סיכום נושאים וחזרה למבחנים

- **רקורסיה**

  מעקב רקורסיה
  רקורסיה כפולה
  רקורסיה ברשימה
  רקורסיה במחסנית/ תור

- **עצמים ומחלקות**
  אוספים במערך
  אוספים ברשימה
  אוספים במחסנית / בתור
  פעולות על אוספים.

- **מערך** של עצמים
  מערך של רשימות
  מערך של תורים / מחסניות

- **סריקות**
  סריקת מערך
  סריקת מערך באוסף

  סריקת רשימות - עד נקודה ידועה
  סריקת מחסנית - עד נקודה ידועה, מחסנית עזר
  סריקת תור - עד הסוף, תור עזר
  סריקת עץ - סדר תחילי / תוכי / תחילי

- **תבניות אלגוריתמיות:**
  אורך רשימה, מחסנית, תור

  בניית רשימה מההתחלה, מהסוף (דף: בנייה והדפסה של שרשרת)
  הכנסה בצורה ממוינת לרשימה, למחסנית ולתור
  תת-רשימה עולה (דף: תת סדרה עולה)
  נמצא ברשימה, במחסנית, בתור, בעץ
  הוצאה מרשימה

- צמצום רשימה
  צמצום מחסנית (שכפול מחסנית)
  צמצום תור (שכפול התור באמצעות תור עזר או באמצעות תור מעגלי)

- **עצים**
  תבניות אלגוריתמיות (דפים: תבניות אלגוריתמיות, אוסף פעולות , תבניות בעצים)
  סריקת עץ לפי רמות
  סכום רמות בעץ
  הורים וצאצאים בעץ

- **רקורסיה**

מעקב רקורסיה
רקורסיה כפולה
רקורסיה ברשימה
רקורסיה במחסנית/ תור

טבלת מעקב לרקורסיה:

עמודה לכל משתנה
עמודה לכל תנאי
מעקב על מערך - עמודה לכל תא במערך המשתתף בתנאים
מעקב על רשימה / מחסנית / תור - עמודה למצב מבנה הנתונים (מוצג ה- toString של המבנה)

רקורסיה כפולה:

עץ מעקב
אין צורך להראות חישובים (אלא אם כן הם נוגעים לעצם המעקב ואינם חישובים פשוטים).

שימוש בשני צבעים: האחד לזימון הרקורסיבי והאחר לחזרה מהרקורסיה.

- **עצמים ומחלקות**
  אוספים במערך
  אוספים ברשימה
  אוספים במחסנית / בתור
  פעולות על אוספים.

עצם בסיסי:

תכונות, בנאים, get / set לתכונות, toString
פעולות חישוביות במידת הצורך.

אוספים:

תכונות:

תכונות מזהות (בסיסיות): שם, מס' מזהה וכד'
אוסף במערך:
גודל המערך כתכונה סטטית, המערך, lastPosition .

אוסף ברשימה / שרשרת חוליות, במחסנית, בתור:
הפנייה למבנה הנתונים

פעולות:

פעולה בונה: מקבלת כפרמטר רק ערכים לתכונות הבסיסיות.
יוצרת אוסף ריק (אינה מקבלת אוסף)

פעולת הוספה: add מקבלת עצם מאותחל ורק מוסיפה אותו לאוסף.
במערך - עדכון lastPosition
בשרשרת חוליות - אם לא נאמר אחרת - בתחילת הרשימה.

האם האוסף ריק? isEmpty

פעולות איתור get - לפי תכונה של העצם במיוצג באוסף

פעולות מחיקה delete - מקבלת תכונה, מאתרת, מוציאה ומחזירה את הערך שהוצא.

toString

פעולות חישוב במידת הצורך.

אין פעולת Get לאוסף (למערך/לרשימה...) - התכנית לא מכירה את מבנה האוסף

אוסף מורכב: אוסף שאחת מתכונותיו מכילה אוסף

האוסף אינו מכיר את מבנה התכונה המכילה את האוסף.
המחלקה המגדירה את התכונה חייבת להכיל פעולות המספקות תשובות.
האוסף פונה לפעולות אלו.

דוגמאות:

- בגרות 2007 - מסעדה טעמים 22

- **מערך**
  מערך בסיסי
  מערך של עצמים
  מערך של רשימות
  מערך של תורים / מחסניות

הגדרת מערך:

מערך של טיפוס בסיסי, למשל מספרים שלמים (כל תא מכיל 0)

```
int [] arrInt = new int [n];
```

מערך של עצמים (כל תא מכיל null)

```
Book [] arrBook = new Book [n]
```

שאר איברי המערך מכילים null

```
arrBook [0] = new Book (פרמטרים של ספר);
```

שימו לב! ב- java **אין** מערך של טיפוסים גנריים. **יש** מערך של הפניות לטיפוסים גנריים.

מערך של רשימות:

מערך של הפניות ל**רשימות**, כל תא מכיל ערך null

```
Node<Integer> [] arrLst = new Node [n]
```

אתחול תא 2 עם חוליה שערכה 3

```
arrLst[2] new Node<Integer> (3);
```

מערך של מחסניות:

יצירת מערך של הפניות ל**מחסניות**

```
Stack <Double> [] arrStk = new Stack [5];
for (int i = 0 ; i < arrStk.length ; i++)
    arrStk [i] = new Stack<Double> ();
```

אתחול כל הפניה במערך למחסנית ריקה.
רק עכשיו אפשר לבצע פעולות על כל אחת מהמחסניות

מערך של תורים:

יצירת מערך של הפניות ל**תורים**

```
Queue <Character> [] arrQue = new Queue [5];
for (int i = 0 ; i < arrQue.length ; i++)
    arrQue [i] = new Queue <Character> ();
```

אתחול כל הפניה במערך לתור ריק.
רק עכשיו אפשר לבצע פעולות על כל אחד מהתורים

- **סריקות**

סריקת מערך
סריקת מערך באוסף

סריקת רשימות - עד הסוף או עד נקודה ידועה
סריקת מחסנית - עד הסוף או עד נקודה ידועה, מחסנית עזר
סריקת תור - עד הסוף, תור עזר

סריקת עץ - סדר תחילי / תוכי / תחילי

סריקת מערך:

for (**int** i = 0 ; i < arr.*length* ; i++) …

סריקת מערך באוסף:

for (**int** i = 0 ; i < **this**.*lastPosition* ; i++) …

סריקת רשימה / שרשרת חוליות:

pos מקבל הפניה לתחילת הרשימה

```
Node<Integer> pos = lst;
while (pos != null  &&  pos.getValue() < x)
{
      pos.getValue() טיפול במידע החוליה
      pos = pos.getNext();
}
```

אם x מטיפוס מחרוזת String,
ההשוואה נעשית ב- equals
או ב- compareTo

סריקת מחסנית:

אין לשלב stk.**pop**() בלולאה (אובדן איבר)

```
while (! stk.isEmpty ()  &&  stk.top() < x)
{
      int x = stk.pop()
      טיפול באיבר המחסנית
      שמירת x במחסנית העזר
}     לא לשכוח להחזיר את האיברים ממחסנית העזר בחזרה למחסנית
```

סריקת תור:

```
while (! que.isEmpty () )
{
      int y = que.remove()
      טיפול באיבר התור
      שמירת x בתור העזר
}           לא לשכוח להחזיר את האיברים מתור העזר בחזרה לתור המקורי
```

סריקת עץ:

PostOrder - סדר סופי	InOrder - סדר תוכי	PreOrder - סדר תחילי

static void doPostOrder
(BinNode<Integer> bt)
{
if (bt != **null**)
{
doPostOrder(bt.getLeft();
doPostOrder(bt.getRight();
bt.getValue() -טיפול ב
}
}
	static void doInOrder
	(BinNode<Integer> bt)
	{
	if (bt != **null**)
	{
	doInOrder(bt.getLeft();
	bt.getValue() -טיפול ב
	doInOrder(bt.getRight();
	}
	}
		static void doPreOrder
		(BinNode<Integer> bt)
		{
		if (bt != **null**)
		{
		bt.getValue() -טיפול ב
		doPreOrder(bt.getLeft();
		doPreOrder(bt.getRight();
		}
		}

סריקת המחסנית תוך שמירה על איברי המחסנית:
נועד למקרים שבהם ניתן לבצע את הפעולה במעבר אחד על המחסנית

```
Stack <Integer> sTemp = new Stack<Integer> ();
while (! stk.isEmpty())
{
      int x = stk.pop();
      :   טיפול באיבר שנשלף מהמחסנית
      sTemp.push (x);
}
while (! sTemp.isEmpty())
      stk.push (sTemp.pop());
```

שימו לב!
כשסורקים מחסנית, ניתן לעצור את הסריקה באמצע ולהחזיר את האיברים למחסנית (סדר האיברים המקורי נשמר).

סריקת התור תוך שמירה על איברי התור:
נועד למקרים שבהם ניתן לבצע את הפעולה במעבר אחד על התור

```
Queue <Integer> qTemp = new Queue <Integer> ();
while (! que.isEmpty())
{
      int x = que.remove();
      :   טיפול באיבר שנשלף מהתור
      qTemp.insert (x);
}
while (! sTemp.isEmpty())
      que. insert (qTemp. remove ());
```

שימו לב!
כשסורקים תור, אסור לעצור את הסריקה באמצע. חייבים להמשיך לרוקן את התור לתור העזר לפני שמחזירים את האיברים לתור (אחרת סדר האיברים המקורי לא יישמר ואיבר שהיה בראש התור עלול למצוא את עצמו בסופו).

מתי משכפלים מחסנית / תור?

רק במקרים בהם סריקת המחסנית / תור אינם שומרים על הסדר או מחייבים מחיקה.
למשל: צמצום איברי המחסנית / התור הכפולים לאיברים המכילים שתי תכונות: ערך המחסנית / התור ומספר מופעיו.

- **תבניות אלגוריתמיות:**
  אורך רשימה, מחסנית, תור

  בניית רשימה מההתחלה, מהסוף (דף: בנייה והדפסה של שרשרת)
  הכנסה בצורה ממוינת לרשימה, למחסנית ולתור
  תת-רשימה עולה (דף: תת סדרה עולה)
  נמצא ברשימה, במחסנית, בתור, בעץ
  הוצאה מרשימה

```
//--- פעולה המוסיפה איבר לרשימה ממוינת כך שהמיון נשמר ---
public static Node<Integer> insertSorted (Node<Integer> lst, int x)
{
    Node<Integer>pos = lst;
    Node<Integer>prev = null;

    //--- חיפוש המקום להכנסה ---
    while (pos != null && pos.getValue() < x)
    {
        prev = pos;
        pos = pos.getNext();
    }

    //--- הכנסה לשרשרת ---
    if (prev == null)        //  הכנסה בהתחלה
        lst = new Node<Integer> (x, lst);
    else                     //  prev הכנסה אחרי
        prev.setNext(new Node<Integer> (x, pos));

    return lst;
}
```

אם x מטיפוס מחרוזת String,
ההשוואה נעשית ב- equals
או ב- compareTo

```
//--- פעולה המוסיפה איבר למחסנית ממוינת כך שהמיון נשמר ---
public static void insertSorted (Stack<Integer> stk, int x)
{
    Stack<Integer> sTemp = new Stack<Integer>();

    //---  העברת האיברים הקטנים מ- x למחסנית העזר
    while (! stk.isEmpty() && stk.top() < x)
        sTemp.push(stk.pop());

    stk.push(x);             //  הכנסת x למקומו המתאים

    //--- החזרת האיברים למחסנית ---
    while (! sTemp.isEmpty())
        stk.push(sTemp.pop());
}
```

```
//--- פעולה המוסיפה איבר לתור ממויין כך שהמיון נשמר ---
public static void insertSorted (Queue<Integer> que, int x)
{
    Queue<Integer> qTemp = new Queue<Integer>();

    //---  העברת האיברים הקטנים מ- x לתור העזר
    while (! que.isEmpty() && que.head() < x)
        qTemp.insert(que.remove());
    qTemp.insert(x);              //  הכנסת x לתור העזר
    //--- העברת שאר האיברים לתור העזר ---
    while (! que.isEmpty())
        qTemp.insert(que.remove());

    //--- החזרת האיברים לתור ---
    while (! qTemp.isEmpty())
        que.insert(qTemp.remove());
}
```

נמצא ברשימה: למעשה אין הבדל גדול בין חיפוש ברשימה ממוינת וחיפוש ברשימה לא ממוינת.
שתי הפעולות דומות הן מבחינת פונקציית זמן ריצה והן מבחינת יעילות.
היתרון נמצא ברשימה ממוינת, אם האיבר לא נמצא ברשימה ייפסק החיפוש כשיימצא איבר ראשון גדול ממנו בשעה שבחיפוש ברשימה לא ממוינת יימשך החיפוש עד סוף הרשימה.

```
//=============== חיפוש - האם נמצא ========
//--- חיפוש האם נמצא ברשימה לא ממויינת ----
public static boolean isExist (Node<Integer> lst, int x)
{
    while (lst != null)
    {
        if (lst.getValue() == x)
            return true;
        lst = lst.getNext();
    }
    return false;
}

//--- חיפוש האם נמצא ברשימה ממויינת ----
public static boolean doesExist (Node<Integer> lst, int x)
{
    while (lst != null && lst.getValue() < x)
        lst = lst.getNext();

    if (lst != null && lst.getValue() == x)
        return true;
    return false;
}
```

אם x מטיפוס מחרוזת String,
ההשוואה נעשית ב- equals
או ב- compareTo

נמצא במחסנית? גם כאן ההבדל הוא כאשר המחסנית לא ממוינת והאיבר לא נמצא.

```
//--- חיפוש האם נמצא במחסנית לא ממויינת ----
public static boolean isExist (Stack<Integer> stk, int x)
{
    Stack <Integer> sTemp = new Stack<Integer>();
    boolean found = false;

    //--- חיפוש האיבר במחסנית תוך שמירת האיברים במחסנית עזר ---
    while (! stk.isEmpty()  &&  ! found)
        if (stk.top() == x)
            found = true;
        else
            sTemp.push(stk.pop());

    //--- החזרת האיברים למחסנית ---
    while (! sTemp.isEmpty())
        stk.push(sTemp.pop());

    return found;
}

//--- חיפוש האם נמצא במחסנית ממויינת ----
public static boolean doseExist (Stack<Integer> stk, int x)
{
    Stack <Integer> sTemp = new Stack<Integer>();
    boolean found = false;

    //--- חיפוש האיבר במחסנית תוך שמירתהאיברים במחסניתעזר ---
    while (! stk.isEmpty() && stk.top() < x)
        sTemp.push(stk.pop());

    //--- האם הלולאה נעצרה כי נמצא האיבר? ---
    if (! stk.isEmpty() && stk.top() == x)
        found = true;

    //--- החזרת האיברים למחסנית ---
    while (! sTemp.isEmpty())
        stk.push(sTemp.pop());

    return found;
}
```

נמצא במתור? שימוש בתור עזר

```
//--- חיפוש האם נמצא בתור  ----
public static boolean isExist (Queue<Integer> que, int x)
{
    Queue <Integer> qTemp = new Queue<Integer>();
    boolean found = false;

    //--- חיפוש האיבר בתור תוך שמירת האיברים בתור עזר ---
    while (! que.isEmpty())
    {
        if (que.head() == x)
            found = true;
        qTemp.insert(que.remove());
    }

    //--- החזרת האיברים לתור ---
    while (! qTemp.isEmpty())
        que.insert(qTemp.remove());

    return found;
}
```

שים לב שגם אם נמצא x, ממשיכים להעביר את איברי התור לתור העזר

חיפוש בתור מעגלי

```
//--- חיפוש האם נמצא בתור - תור מעגלי ----
public static boolean doseExist (Queue<Integer> que, int x)
{
    int deme = Integer.MIN_VALUE;    //  הנחה איבר זה לא קיים בתור
    boolean found = false;

    que.insert(deme);
    while (que.head() != deme)
    {
        if (que.head() == x)
            found = true;
        que.insert(que.remove());
    }

    que.remove();    //  הוצאת איבר הדמה

    return found;
}
```

```
//---   פעולה המחזירה אמת אם הערך x ---
//---            נמצא בעץ ושקר אחרת ---
public static boolean isExist (BinNode<Integer> bt, int x)
{
    if (bt == null)
        return false;
    if (bt.getValue() == x)
        return true;
    return isExist (bt.getLeft(), x)  ||  isExist (bt.getRight(), x);
}
```

מחיקה מרשימה יש להחזיר את הרשימה כי ייתכן ו- x הוא האיבר הראשון.

```
//---         פעולה המקבלת רשימה ומספר ---
//--- אם המספר קיים ברשימה, הוא יימחק ---
//---              אחרת - לא יתבצע דבר ---
public static Node<Integer> remove (Node<Integer> lst, int x)
{
    Node<Integer> pos = lst, prev = null;
    while (pos != null  &&  pos.getValue() != x)
    {
        prev = pos;
        pos = pos.getNext();
    }
    if (pos != null)     //  יצאנו מהלולאה כי נמצא x
    {
        if (pos == lst)          //  מחיקה האיבר הראשון ברשימה
            lst = lst.getNext();
        else                     //  מחיקת איבר שאינו ראשון
            prev.setNext(pos.getNext());
    }
    return lst;
}
```

שים לב!

ברשימה סטטית (לא חלק ממחלקה המגדירה אוסף), הרשימה (המזמנת) lst והפרמטר lst הם שתי הפניות שונות. שינוי ב- lst של הפעולה אינו משפיע על lst של התכנית, ולכן:

- מותר לרוץ על הרשימה עם lst (אין צורך בהפניה מיוחדת)
- אם מוחקים בתחילת הרשימה, יש להחזיר את ההפניה החדשה לתחילת הרשימה (lst של התכנית לא יודע שנמחק האיבר הראשון)

ברשימה שהיא תכונה של המחלקה (רשימת הספרים בחנות ספרים, רשימת הקלפים בחפיסת קלפים וכד'), ההפניה היא הקישור היחיד לתחילת הרשימה, ולכן:

- הריצה על הרשימה תיעשה באמצעות הפנית עזר.
- אין צורך להחזיר רשימה כשמוחקים את האיבר הראשון או כשמוסיפים איבר בהתחלה, העדכון נעשה ממילא על **this**.lst.

- צמצום רשימה
  צמצום מחסנית (שכפול מחסנית)
  צמצום תור (שכפול התור באמצעות תור עזר או באמצעות תור מעגלי)

להלן ממשק חלקי של המחלקה Item:

מספר שלם המציין מספר ברשימה מספר מופעיו של המספר ברשימה	**private int** num **private int** amount
פעולה בונה היוצרת עצם מסוג Item שתכונותיו הן מספר שלם המופיע ברשימה ומספר מופעיו ברשימה.	**public** Item (**int** num, **int** amount)

ניתן להניח את קיומן של פעולות get ו- set לכל תכונה.

נתון מבנה נתונים (מסוג רשימה, מחסנית או תור) של מספרים שלמים lst שיש בהם מספרים החוזרים על עצמם יותר מפעם אחת.
כתוב פעולה המקבלת כפרמטר מבנה נתונים כזה של מספרים שלמים ומחזירה רשימה מצומצמת שבה עבור כל מספר המופיע במבנה הנתונים המקורי יופיע איבר אחד מסוג Item שערכיו הם המספר ומספריו מופעיו במבנה הנתונים. אף מספר לא יופיע יותר מפעם אחת ברשימה החדשה. סדר האיברים ברשימה החדשה יהיה כסדר הופעת המופע הראשון במבנה הנתונים המקורי.
אם אתה משתמש בפעולות עזר, כתוב את הפעולה.
שים לב! אין להשתמש במבנה נתונים אחר מעבר לרשימה ול- Item.

לדוגמה, עבור מבנה הנתונים הבא: lst: [3, 2, 1, 5, 3, 3, 8, 9, 7, 1, 7, 5, 5, 3]
תוחזר הרשימה הבאה: lst1: [(3,4), (2,1), (1,2), (5,3), (8,1), (9,1), (7,2)]

צמצום רשימה (lst)

1. צור רשימה מסוג Item ← lst1
2. עבור כל איבר x ברשימה lst
   2.1 אם הוא עדיין לא **<u>מופיע ברשימה</u>** lst1 // lst1.getValue().getNum() == x
      2.1.1 **<u>ספור כמה פעמים</u>** הוא מופיע ב- lst ← count
      2.1.2 צור איבר מסוג Item עם x ו- count והכנס אותו לסופה של הרשימה lst1
   2.2 קדם את lst לאיבר הבא
4. החזר את הרשימה lst1

יעילות הפעולה: $O(n^2)$

מעבר על n האיברים ברשימת המספרים lst, ועבור כל איבר מבצעים:

א.	חיפוש ברשימה החדשה שנבנית	O(n)
ב.	ספירת מספר מופעיו ברשימה lst	O(n)
ג.	הוספת האיבר לרשימת העזר	O(1)
ד.	יצירת איבר והכנסתו לרשימה החדשה	O(1)

סה"כ: $f(n) = n(2n + 2) \Rightarrow 2n^2 + 2n \Rightarrow O(n^2)$

צמצום מחסנית (stk)

1. צור רשימה מסוג Item ← lst
2. שכפל את המחסנית ← sCopy
3. כל עוד המחסנית sCopy לא ריקה (עבור כל איבר x במחסנית sCopy)
   - 3.1 שלוף איבר מראש המחסנית ← x
   - 3.2 count ← 1
   - 3.3 עבור כל איבר y במחסנית
     - 3.3.1 אם הוא שווה ל- x
       count ++
     - 3.3.2 אחרת - הוסף את y ל- sTemp
   - 3.4 צור איבר מסוג Item עם x ו- count והכנס אותו לסופה של הרשימה lst
   - 3.5 החזר (את האיברים שנותרו) ממחסנית העזר ל- sCopy
4. החזר את הרשימה lst

שכפול מחסנית (stk)

1. צור 2 מחסניות sTemp ו- sCopy
2. העבר את כל איברי stk ל- sTemp
3. על עוד sTemp אינה ריקה
   - 3.1 הוצא מהמחסנית ← x
   - 3.2 הכנס אותו ל-2 המחסניות stk ו- sCopy
4. החזר את sCopy

יעילות הפעולה : $O(n^2)$

שיכפול המחסנית (2n צעדים) $O(n)$

מעבר על n האיברים במחסנית המספרים stk, ועבור כל איבר מבצעים :

א. ספירת מספר מופעיו במחסנית stk 2n

ג. הוספת האיבר למחסנית העזר 1

ד. יצירת איבר והכנסתו לרשימה החדשה $O(1)$

סה"כ : $f(n) = O(n) + n(2n + 2) \Rightarrow 2n^2 + 3n \Rightarrow O(n^2)$

צמצום תור (que)

אותו אלגוריתם כמו צמצום מחסנית תוך שימוש בתורי עזר.

שכפול תור - באמצעות תור עזר או באמצעות תור מעגלי.

- **עצים**

תבניות אלגוריתמיות (דפים: תבניות אלגוריתמיות, אוסף פעולות , תבניות בעצים)
סריקת עץ לפי רמות
סכום רמות בעץ
הורים וצאצאים בעץ

כללים לטיפול בעץ בינארי:

- אין לפנות למידע או לבניו של צומת לפני שווידאנו שהצומת לא null
- אין לפנות למידע בבניו של הצומת לפני שווידאנו שיש לו בנים.
- כאשר ערך הצומת הינו מבנה נתונים (מערך, רשימה, מחסנית, תור או אפילו עץ אחר), טפל תחילה הערך הצומת לפי מבנה הנתונים שלו ורק אחר כך המשך בסריקה תחילה על צד שמאל ואחר כך על צד ימין, אלא אם כן נאמר אחרת.
- כאשר בודקים צומת בעץ, יש לבדוק את כל 5 המקרים:
  א. bt הוא עץ ריק bt == null
  ב. bt הוא עלה
  ג. יש ל- bt בן יחיד והוא בן שמאלי
  ד. יש ל- bt בן יחיד והוא בן ימני
  ה. יש ל- bt שני בנים

  שים לב! חובה להתחיל בסעיף א' - עץ ריק. סדר שאר הסעיפים תלוי בשאלה.
- למרות חשיבותה, הפעולה **עלה?** (bt) אינה קיימת ויש לממש אותה

תבניות אלגוריתמיות:

האם כל הצמתים מקיימים תנאי ? (bt)

1. אם (bt == null) החזר אמת
2. אם הצומת הנוכחי אינו מקיים את התנאי החזר שקר
3. החזר צד שמאל מקיים את התנאי **וגם** צד ימין מקיים את התנאי

לדוגמה:
עץ ממויין הוא עץ שבו ערך כל צומת קטן מהערך של כל אחד מבניו וגם בן שמאלי קטן מערך בן ימני.

עץ ממויין ? (bt)

1. אם (bt == null או עלה? (bt))
   החזר אמת
2. אם (יש ל- bt בן שמאלי, וגם ערך הבן השמאלי קטן מערך השורש) החזר שקר
3. אם (יש ל- bt בן ימני, וגם ערך הבן הימני קטן מערך השורש) החזר שקר
4. אם (יש ל- bt בן שמאלי וגם יש ל- bt בן ימני
   וגם ערך הבן השמאלי גדול מערך הבן הימני) החזר שקר
5. החזר עץ ממויין (בן שמאלי) **וגם** עץ ממויין (בן ימני)

שים לב! אם אחד הצדדים יחזיר שקר, יהיה הערך הביטוי שקר.

האם לפחות צומת אחד מקיים תנאי (bt) (או האם **חלק מהעץ מקיים את התנאי**)

1. אם (bt == null) החזר שקר
2. אם (הצומת הנוכחי מקיים את התנאי) החזר אמת
3. החזר צד שמאל מקיים את התנאי **או** צד ימין מקיים את התנאי

<u>דוגמאות</u>:

* האם נמצא (bt, x)
* עץ בעל שרשרת x הוא עץ שקיים בו מסלול המתחיל בשורש ומסתיים באחד העלים וערך כל צמתי המסלול הוא x

עץ-בעל-שרשרת-x (bt, x)

1. אם (bt == null) החזר שקר
2. אם (עלה? (bt) וגם ערך העלה הוא x) החזר אמת
3. אם (ערך הצומת שונה מ- x) החזר שקר
4. החזר **עץ-בעל-שרשרשת-x** (בן שמאלי) **או** **עץ-בעל-שרשרשת-x** (בן ימני)

סריקה לפי רמות (bt)

1. צור תור של עצים que ← תור מסוג Queue<BinNode<Integer>>
2. הכנס את bt לתור
3. כל עוד התור לא ריק,
   - 3.1 הוצא מהתור bt ←
   - 3.2 הצג את ערך השורש של bt
   - 3.3 אם יש ל- bt בן שמאלי, הכנס אותו לתור
   - 3.4 אם יש ל- bt בן ימני, הכנס אותו לתור

סכום צמתים ברמה (bt, level)

1. אם (bt == null) החזר 0
2. אם (level == 0) החזר את ערך הצומת
3. החזר **סכום-צמתים-ברמה** (בן שמאלי של bt, level-1) +
   סכום-צמתים-ברמה (בן ימני של bt, level-1)

רמת-הצומת (bt, t) /* הפעולה מחזירה את מספר הרמה של צומת t אם הוא בעץ bt אחרת יוחזר 1- */

1. אם (bt == null) החזר 1-
2. אם (bt == t) החזר 0
3. **רמת-הצומת** (t, בן שמאלי של bt) ← left
4. אם (left >= 0) החזר left+1 // נמצא הצומת בצד שמאל
5. **רמת-הצומת** (t, בן ימני של bt) ← right
6. אם (right >= 0) החזר right +1 // נמצא הצומת בצד ימין
7. החזר 1- // הצומת לא נמצא בצד זה

הורה של צומת: פעולה המקבלת הפנייה לשורש העץ והפנייה לצומת בעץ שאינו השורש ומחזירה הפניה להורה של הצומת.

הורה (bt, t)

1. אם (bt == null **או** בן שמאלי של bt שווה ל- t **או** בן ימני של bt שווה ל t) החזר את bt
2. **הורה** (t , בן-שמאלי של bt) ← left
3. אם (left != null) // ההורה נמצא בצד שמאל
   החזר את left
4. החזר **הורה** (t , בן-ימני של bt)

חיפוש צאצאים של צומת:

- האם צאצא? (bt, x, y) מוחזר אמת אם קיימים בעץ bt שני צמתים שערכיהם x ו- y והאחד צאצא של האחר, ושקר אחרת.
- צאצא? (bt, t1, t2) - t1 הוא צאצא של t2 אם קיים מסלול מצומת t2 לצומת t1 .
  הנחה: t1 ו- t2 הם הפניות לצמתים בעץ bt.
- סבא-רבא (bt, t1, t2) חיפוש הצומת הקרוב ביותר לעלים t1 ו- t2 המהווה אב קדמון של שניהם

האם-צאצא? (bt, x, y) // x ו- y הם ערכי הצמתים

1. אם (bt == null) החזר שקר
2. אם (ערך הצומת הוא x)
   החזר **נמצא-בעץ?** (y, בן-שמאלי של bt) **או** **נמצא-בעץ?** (y, בן-ימני של bt)
3. אם (ערך הצומת הוא y)
   החזר **נמצא-בעץ?** (x, בן-שמאלי של bt) **או** **נמצא-בעץ?** (x, בן-ימני של bt)
4. החזר **האם-צאצא?** (bt.GetLeft(), x, y) **או** **האם-צאצא?** (bt.GetRight(), x, y)

צאצא? (bt, t1, t2) // t1 ו- t2 הם **הפניות** לשני צמתים. מוחזר אמת אם t1 צאצא של t2

1. אם (t2 == null)
   החזר שקר // t1 לא נמצא במסלול זה
2. אם (t2 == t1)
   החזר אמת
3. החזר **צאצא?** (t2, בן-שמאלי של bt ,t1) **או** **צאצא?** (t2, בן-ימני של bt ,t1)

(*) הפעולה בושקת האם t2 הוא צאצא של t1.
כדי לדעת האם קיים יחס של צאצאים בין t1 ו- t2
נפעיל את הפעולה: החזר **צאצא?** (bt, t1, t2) או **צאצא?** (bt, t2, t1)

בגרות תש"ס, 2000:

נגדיר **סבא-רבא** של שני עלים שונים בעץ בינארי הוא ה**אב הקדמון** המשותף לשני העלים כלשהם בעץ.

לפניך הפעולה: **סבא-רבא** (עלה1, עלה2, עץ): הפעולה מחזירה את הצומת שהוא **סבא-רבא** של שני העלים. הנחה, שני העלים נמצאים בעץ.

יש לממש את הפעולה תוך שימוש בפעולות: **האם-צאצא?** (צומת1, צומת2, עץ)

הורה (צומת, עץ)

רעיון הפתרון: t1 ו- t2 הם הפניות לשני עלים בעץ bt.
נבחר באחד העלים, נניח t1 ונבדוק אם t2 הוא צאצא של אבא של t1
אם כן, נחזיר את ההורה.
אחרת, נבדוק באופן רקורסיבי את ההורה של האבא.

הרקורסיה תעצור כאשר נמצא צומת שהוא אב-קדמון של שני העלים או כאשר נגיד לשורש bt

סבא-רבא (bt, t1, t2)
1. **צאצא?** ((bt, t1, t2)) // אם t2 הוא צאצא של t2, נמצא האב הקדמון של שני העלים
   החזר את t1
2. החזר **סבא-רבא** (t2, **הורה** (bt, t1), bt)