**אוסף שאלות על עצים בינריים: 2**

**25.// מחזירה הפנייה לאבא של צומת**

public static BinNode<Integer> **parent**(BinNode<Integer> tr, BinNode<Integer> son)

{

if (tr == null || son == null)

return null;

if (son == tr.getLeft() || son == tr.getRight())

return tr;

BinNode<Integer> p = parent(tr.getLeft(), son);

if (p != null)

return p;

// p==null

return parent(tr.getRight(), son);

}

**26. // האם שני צמתים הם אחים**

public static boolean **areBrothers**(BinNode<Integer> tr,

BinNode<Integer> son1, BinNode<Integer> son2)

{

return son1 != son2

&& parent(tr, son1) == parent(tr, son2);

}

**26.// האם אחים גרסה 2**

public static boolean **areBrothers1**(BinNode<Integer> tr,

BinNode<Integer> son1, BinNode<Integer> son2)

{

BinNode<Integer> p = parent(tr, son1);

return son2 == tr.getLeft() || son2 == tr.getRight();

}

**בודק אם x נמצא בעץ, אם כן, האם y הוא צאצא שלו 27.//**

public static boolean **isDescendent**(BinNode<Integer> tr, int x, int y)

{

BinNode<Integer> xNode = existN(tr, x);

if (xNode == null)

{

return false;

}

// x exists in tr

return existN(xNode, y);

}

**28.// מחזירה אב קדמון ראשון שמשותף לשני צמתים בעץ**

public static BinNode<Integer> **commonAncestor**(BinNode<Integer> tr,

BinNode<Integer> s1, BinNode<Integer> s2)

{

BinNode<Integer> parentS1 = parent(tr, s1);

if (parentS1 == null)

{

return null;

}

if (isDescendent(tr, parentS1, s2))

{

return parentS1;

}

return commonAncestor(tr, parentS1, s2);

}

**29. // רמה של צומת: הנחה העץ לא ריק , הצומת נמצאת**

public static int **level**(BinNode<Integer> tr, BinNode<Integer> node)

{

if (tr == null || node == tr)

{

return 0;

}

return 1 + level(tr, parent(tr, node));

}

**30. // דומים- האם שני העצים זהים במבנה שלהם**

public static boolean **similar**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

if (tr1 == null && tr2 == null)

{

return true;

}

// not both are null

if (tr1 == null || tr2 == null)

{

return false;

}

return similar(tr1.getLeft(), tr2.getLeft())

&& similar(tr1.getRight(), tr2.getRight());

}

**30. // גירסה2 האם שני העצים זהים במבנה שלהם**

public static boolean **similar1**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

if (tr1 == null && tr2 == null)

{

return true;

}

if (tr1.hasLeft() && !tr2.hasLeft()

|| !tr1.hasLeft() && tr2.hasLeft()

|| tr1.hasRight() && !tr2.hasRight()

|| !tr1.hasRight() && tr2.hasRight())

{

return false;

}

return similar1(tr1.getLeft(), tr2.getLeft())

&& similar1(tr1.getRight(), tr2.getRight());

}

**31.// האם שני העצים זהים במבנה שלהם וגם בתוכן**

public static boolean **equal**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

if (tr1 == null && tr2 == null)

return true;

if (tr1 == null || tr2 == null)

return false;

// both are not null

return tr1.getValue() == tr2.getValue()

&& equal(tr1.getLeft(), tr2.getLeft())

&& equal(tr1.getRight(), tr2.getRight());

}

**32.// בודקת אם כל ערכי העץ הראשון נמצאים בעץ השני**

public static boolean **allT1InT2**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

if (tr1 == null)

return true;

if (tr2 == null)

return false;

return exist(tr2, tr1.getValue())

&& allT1InT2(tr1.getLeft(), tr2) && allT1InT2(tr1.getRight(), tr2);

}

**33.// בודק עם לשני העצים אותם ערכים**

public static boolean **allExists**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

return allT1InT2(tr1, tr2) && allT1InT2(tr2, tr1);

}

**34.// בודק אם העץ השני מוכל בעץ הראשון החל מהשורש**

public static boolean **insideFromRoot**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

if (tr1 == null)

return true;

if (tr2 == null)

return false;

if (tr1.getValue() != tr2.getValue())

return false;

return insideFromRoot(tr1.getLeft(), tr2.getLeft())

&& insideFromRoot(tr1.getRight(), tr2.getRight());

}

**// בודק אם העץ השני מוכל בעץ הראשון**

35. public static boolean **inside**(BinNode<Integer> tr1, BinNode<Integer> tr2)

{

return insideFromRoot(tr1, tr2)

|| insideFromRoot(tr1, tr2.getLeft())

|| insideFromRoot(tr1, tr2.getRight());

}

}

**36. פעולה המקבלת עץ של תווים ומחזיר רשימה של Pair בה יופיע כל תו ומספר הופעותיו בעץ**

**public** **static** Node<Pair> fromTreeToList (BinNode<Character> t,BinNode<Character> t2)

{

Node< Pair > r = **null**;

r=*add*(t,t2, r);

**return** r;

}

**public** **static** Node<Pair> add(BinNode<Character> t,BinNode<Character> t2, Node<Pair> r)

{

**if**(t==**null**)

**return** r;

**if** (!*exists*(r, t.getValue()))

r=**new** Node<Pair>(**new** Pair(t.getValue(),*count*(t2,t.getValue())),r);

r=*add*(t.getLeft(),t2,r);

r=*add*(t.getRight(),t2,r);

**return** r;

}

**public** **static** **int** count( BinNode<Character> t, **char** c )

{

**if**( t == **null** )

**return** 0;

**if**( t.getValue() == c )

**return** 1 + *count*( t.getLeft(),c ) + *count*( t.getRight(),c );

**return** *count*( t.getLeft(),c ) + *count*( t.getRight(),c );

}

**public** **static** **boolean** exists(Node<Pair> list, **char** x)

{

**if**(list==**null**)

**return** **false**;

**if**(list.getValue().getTav()==x)

**return**(**true**);

**return** *exists*(list.getNext(), x);

}