עץ חיפוש בינארי

אוסף פעולות

בניית עץ חיפוש בינארי:

```
//--- בנית עץ חיפוש בינארי ---
public static BinNode<Integer> buildSearchTree()
{
	//--- בניית צומת ראשון בעץ ---
	System.out.print ("type 1st number (-1 to finish) --> ");
	int num = input.nextInt();
	BinNode<Integer> bt = new BinNode<Integer>(num);

	//--- קלט והוספת הצמתים הבאים ---
	System.out.print ("type next number (-1 to finish) --> ");
	num = input.nextInt();
	while (num !=  -1)
	{
		add(bt, num);

		System.out.print ("type next number --> ");
		num = input.nextInt();
	}
	return bt;
}

//--- פעולה המוסיפה צומת לעץ חיפוש בינארי באופן איטרטיבי ---
public static void add (BinNode<Integer> bt, int x)
{
	BinNode<Integer> t = new BinNode<Integer>(x);

	while (x < bt.getValue()  &&  bt.hasLeft() ||
	       x >= bt.getValue()  &&  bt.hasRight())
	{
		if (x < bt.getValue())
			bt = bt.getLeft();
		else
			bt = bt.getRight();
	}
	if (x < bt.getValue())
		bt.setLeft(t);
	else
		bt.setRight(t);

}
//--- הצגת צמתי העץ בסדר תוכי ---
public static void printInOrder (BinNode<Integer> bt)
{
	if (bt != null)
	{
		printInOrder(bt.getLeft());
		System.out.print(bt.getValue() + ", ");
		printInOrder(bt.getRight());
	}

}
```

הוספת צומת לעץ חיפוש בינארי - פתרון רקורסיבי

```
//--- פעולה המוסיפה צומת לעץ חיפוש בינארי באופן רקורסיבי ---
public static void add (BinNode<Integer> bt, int x)
{
     if (x < bt.getValue())
     {
         if (! bt.hasLeft())                              //    אם אין בן שמאלי
             bt.setLeft(new BinNode<Integer>(x));         // הוספת הצומת כבן שמאלי
         else
             add (bt.getLeft(), x);          //  המשך חיפוש המקום המתאים בצד שמאל
     }
     else    // x >= bt.getValue()
     {
         if (! bt.hasRight())                             //    אם אין בן ימני
             bt.setRight(new BinNode<Integer>(x));        // הוספת הצומת כבן ימני
         else
             add (bt.getRight(), x);         //  המשך חיפוש המקום המתאים בצד ימין
     }
}
```

חיפוש איבר בעץ חיפוש בינארי:

- **פתרון רקורסיבי:**

```
//--- פעולה המחזירה "אמת" אם x נמצא בעץ חיפוש בינארי bt ---
//---                                                  ו-"שקר" אחרת ---
public static boolean exist (BinNode<Integer> bt, int x)
{
      if (bt == null)
            return false;
      if (bt.getValue() == x)
            return true;
      if (x < bt.getValue())
            return exist (bt.getLeft(), x);
      return exist (bt.getRight(),x);
}
```

- **פתרון איטרטיבי:**

```
//--- פעולה המחזירה "אמת" אם x נמצא בעץ חיפוש בינארי bt ---
//---                                                  ו-"שקר" אחרת ---
public static boolean exist (BinNode<Integer> bt, int x)
{
      while (bt != null)
      {
            if (bt.getValue() == x)
                  return true;

            if (x < bt.getValue())
                  bt = bt.getLeft();
            else
                  bt = bt.getRight();
      }
      return false;

}
```

החזרת הפנייה לצומת:

```
//--- פעולה המקבלת עץ חיפוש ומספר, ומחזירה הפנייה לצומת. ---
//---                        אם הערך לא נמצא בעץ יוחזר null ---
public static BinNode<Integer> getNode
                              (BinNode<Integer> bt, int x)
{
      if (bt == null  ||  bt.getValue() == x)
            return bt;
      if (x < bt.getValue())
            return getNode (bt.getLeft(), x);
      return getNode (bt.getRight(),x);

}
```

מציאת הערך הגדול ביותר בעץ חיפוש:

- **פתרון רקורסיבי:**

```
//--- מציאת האיבר הגדול ביותר בעץ חיפוש ---
//---                  null הנחה: העץ אינו ---
public static int biggest (BinNode<Integer> bt)
{
      if (bt.hasRight())
            return bt.getValue();
      return biggest(bt.getRight());
}
```

מציאת הערך הקטן ביותר בעץ חיפוש:

- **פתרון איטרטיבי:**

```
//--- מציאת האיבר הקטן ביותר בעץ חיפוש ---
//---                  null הנחה: העץ אינו ---
public static int smallest (BinNode<Integer> bt)
{
      while (bt.hasLeft())
            bt = bt.getLeft();
      return bt.getValue();
}
```

מציאת ההורה הישיר של הצומת בעץ חיפוש:

- **פתרון רקורסיבי:**

```
//--- פעולה המחזירה הפנייה להורה של צומת נתון בעץ חיפוש --
public static BinNode<Integer> parent
                  (BinNode<Integer>bt, BinNode<Integer>node)
{
      if (bt==null || bt.getLeft() == node || bt.getRight() == node)
            return bt;
      if (node.getValue() < bt.getValue())
            return parent(bt.getLeft());
      return parent(bt.getRight());
}
```

- **פתרון איטרטיבי:**

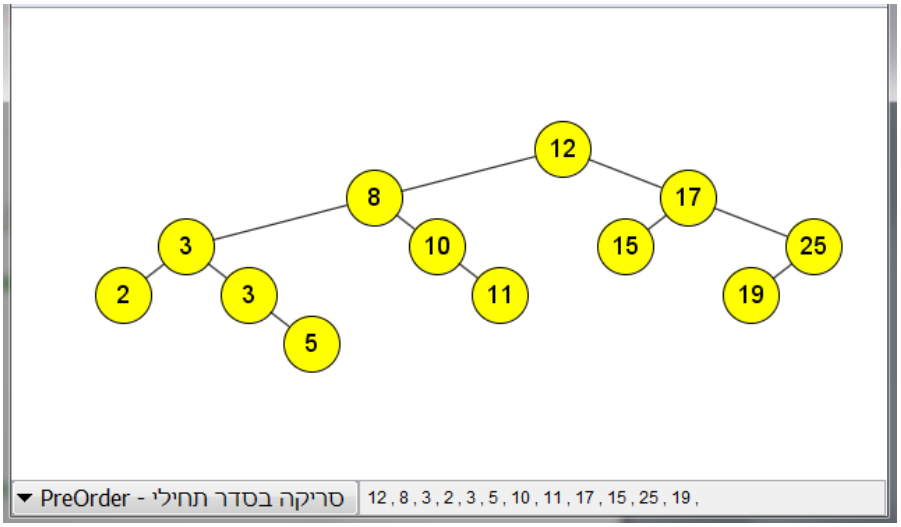
```
//--- פעולה המחזירה הפנייה להורה של צומת נתון בעץ חיפוש --
public static BinNode<Integer> parent
                  (BinNode<Integer>bt, BinNode<Integer>node)
{
      while (bt != null)
      {
            if (bt.getLeft() == node || bt.getRight() == node)
                  return bt;

            if (node.getValue() < bt.getValue())
                  bt = bt.getLeft();
            else
                  bt = bt.getRight();
      }
      return bt;

}
```

הפנייה לצומת בעל הערך העוקב בסדר תחילי בצומת בעץ חיפוש:

```
//--- האיבר העוקב - פעולה המקבלת הפנייה לצומת בעץ חיפוש ---
//---        ומחזירה הפניה לצומת העוקב לו  (בסדר עולה).  ---
//---
//---      bt העץ עצמו , node הצומת שאת העוקב לו מחפשים ---
//---                 null יוחזר ,אם אין עוקב לערך בעץ זה ---
public static BinNode<Integer> successor
                  (BinNode<Integer> bt, BinNode<Integer> node)
{
      //--- אם הכי גדול בעץ, אין לו עוקב ---
      if (node.getValue() == biggest(bt))
            return null;

      BinNode<Integer> t = node.getRight();

      //--- אם יש בן ימני, נחפש את המינימום בעץ שבראשו הבן הימני ---
      if (t != null)
      {
            while (t.hasLeft())
                  t = t.getLeft();
      }
      else //--- חיפוש ההורה באבות הקדמונים ---
      {
            t = parent (bt, node);
            while (t != bt && t.getRight() == node)
            {
                  node = t;
                  t = parent (bt, node);
            }
      }
      return t;

}
```

למשל: העוקב ל- 5 הוא 8

למשל: העוקב ל- 17 הוא 19



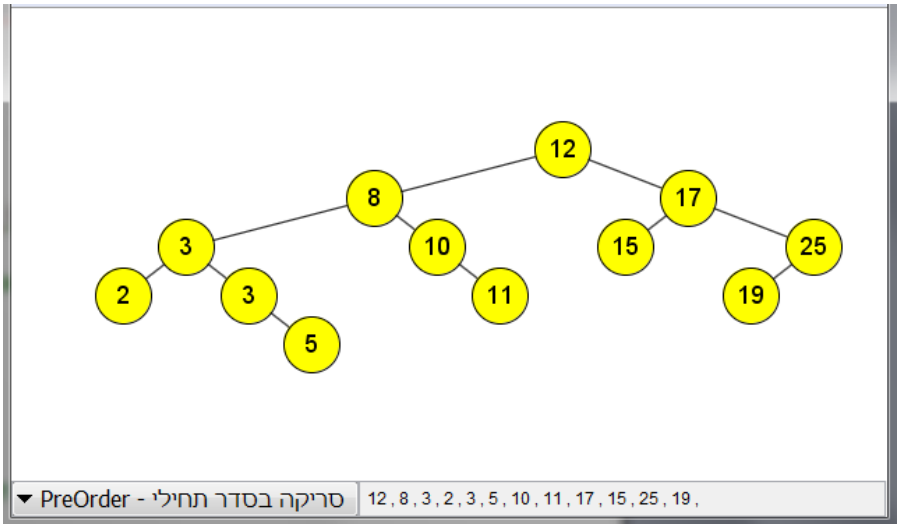
פתרון פשוט יותר: סרוק את העץ לתוך רשימה ומצא את העוקב-ברשימה לצומת.

הפנייה לצומת בעל הערך הקודם בסדר תחילי בצומת בעץ חיפוש:

```
//--- האיבר הקודם - פעולה המקבלת הפנייה לצומת בעץ חיפוש ---
//---        ומחזירה הפניה לצומת הקודם לו  (בסדר עולה).  ---
//---
//---      bt העץ עצמו , node הצומת שאת הקודם לו מחפשים ---
//---                 null אם אין קודם לערך בעץ זה, יוחזר ---
public static BinNode<Integer> predeccessor
                  (BinNode<Integer> bt, BinNode<Integer> node)
{
      //--- אם הכי קטן בעץ, אין לו קודם ---
      if (node.getValue() == smallest(bt))
            return null;

      BinNode<Integer> t = node.getLeft();

      //--- אם יש בן שמלי, נחפש את המקסימום בעץ שבראשו הבן השמלי ---
      if (t != null)
      {
            while (t.hasRight())
                  t = t.getRight();
      }
      else //--- חיפוש ההורה באבות הקדמונים ---
      {
            t = parent (bt, node);
            while (t != bt && t.getLeft() == node)
            {
                  node = t;
                  t = parent (bt, node);
            }
      }
      return t;
}
```

למשל: הקודם ל- 8 הוא 5

למשל: הקודם ל- 19 הוא 17



פתרון פשוט יותר: סרוק את העץ לתוך רשימה ומצא את הקודם-ברשימה לצומת.