חומרים שהוכנו על-ידי משתתפי קורס מורים מובילים תשע"א

ניתן להשתמש בחומרים לצורך הוראה בלבד.

לא ניתן לפרסם את החומרים או לעשות בהם כל שימוש מסחרי

ללא קבלת אישור מראש מצוות הפיתוח

תבניות

מותאם לסביבות JAVA ,C#

כתיבה ועריכה:

רחל לודמר

זיוה קונצמן

שירלי רוזנברג-כהן

תבניות לפעולות בעצים בינאריים

המדריך הנוכחי יכול לשמש המשך לספר הלימוד. לא נידונו בו נושאים ושאלות המופיעות בספר, מאידך הדוגמאות כאן ברובן אינן מקוריות ונלקחו מחומרים מפורסמים של מורים שונים –

ועל כך תודה.

עצים כמבני נתונים נחשבים חומר קשה להוראה ולימוד בקרב חלק גדול מהמורים והתלמידים.
נראה לנו כי ניתן לרכך את ההוראה על-ידי שימוש בתבניות נפוצות בעצים וכמה טקטיקות מקובלות.

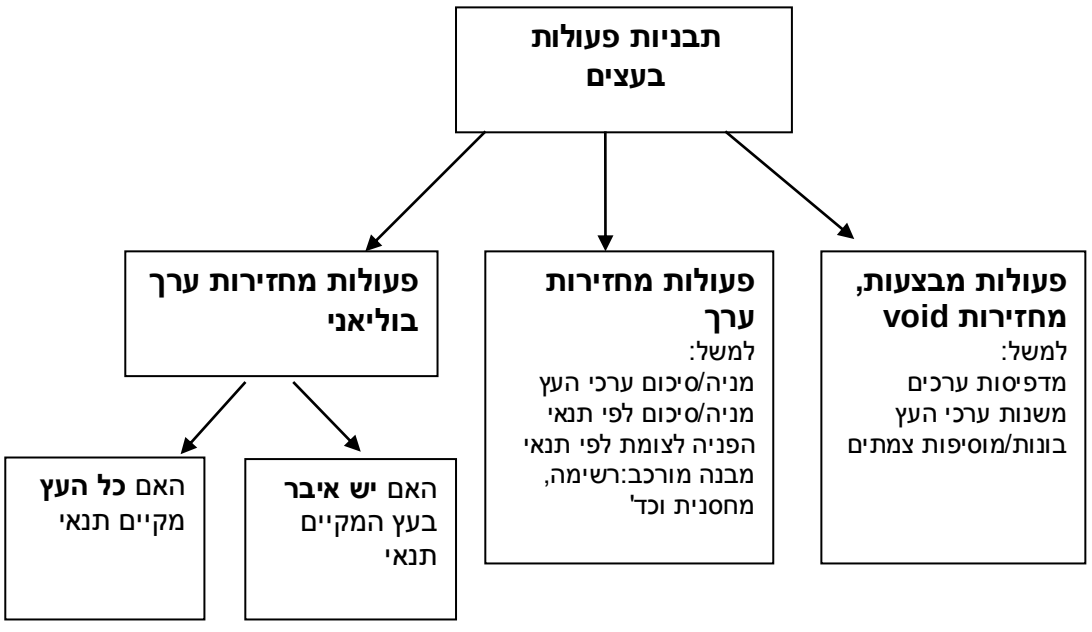
מכיון שרוב האלגוריתמים הם רקורסיביים, גם המבנה הכללי הוא תבנית רקורסיבית.
המבנה הכללי – לכל פעולה 3 חלקים:

- תנאי עצירה
- טיפול בצומת
- קריאות רקורסיביות – המשך המעבר מהצומת לצאצאים שלה.

נדון גם בטקטיקות נוספות:

פעולה לא רקורסיבית על צומת בודד
שימוש בפעולות עזר
עבודה (ועצירה) דרך צומת האב
פעולה עוטפת

חלוקת העצים לתבניות בסיסיות:



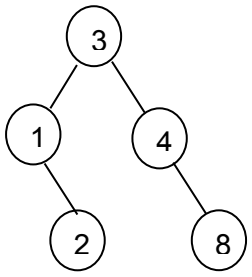
פעולות מבצעות :

המאפיין – פעולות שאינן מחזירות דבר. פעולות שעוברות על העץ ומטפלות בצמתים, ז.א. מעדכנות משהו בצמתי העץ, או רק בצמתים מסויימים, מדפיסות , מוסיפות משהו וכו'.

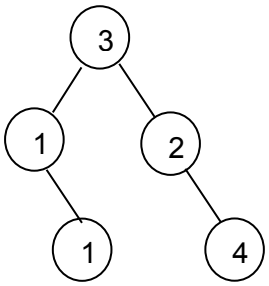
דוגמה 1 :

פעולה המקבלת עץ בינארי ומעדכנת את ערכי הצמתים הזוגיים לחצי מערכם.

לפני :



אחרי :



התבנית:

- **תנאי עצירה** : אם אין צומת – אין לעשות דבר. ומכיוון שזו פעולה שמחזירה void – נכנסים לפעולה רק אם יש צומת, כלומראם t != null. הפעולה תעצור כאשר t = null.
- **טיפול בצומת:** אם הצומת מקיים את התנאי– מעדכנים, מדפיסים וכו' לפי המשימה בשאלה.
- **רקורסיה שמאלה**
- **רקורסיה ימינה**

הפתרון:

```
public static void Update(BinNode<int> t)
{
    if (t != null)
    {
        if (t.GetValue() % 2 == 0)
            t.SetValue(t.GetValue() /2);
        Update(t.GetLeft());
        Update(t.GetRight());
    }
}
```

דוגמה 2:

פעולה המקבלת עץ בינארי ומוסיפה אח לכל צומת שהוא בן יחיד. ערך האח יהיה 0.

התבנית:

**** הפעולה עובדת דרך האב. משתמשת בפעולת עזר .***

- **תנאי עצירה**: אם האב הוא **עלה** אין צורך לעשות דבר.
- **טיפול בצומת:** אם לצומת רק בן ימני – מוסיפים בן שמאלי, אם לצומת רק בן שמאלי – מוסיפים בן ימני. (אם יש לו שני בנים לא עושים דבר – הרקורסיה ממשיכה). אין צורך לטפל בהגעה לnull כיוון שבמצב של בן יחיד מתווסף צומת שהוא עלה ואז הוא ייעצר בתנאי העצירה – עלה.
- **רקורסיה שמאלה**
- **רקורסיה ימינה**

פעולת עזר שימושית במיוחד:

פעולה המקבלת צומת ומחזירה קוד המציין מספר הבנים: 0 – אם הוא עלה, 1 – אם יש רק בן שמאלי, 2 – אם יש רק בן ימני, 3 – אם יש שני בנים.

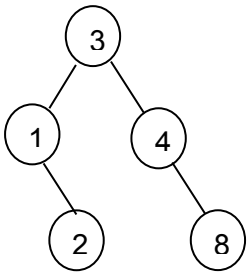
שים ♥: הפעולה אינה רקורסיבית, עובדת על צומת בודד.

```
public static int ChildKod (BinNode<int> t)
{
    if (t.GetLeft() == t.GetRight() )          // מכיון שמדובר בהפניות,
        return 0;                              // הערכים זהים רק כאשר שניהם null

    if (t.GetLeft() != null && t.GetRight != null )  return 3;

    if (t.GetLeft() !=  null )        return 1;

    return 2;
}
```

פתרון:

```
public static void AddBro(BinNode<int> t )
{
    int ck = ChildKod(t);

    if (ck != 0 )
    {
        if ( ck == 1)
            t.SetLeft(new BinNode<int> 0));
        if (ck == 2)
            t.SetRight(new BinNode<int> (0));

        AddBro(t.GetLeft());
        AddBro(t.GetRight());

    }

}
```

פעולות מחזירות ערך:



דוגמה 3:

פעולה המחזירה את **מספר** הצמתים בעץ שיש להם תוכן זוגי.

התבנית:

- **תנאי עצירה**: – אם t = null - החזר 0. אין צומת, אין מה לבדוק ובוודאי שאין מה לספור.
- **טיפול בצומת**: אם ערך הצומת זוגי (מקיים את תנאי השאלה) – החזר
  1+ רקורסיה שמאלה + רקורסיה ימינה. (סופרים את הצומת).
- אם הצומת לא מקיים את התנאי – לא סופרים אותו , אך ממשיכים לצאצאים. ז.א.
  החזר **רקורסיה שמאלה + רקורסיה ימינה**.

הפתרון:

```
public static int NumInfoZugi(BinNode<int> t)
{
    if (t == null)   return 0;
    if (t.GetValue() % 2 == 0)
        return 1 + NumInfoZugi(t.GetLeft()) + NumInfoZugi(t.GetRight());   // הוספת 1 לסכום
    return NumInfoZugi(t.GetLeft()) + NumInfoZugi(t.GetRight());
}
```

דוגמה 4:

הפעולה מקבלת עץ בינארי ומחזירה את **סכום** ערכי הצמתים שהם בנים ימניים.

התבנית:

**** הפעולה עובדת דרך האב.***

- **תנאי עצירה**: – אם t=null - החזר 0. אין צומת, אין מה לבדוק ובוודאי שאין מה לסכם.
- **טיפול בצומת**: אם יש לצומת בן ימני – החזר

  ערך הבן הימני+ רקורסיה שמאלה + רקורסיה ימינה.
- אם אין בן ימני החזר **רקורסיה שמאלה.**

פתרון :

```
public static int SumRightChild(BinNode<int> t)
{
    if (t == null ) return 0;
    if (t.GetRight() != null )
        return  t.GetRight().GetValue() + SumRightChild(t.GetLeft()) +
                                          SumRightChild(t.GetRight);
    return SumRightChild (t.GetLeft());
}
```

דוגמה 5:

פעולה המקבלת עץ בינארי ומספר שהוא ערך צומת בעץ, ומחזירה **הפניה** לאביו בעץ. אם המספר אינו נמצא בעץ – יוחזר null

הנחה : הערכים בעץ שונים זה מזה.

תבנית:

- **תנאי עצירה** – הגענו ל null – פירושו שהמספר לא נמצא בשלב זה – יוחזר null
- אם הגענו לצומת שאחד הבנים שלה ערכו x , תוחזר ההפניה לצומת.
- נחפש את צומת האב בענף השמאלי. אם נמצא שם – תוחזר ההפניה לאב. אם לא נמצא – נחפש בענף הימני והתוצאה שלו תוחזר.

```
public static BinNode<int> Aba(BinNode<int> bt,  int  x)
{
    if (bt == null)
        return null;
    if (bt.GetRight()  !=  null && bt.GetRight()== x)
        return bt;
    if (bt.GetLeft()  !=  null && bt.GetLeft()== x)
        return bt;
    BinNode<int> temp = Aba(bt.GetLeft(), x)
    if (temp == null)
        return Aba(bt.GetRight(), x);
    return temp;
}
```

פעולות המחזירות ערך בוליאני:

"האם יש"

דוגמה 6:
פעולה המקבלת עץ ותו ומחזירה 'אמת' אם התו נמצא בעץ ו'שקר' אחרת.

התבנית:

- **תנאי עצירה** – אם t == null - החזר **'שקר'**. אם הגענו ל null סימן שלא מצאנו עדיין את מה שחיפשנו, כי אם היינו מוצאים כבר היינו מפסיקים לפני ה- null ומחזירים 'אמת'
- **טיפול בצומת** – אם הצומת מקיים את התנאי של השאלה – החזר 'אמת', מצאנו.
  אם הצומת אינו מקיים את התנאי של השאלה – נמשיך לחפש באחד הצאצאים שלה – לא חובה שנמצא בשניהם.
- החזר רקורסיה שמאלה **או** רקורסיה ימינה.

הפתרון:

```
public static bool IsInTree(BinNode<char> t, char  x)
{
    if (t == null)
        return false;
    if (t.GetValue() == x)
        return true;
    return IsInTree(t.GetLeft(), x)  ||  IsInTree(t.GetRight(), x);
}
```

"האם כל"

דוגמה 7:
פעולה המחזירה 'אמת' אם **כל** צמתי העץ תוכנם זוגי ו'שקר' אחרת.

התבנית:

- **תנאי עצירה** - אם t == null - החזר 'אמת' . אם הגענו ל null פירושו שלא "עפנו" בדרך עם false, על ידיע צומת שלא מקיים את התנאי.
- **טיפול בצומת** – אם הצומת לא מקיים את התנאי– החזר 'שקר'. אין מה להמשיך בעץ, מספיק שצומת אחת מפר את התנאים .
- אם הצומת מקיים את התנאי, עדיין צריך לבדוק את צאצאיה –
  החזר רקורסיה שמאלה **וגם** רקורסיה ימינה.

הפתרון:

```
public static bool AllZugi(BinNode<int> t)
{
    if (t == null)
        return true;
    if (t.GetValue() % 2  !=  0)
        return false;
    return AllZugi(t.GetLeft())  &&  AllZugi(t.GetRight());
}
```

השימוש בתבניות בשאלות מורכבות:

דוגמה 8:

"עץ חילקיהו" הוא :

עץ עלה או

צומת בעל שני בנים, שערכו של הבן השמאלי מתחלק ללא שארית בערכו של הבן הימני ומנת החלוקה שלהם שווה לערכו של אביהם , וכל אחד מבניו הוא "עץ חילקיהו".

כתוב פעולה המקבלת עץ בינארי ומחזירה 'אמת' אם הוא עץ חילקיהו, אחרת תחזיר 'שקר'.

ניתוח הפתרון:

פעולה בוליאנית.

"עץ עלה" – עלה יחזיר אמת.

"צומת בעל שני בנים" - צומת שהוא אב לבן יחיד יחזיר 'שקר', ללא בנים = עלה, כבר קודם יחזיר 'אמת'.

בדיקת הערכים – העבודה תעשה דרך האב.

תבנית: "האם כל".

שימוש בפעולת עזר: קוד מספר הבנים.

תבנית:

- **תנאי עצירה** – אם עלה יחזיר 'אמת'.
- **טיפול בצומת** – אם קיים בן יחיד – יחזיר שקר.
- אם ערכי הצומת ובניו אינם עונים על התנאים – נחזיר שקר.
- רקורסיה שמאלה **וגם** רקורסיה ימינה.

פתרון:

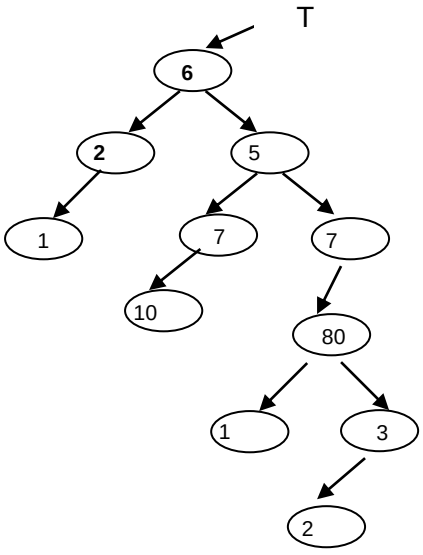
```
public static bool Hilkiyahoo (BinNode<int> t)
{
    int sons = ChildKod(t);
    if (sons == 0)   return true;     // עלה
    if (sons <3)   return false;      //בן יחיד
    if  (t.GetLeft().GetValue() %  t.GetRight().GetValue()  !=  0    ||
        t.GetLeft().GetValue() / t.GetRight().GetValue()  !=  t.GetValue())
            return false;
    return Hilkiyahoo (t.GetLeft() && Hilkiyahoo (t.GetRight());
}
```

דוגמה 9:

כתוב פעולה המקבלת עץ בינארי ומחזירה את **"סכום השרשרת המקסימלי"**.

שרשרת היא מסלול מהשורש עד עלה. הפעולה תחזיר את הסכום של השרשרת המקסימלית.

דוגמא:



אם T הוא העץ הנתון, אז יוחזר 31

2+3+8+7+5+6 = 31 (לעומת 28=6+5+7+10)

ועבור העץ

יוחזר 8.

ניתוח הפתרון:

"המקרה הקטן ביותר" – צומת בודד – נחזיר את ערך הצומת.

לא ניתן לקרוא רקורסיבית ימינה ושמאלה באופן סימטרי כי יש לבחור מסלול ללכת בו – או ימינה או שמאלה.

נבחר – אם יש רק בן אחד נמשיך אליו.

אם יש שני בנים נבחר בערך של הגדול מביניהם.

אם ערכי הבנים זהים נבחר בתת-העץ שנותן את הסכום הסופי הגדול יותר.

עבודה דרך האב כי צריך לסכם אותו בכל שלב.

נעזר בפעולת העזר childKod(t)

פתרון:

```
public static int MaxSharsheret(BinNode<int> bt)
{
    int k = ChildKod(bt);
    if (k == 0)
        return bt.GetValue();          //      יש לו רק בן שמאלי
    if (k == 1)
        return bt.GetValue() + MaxSharsheret(bt.GetLeft());        //      יש לו רק בן ימני
    if (k == 2 )
        return bt.GetValue() + MaxSharsheret(bt.GetRight());       //      יש לו שני בנים
    return   bt.GetValue() + Math.Max(MaxSharsheret(bt.GetLeft()),
                                      MaxSharsheret(bt.GetRight()));
}
```

דוגמה 10:

פעולה המקבלת עץ בינארי של תווים ,ומחזירה רשימה של עצמים מטיפוס Pair.

אין חשיבות לסדר התווים ברשימה, אך כל תו לא יופיע יותר מפעם אחת.

Pair	
private char tav // תו המופיע בעץ private int num // מספר הופעותיו בעץ	תכונות
הנח קיום פעולות בונות ומאחזרות	פעולות

ניתוח הפתרון:

נצטרך מחלקה עבור העצמים שהם איברי הרשימה.

נצטרך פעולה עוטפת – בה תאותחל הרשימה.

נצטרך פעולה הבודקת שהאיבר (התו) אינו מופיע כבר ברשימה. **ExistInLis**t

פעולה הסופרת את מס ההופעות של התו x בעץ. (לפי דוגמה 4 כאן) **NumTav**

פתרון :

```
public static bool ExistInList( Node<Pair> lst, char tav)
{
	while(lst != null)
	{
		if (lst.GetValue().GetTav() == tav)
			return true;
		lst = lst.GetNext();
	}
	return false;
}


public static void Update(BinNode<int> t, Node<Pair> lst)
{
	if (t != null)
	{
		if (! ExistInList (lst, t.GetValue())
		{
			int num= NumTav (t, t.GetValue());		//	פעולת עזר בדוגמה 4
			lst = new Node<Pair>(new Pair (t.GetValue(), num), lst);
		}
		Update (t.GetLeft(), lst);
		Update (t.GetRight(), lst);
	}
}


//	פעולה עוטפת
public static Node<Pair> Update(BinNode<int> t)
{
	Node<Pair> lst = null;
	Update(t, lst);
	return lst;
}
```

ברעיון דומה אפשר ליצור את השאלה הבאה:

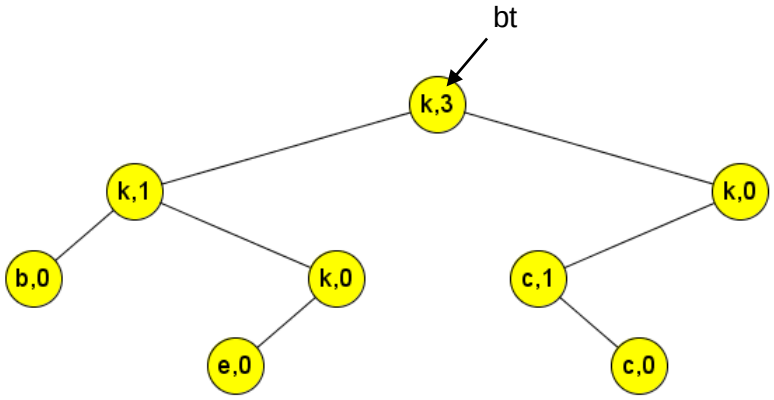
דוגמה 11:

עץ בינארי שאיבריו מטיפוס Pair, ייקרא "תו_ועוד" , אם **בכל אחד מהצמתים** שלו מתקיים:

הערך המספרי num שבצומת, הוא מספר ההופעות של האות tav בכל הצאצאים שלו, וגם האות tav שבצומת גדולה או שווה מערכי האותיות של הצאצאים שלו.

כתוב פעולה המקבלת עץ בינרי bt מטיפוס InfoTree, ומחזירה 'אמת' אם הוא מסוג "תו_ועוד" ו'שקר' אחרת.

דוגמא לעץ "תו_ועוד":



ניתוח הפתרון:

כל צומת בעץ צריך לקיים את התנאים . לכן נשתמש בתבנית "האם כל".

כדי לבדוק את קיום התנאים נצטרך לפרק לשני תנאים שונים. כל אחד מהם יהווה פעולת עזר רקורסיבית לפי התבניות הבוליאניות:

numTav – המונה כמה פעמים מופיע התו. נקרא לפעולה כל פעם מתת-עץ אחר.

checkBig – תחזיר אמת אם התו הוא הגדול בתת-העץ שלו.

פתרון ב- java:

```
//      מחזיר אמת אם התו הוא הגדול ביותר בעץ
public static bool CheckBig(BinNode<InfoTree> bt, char tav)
{
      if (bt == null)  return true;            //      תנאי עצירה – הכל בסדר

      if (tav  <  bt.GetValue().GetTav())
            return false;

      return CheckBig (bt.GetLeft(),tav)   &&   CheckBig (bt.GetRight(), tav);
}
```

```
public static bool TavPlus (BinNode<InfoTree> bt)
{
    bool ok1, ok2;
    if (bt == null)   return true;          //      תנאי עצירה
        char tav = bt.GetValue().GetTav();
    ok1 = bt.GetValue().GetNum() == NumTav(bt.GetLeft(),tav) +
                                    NumTav(bt.GetRight(),tav);
    ok2 = CheckBig( bt, tav);
    if (!ok1 || !ok2)   return false;
    return  TavPlus(bt.GetLeft())   &&  TavPlus(bt.GetRight());
}
```

<u>דוגמה 12: סבא רבא</u> (בגרות תש"ס)

"סבא-רבא" של שני **עלים** שונים בעץ בינרי הוא האב הקדמון המשותף לשני העלים ברמה הגבוהה ביותר בעץ.

כתוב פעולה המחזירה את הצומת שהוא "סבא-רבא" של העלים ale1, ale2 בעץ.

(ציטוט חלקי)

<u>ניתוח הפתרון :</u>

לפי המוצע בשאלה מומלץ להשתמש בשתי פעולות עזר:

הורה – aba (דוגמה 5 כאן)

האם צאצא - המחזירה 'אמת' אם x הוא צאצא של y בעץ ו'שקר' אחרת.

לצורך הפעולה נשתמש בפעולה המקבלת עץ בינארי t והפניה לצומת bt. הפעולה מחזירה 'אמת' אם הצומת bt נמצא בעץ t. לפי תבנית "האם יש".

נמצא את x ואחר כך נבדוק אם y מופיע בתת-העץ מתחתיו.

```
public static bool  IsIn (BinNode<int> t, BinNode<int> bt)
{
    if (t == null ) return false;          //      יצאנו ולא מצאנו
    if (t == bt ) return true;             //      הפניות זהות – מצאנו
    return IsIn(t, t.GetLeft()) ||  IsIn(t, t.GetRight());
}
```

```
public static bool IsZeeza(BinNode<int> bt, BinNode<int> x, BinNode<int> y)
{
    if (bt == null)    return false;
    if (bt == x)
        if (IsIn(bt.GetLeft(), y) ||  IsIn(bt.GetRight(), y))
            return true;
    return IsZeeza(bt.GetLeft(), x, y) ||  IsZeeza(bt.GetRight(), x, y);
}
```

הרעיון הלא שגרתי כאן הוא , שתנאי העצירה הוא הפוך בכיוון – לא נלך כלפי מטה, אלא דווקא להיפך – נעלה כל פעם לאבא של אחד העלים ונברר אם העלה השני הוא צאצא שלו. מכאן שתנאי העצירה יהיה, כאשר נגיע לשורש העץ, שהרי כל הצמתים הם צאצאיו. ההליכה מלמטה למעלה מחוייבת כאן, כיוון שאנחנו מחפשים את הצומת שהיא הסבא רבא ברמה הגבוהה ביותר, ז.א. הצומת הראשונה המקיימת את התנאי .

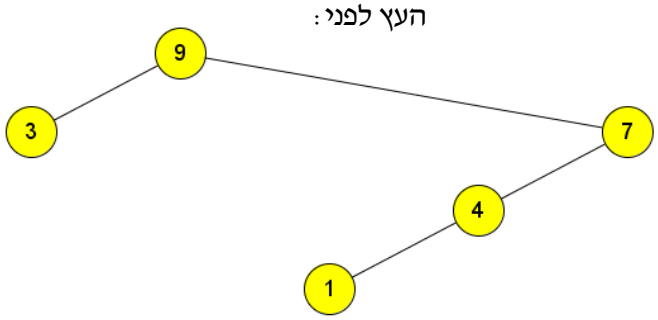
כאן נשתמש בפעולת עזר Aba המחזירה את ההורה של הצומת בו אנחנו מטפלים, תוך בירור אם הצומת השנ היא צאצא של האב, ואז זוהי התשובה.
צאצאיו. מתאים לתבנית השלישית והשוני היחיד הוא במיוחדות של תנאי העצירה.

```
public static BinNode<int> SabaRaba ( BinNode<int> t,
                                      BinNode<int> leaf1, BinNode<int> leaf2)
{
    BinNode<int> father  = Aba(t,leaf1);
    if (father == t) return t;
    if  (IsZeeza(t, leaf2, father)  return father;
    return SabaRaba(t, father, leaf2);
}
```

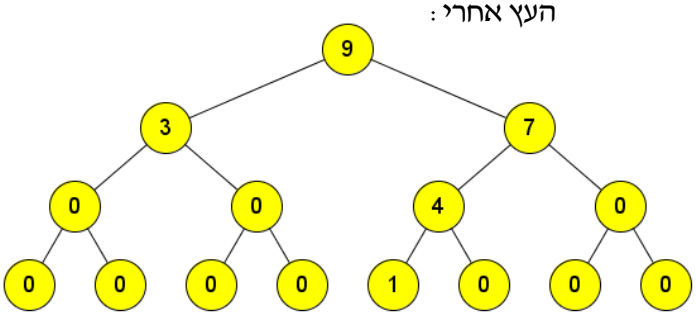
דוגמא 13:

פעולה המקבלת עץ בינארי של מספרים, ומוסיפה בו צמתים עד שיהיה עץ מאוזן מלא. ערכו של כל צומת שנוסף יהיה 0.

העץ לפני:



העץ אחרי:



נתוח הפתרון:

ניצור פעולה עוטפת שבה:

נחשב את גובה העץ.

נקרא לפעולה רקורסיבית שבה:

ניכנס עם רמה 0 ונוסיף 1 כל פעם שנרד בעץ.

נוסיף צומת עם הערך 0 כשנפגוש עלה או בן יחיד.

```
//      פעולה עוטפת
public static void Meuzan((BinNode<int> t)
{
      int h = HightTree(t);          //      פעולת עזר לחישוב גובה העץ
      HelpMeuzan(t,h,0);
}


public static void HelpMeuzan(BinNode<int> t, int h, int rama)
{
      if (rama < h)
      {
            if (t.GetLeft() == null)
                  t.SetLeft(new BinNode<int>(0));
            if (t.GetRight() == null)
                  t.SetRight(new BinNode<int>(0));
            HelpMeuzan(t.GetLeft(), h, rama+1);
            HelpMeuzan(t.GetRight(), h, rama+1);
      }
}
```

דוגמה 14: בגרות תשס"א

א. לפניך כותרת הפעולה: public static void Leaves(BinNode<int> t, Stack<int> s)

הפעולה מקבלת עץ בינרי לא ריק t של מספרים שלמים ומחסנית ריקה s של מספרים שלמים.

הפעולה מכניסה למחסנית את ערכי **כל** העלים של העץ t , על פי סדר סריקה מימין לשמאל.

הפעולה היא פעולת void שסורקת את העץ ומטפלת רק בעלים, המחסנית מופיעה בפרמטר – לכן מתאימה **לתבנית הראשונה** של הסריקה.

בטיפול בצומת מבררים אם הצומת הוא עלה, אם כן, מכניסים אותו למחסנית.

לפי בקשת השאלה ברקורסיה פונים קודם לתת העץ הימני לפני השמאלי, וכך מקבלים את העלים בסדר מימין לשמאל ולא משמאל לימין, כפי שזה מופיע בתבנית.

```
public static void Leaves(BinNode<int> t, Stack<int> s)
{
    if (t != null)
    {
        if (Ale(t))
            s.Push(t.GetValue());
        Leaves(t.GetRight(), s);
        Leaves(t.GetLeft(), s);
    }
}
```

המשך השאלה:

ב. כתוב פעולה בוליאנית שתקבל 2 עצים בינריים לא ריקים של מספרים שלמים, ותחזיר true אם מתקיימים שני התנאים האלה:
- יש להם אותו מספר עלים
- על פי סדר הסריקה מימין לשמאל, ערכי העלים שווים

אחרת, הפעולה תחזיר false.

נכתוב פעולה היוצרת שתי מחסניות בעזרת הפעולה מסעיף א'. נחזיר 'אמת' אם ערכיהן זהים והן מסתיימות יחד.

```
public static bool IsEqual(BinNode<int> t1, BinNode<int> t2)
{
    Stack<int> s1 = new Stack<int> ();
    Stack<int> s2 = new Stack<int> ();

    Leaves(t1, s1);
    Leaves(t1, s1);

    while (!s1.IsEmpty()  &&  !s2.IsEmpty())
    {
        if (s1.Pop() != s2.Pop())
            return false;
    }

    return s1.IsEmpty() && s2.IsEmpty();

}
```