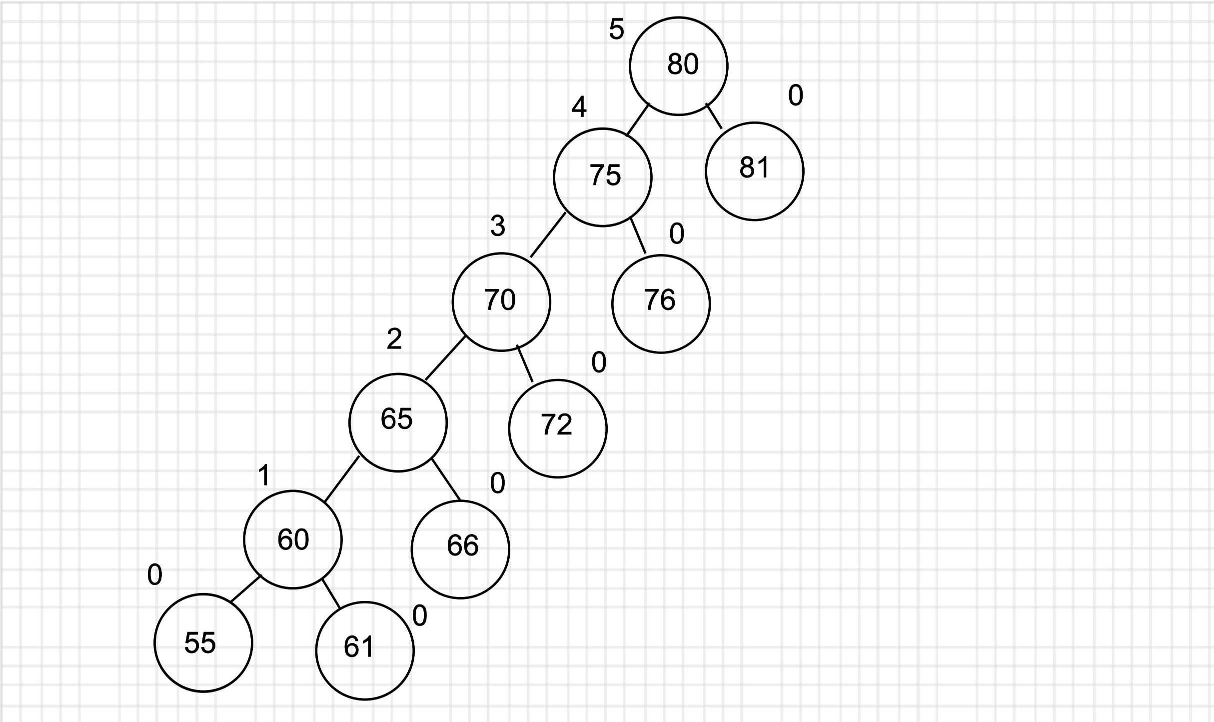
מש"ל.

<u>סעיף ב:</u>

נניח בשלילה כי T אינו BST, על כן בתת העץ השמאלי ישנם איברים שגדולים מהשורש, וכן בתת העץ הימני איברים שקטנים מהשורש. לכן הדפסת in-order לא תדפיס את העץ בסדר ממוין מכיוון שיודפסו איברים שגדולים מהשורש (או מכל צומת אחר) **לפני** השורש עצמו, וכן יודפסו איברים שקטנים מהשורש (או מכל צומת אחר) **אחרי** השורש (או הצומת). אבל נתון שin-order מדפיסה בסדר ממויין, וזוהי סתירה. על כן בהכרח T הוא BST.

איתמר קרייטמן ת.ז 208925578

סעיף ג:



נשים לב כי מספר הצמתים הוא $n = 11$, $log\ 11 = 3.5$.
נשים לב כי גובה העץ הוא 5, לכן גובה העץ הוא יותר מ$log\ n$ רמות, ולכן העץ אינו מאוזן. לכן הטענה אינה נכונה.

סעיף ד:

נכון. על פי תכונת עץ בינארי שלם- כל רמה, חוץ מהרמה האחרונה, תמיד מלאה, והעלים ברמה האחרונה מיושרים כמה שיותר לצד שמאל. יהי n מספר הצמתים בעץ לא כולל מספר הצמתים ברמה האחרונה ויהי m מספר העלים ברמה האחרונה ויהי k גובה העץ לפי הרמה האחת לפני האחרונה. אם הרמה האחרונה מלאה, הרי שהעץ הוא עץ מלא והיותו מאוזן הוכחה בסעיף קודם. אחרת, ניתן לומר כי מספר הצמתי בעץ הוא:

$$n + m = \sum_{i=0}^{k} 2^i + m = 2^k + m$$

$$\implies n = 2^k$$

איתמר קרייטמן ת.ז 208925578

$$\implies n = 2^k$$
$$\implies log(n) = k = O(logn)$$

מכיוון שהוכחנו כי גובה העץ הוא מסדר גודל $O(logn)$ ניתן לומר כי אכן עץ שלם הוא עץ מאוזן.

איתמר קרייטמן ת.ז 208925578

<u>שאלה 4:</u>

<u>סעיף א:</u>

```java
//Q4.B-Print BT in level order. Complexity O(n)
public static void levelOrderPrint(Node root) {
    //base case- root is null
    if (root == null)
        return;
    int numberOfElementAtLevel;
    Queue<Node> temp = new LinkedList<>();
    temp.add(root);
    while (!temp.isEmpty()) {
        numberOfElementAtLevel = temp.size();//equal to number of elements in each level
        while (numberOfElementAtLevel > 0) {
            //iterating over the tree, inserting every element to end of the que and print it when it's first.
            Node top = temp.poll();
            System.out.print(top.getData() + " ");
            if (top.getLeft() != null)
                temp.add(top.getLeft());
            if (top.getRight() != null)
                temp.add(top.getRight());
            numberOfElementAtLevel--;
        }
    }
}
```

איתמר קרייטמן ת.ז 208925578

<u>סעיף ב:</u>

```java
//    Q4.B- Print BT in reverse level order. complexity O(n)
public static void reverseLevelOrder(Node root) {
    //base case- root is null
    if (root == null)
        return;
    Queue<Node> que = new LinkedList<>();
    Stack<Node> stack = new Stack<>();
    int numberOfElementAtLevel;
    //Inserting all Nodes to que in level order.O(n)
    que.add(root);
    while (!que.isEmpty()) {
        numberOfElementAtLevel = que.size();//equal to number of elements in each level
        while (numberOfElementAtLevel > 0) {
            Node top = que.poll();
            stack.push(top);
            if (top.getRight() != null)
                que.add(top.getRight());
            if (top.getLeft() != null)
                que.add(top.getLeft());
            numberOfElementAtLevel--;
        }
    }
    //pushing all elements in que to stack in order to reverse its order
    while (!que.isEmpty()) {
        Node top = que.poll();
        stack.push(top);
    }
    //print every element, should be printed in reverse level order
    while (!stack.isEmpty()) System.out.print(stack.pop().getData() + " ");
}
```