

מבני נתונים

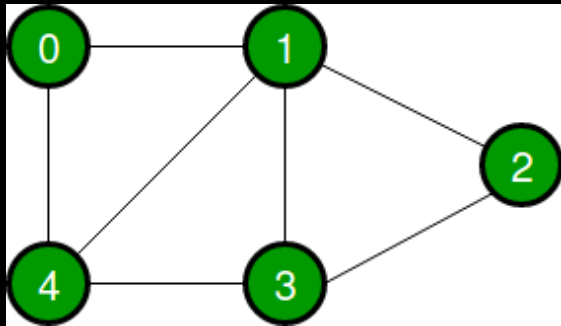
תרגול 4 – עצים

היום

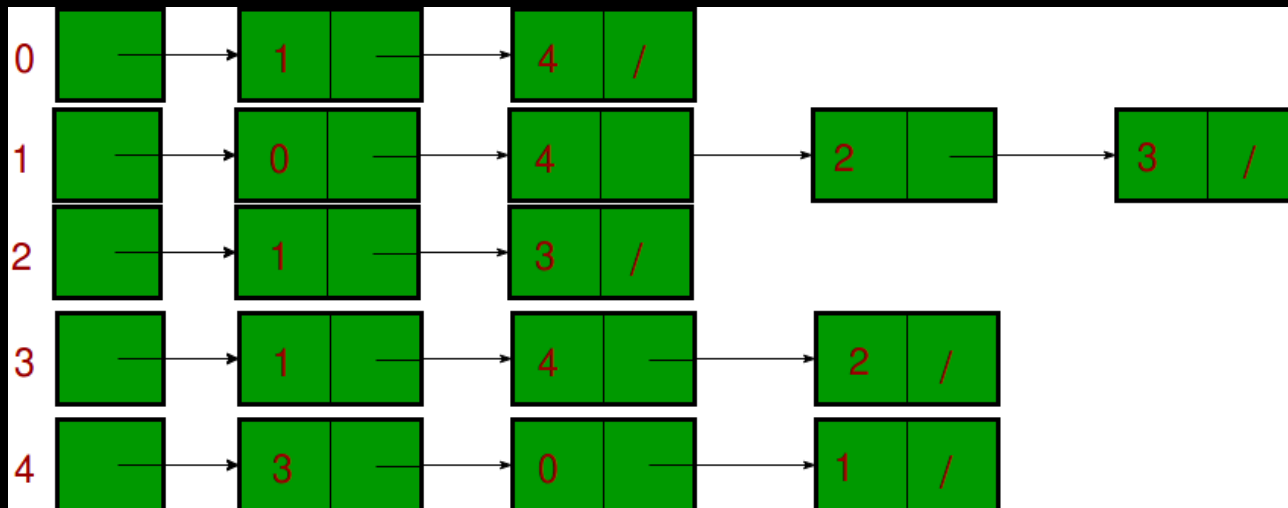
- עצים
- עצים בינאריים
- עץ חיפוש בינארי
- תכונות
- מימוש
- פתרון תרגילים

גרף

גרף Graph (G) – אוסף של כל הקודקודים וצלעות המחברים זוגות של קודקודים
מיוצג במחשב בד"כ ע"י Adjacency Matrix או Adjacency List



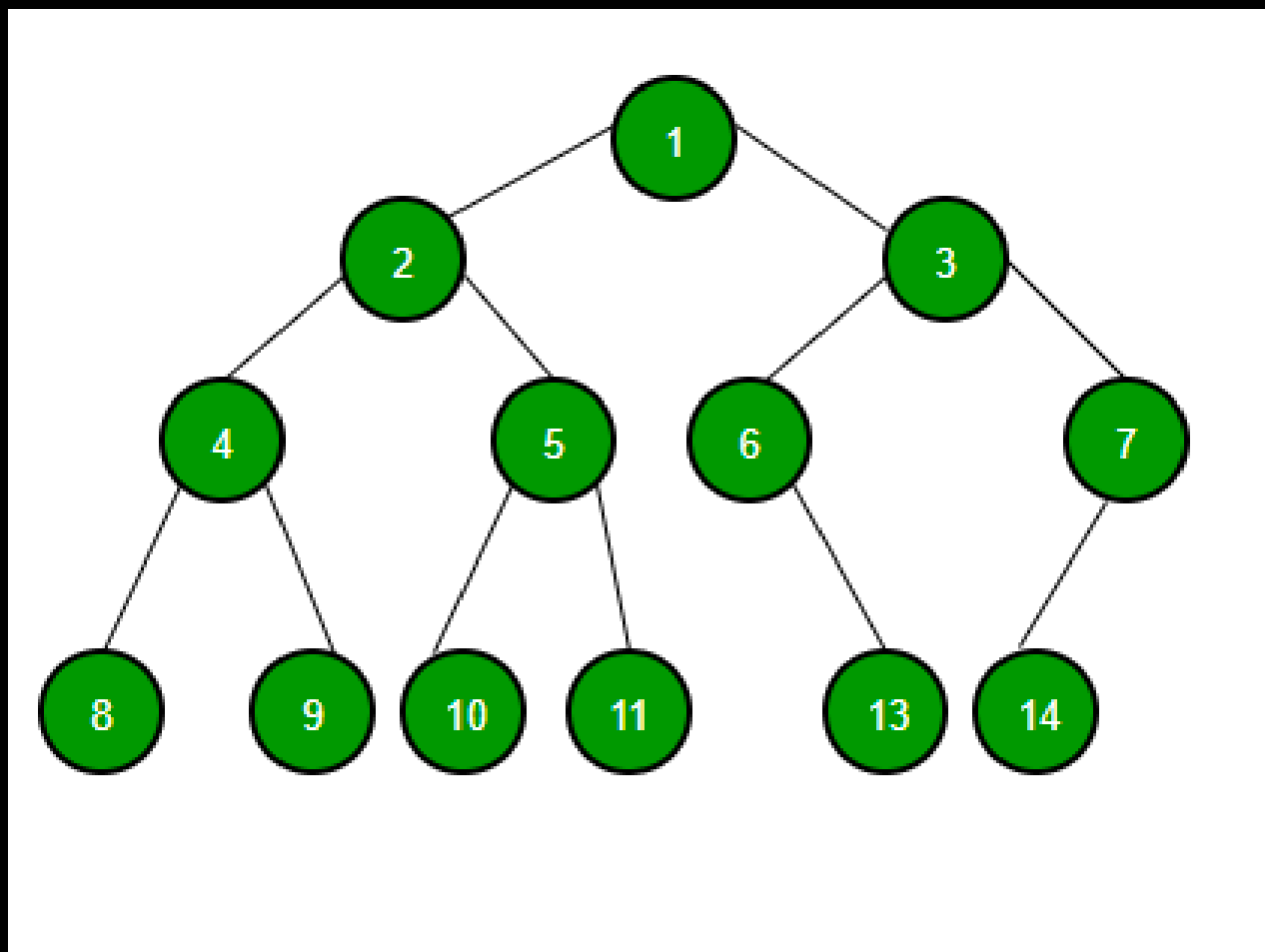
Adjacency List



Adjacency Matrix

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

עץ



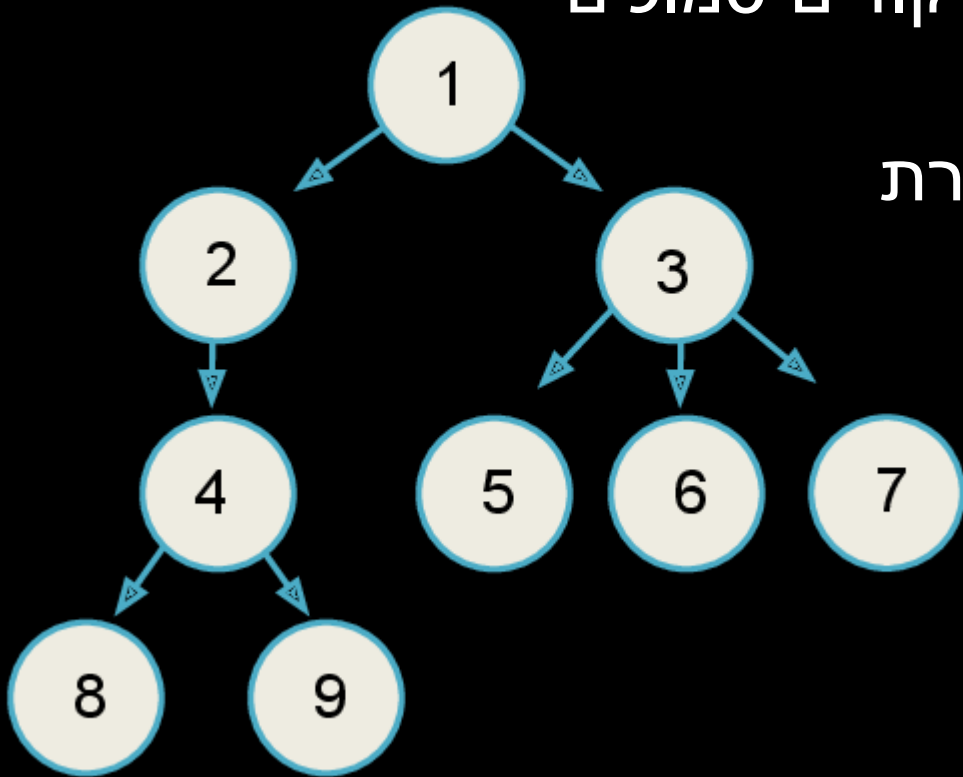
עץ Tree (T) – עץ הוא גרף חסר מעגלים

יהי G גרף עם n קודקודים, אז G הוא עץ אם קיימים שניים מבין התנאים הבאים:

1. G קשיר
2. G חסר מעגלים
3. G מכיל $n - 1$ צלעות

הגדרות

Tree



צומת (Node) – איבר בתוך העץ

מסלול (Path) – סדרת קודקוד בגרף בה כל שני קודקודים סמוכים מחוברים ע"י צלע

אב (Parent) – צומת שיוצאת ממנה צלע לצומת אחרת

בן (Child) – צומת שיש לו אב

- Left Child

- Right Child

שורש (Root) – צומת יחיד בעץ שאין לו אב

עלה (Leaf) – צומת שאין לו בנים

הגדרות

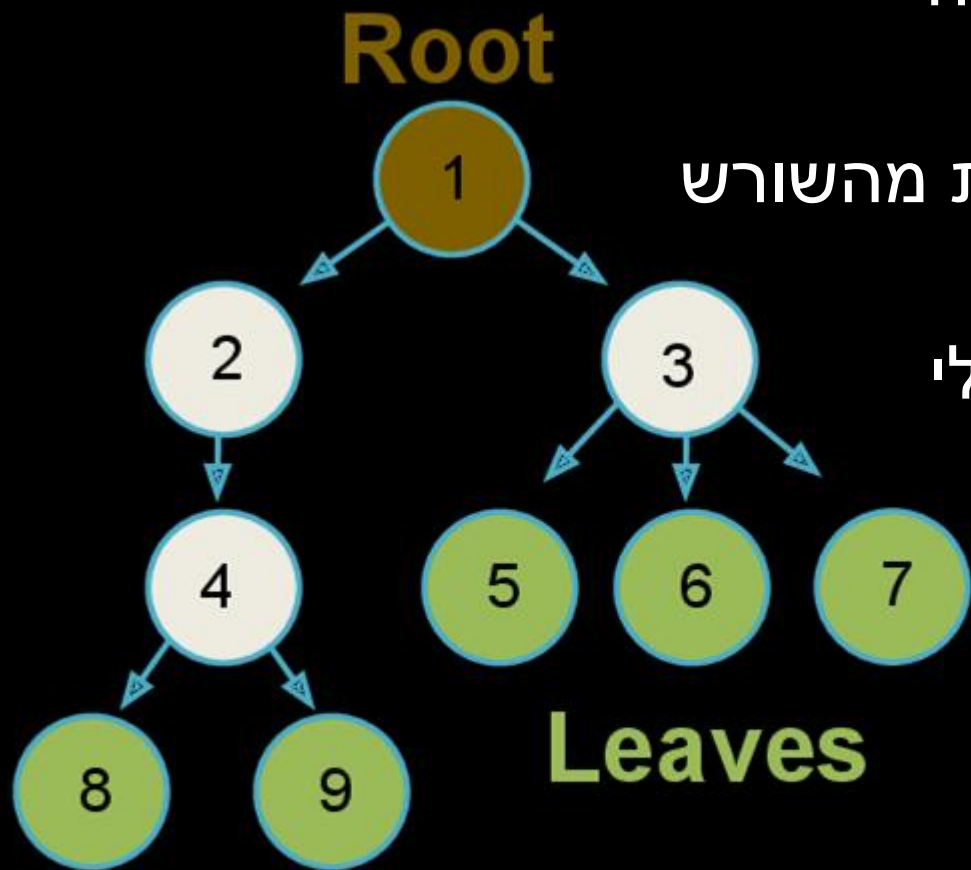
צומת פנימי (Internal Node) – צומת אשר אינו עלה

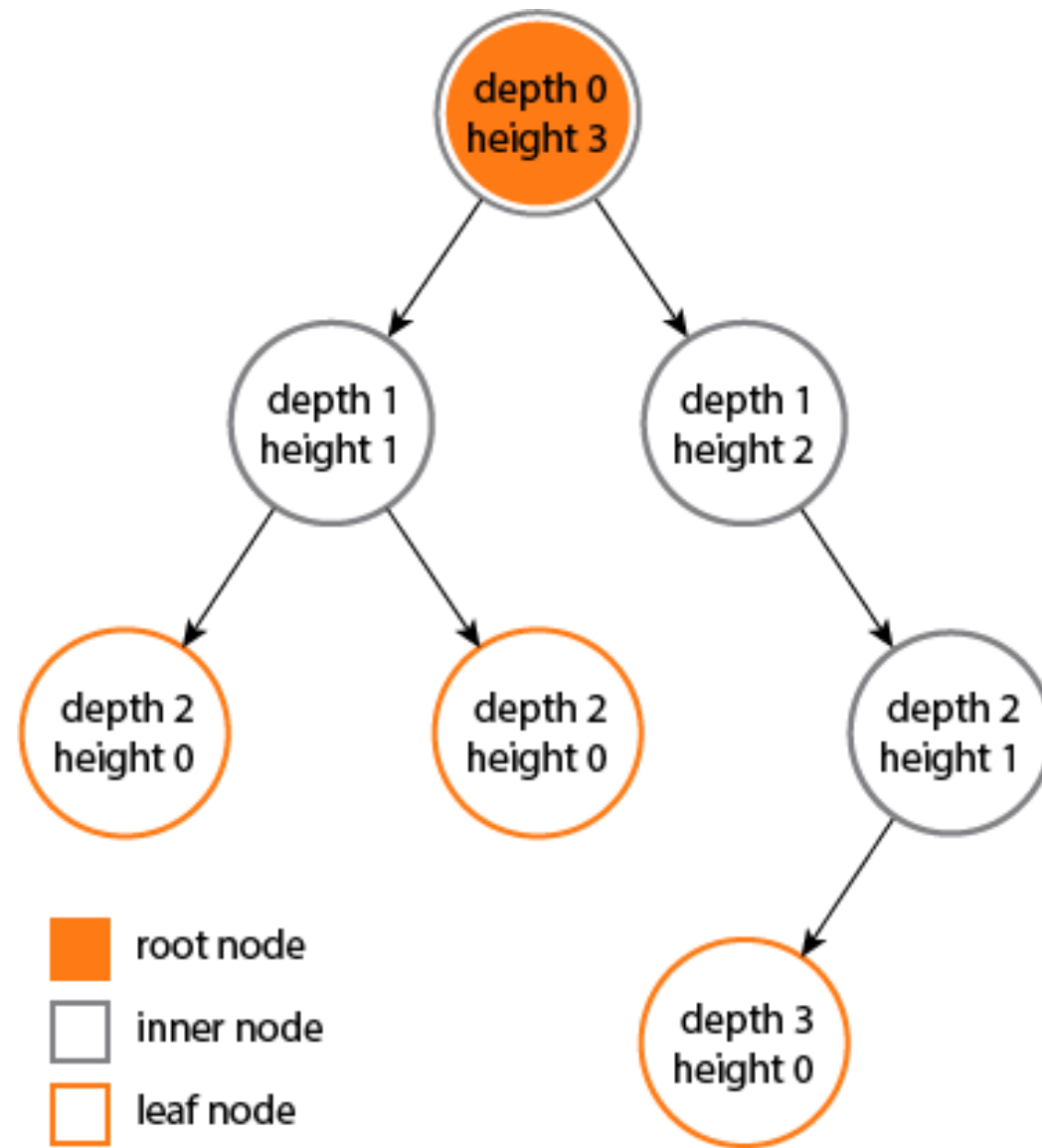
עומק צומת (Depth) – מרחק (מספר צלעות) הצומת מהשורש

גובה (Height) – המרחק (מספר צלעות) המקסימלי של צומת מעלה (בתת עץ שלו)

גובה העץ (Tree Height) – המרחק המקסימלי מהשורש עד עלה (נסמן בד"כ ע"י h)

אחים/שכנים (Siblings) – צומת שיש להם אותו אב





שימוש בעצים

משתמשים בעצים על מנת לאחסן מידע **היררכי**.

מע"ה משתמשת בעצים בשביל תיקיות Files and Folders

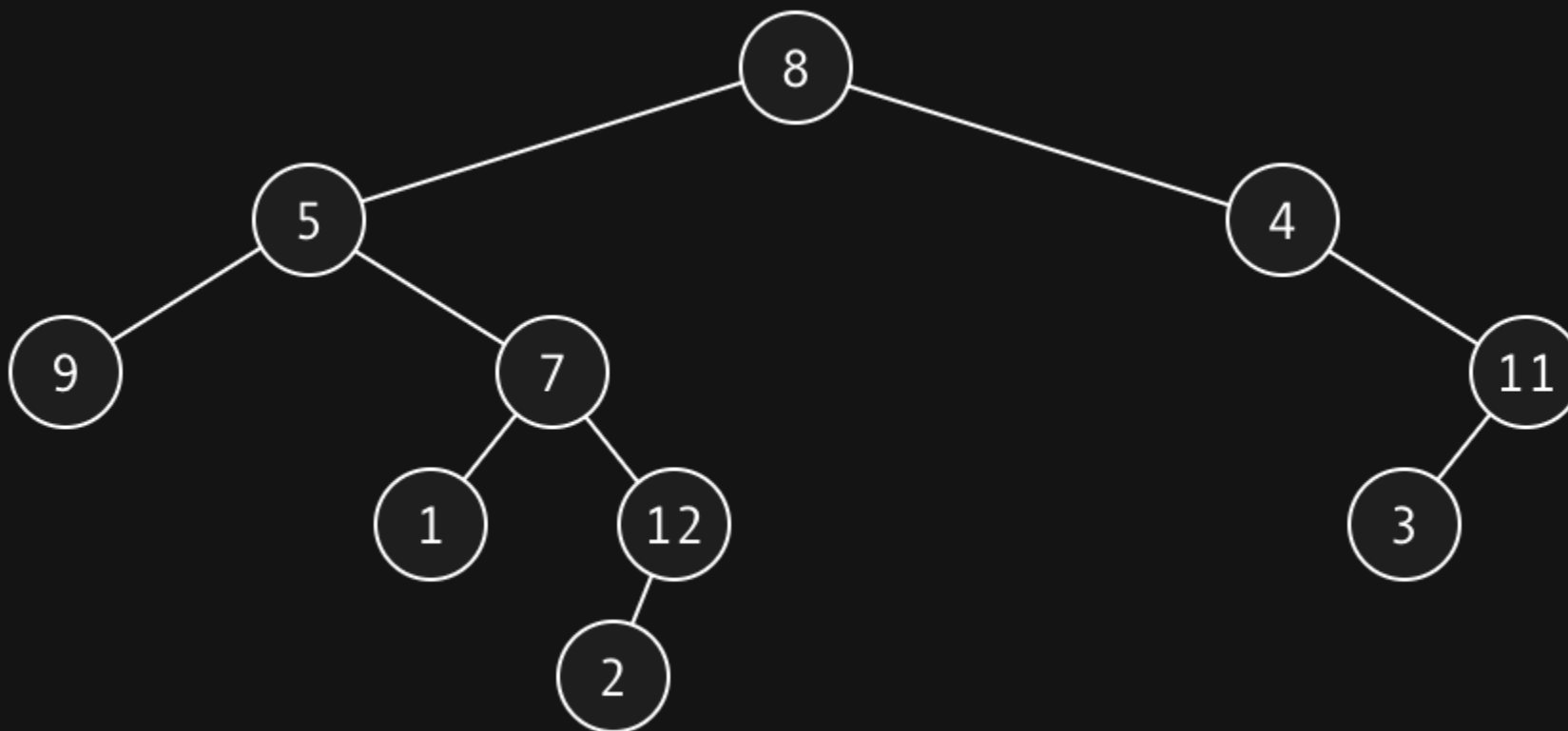
מבנה נתונים **דינאמי** – קל להוסיף ולמחוק קודקודים (מידע)

בעץ בעל **סדר** קל לבצע חיפוש, מיון ואלגוריתמי סיור בעץ (Tree traversal algorithms)

...

עץ בינארי

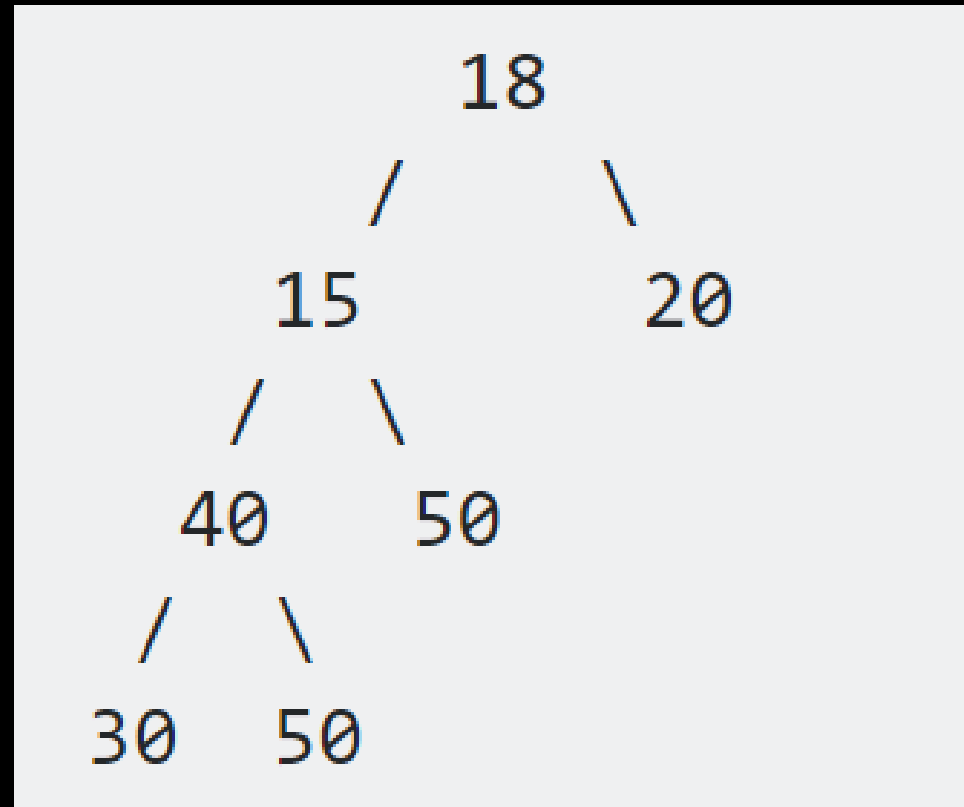
עץ בינארי (Binary Tree) – עץ אשר לכל קודקוד יש לכל היותר 2 בנים



עץ בינארי מלא

עץ בינארי מלא (Full Binary Tree)

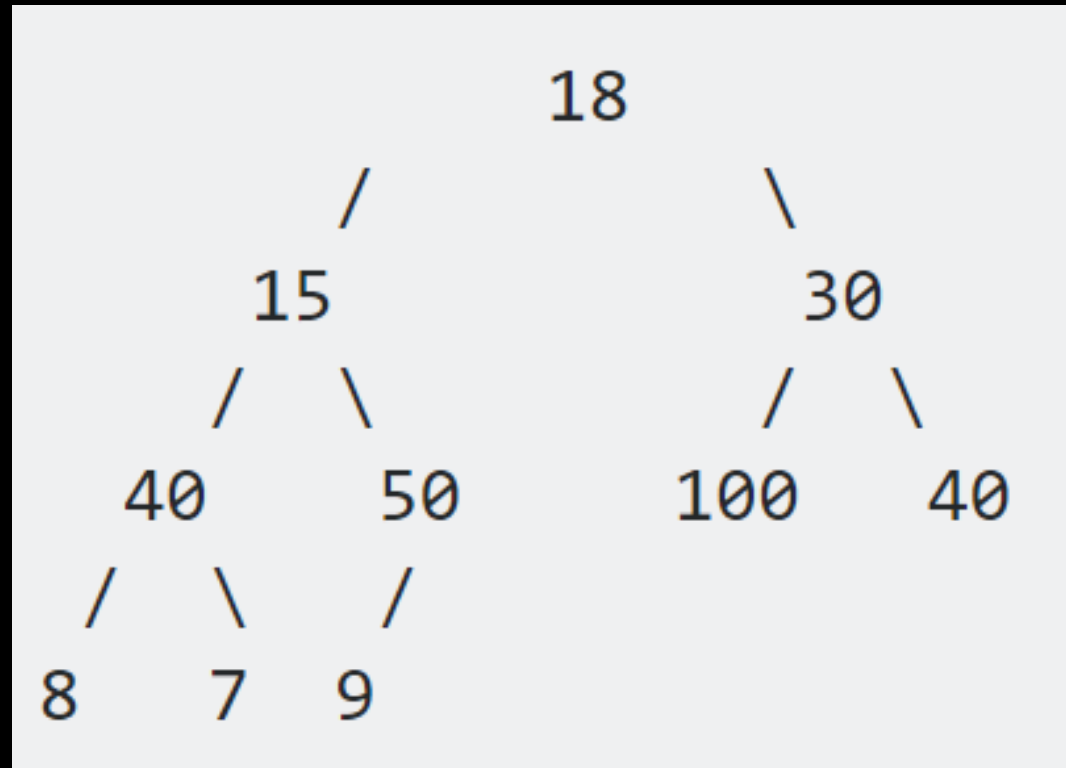
עץ בינארי הוא מלא אם כל לול קודקוד יש 0 או 2 ילדים



עץ בינארי שלם

עץ בינארי שלם (Complete Binary Tree)

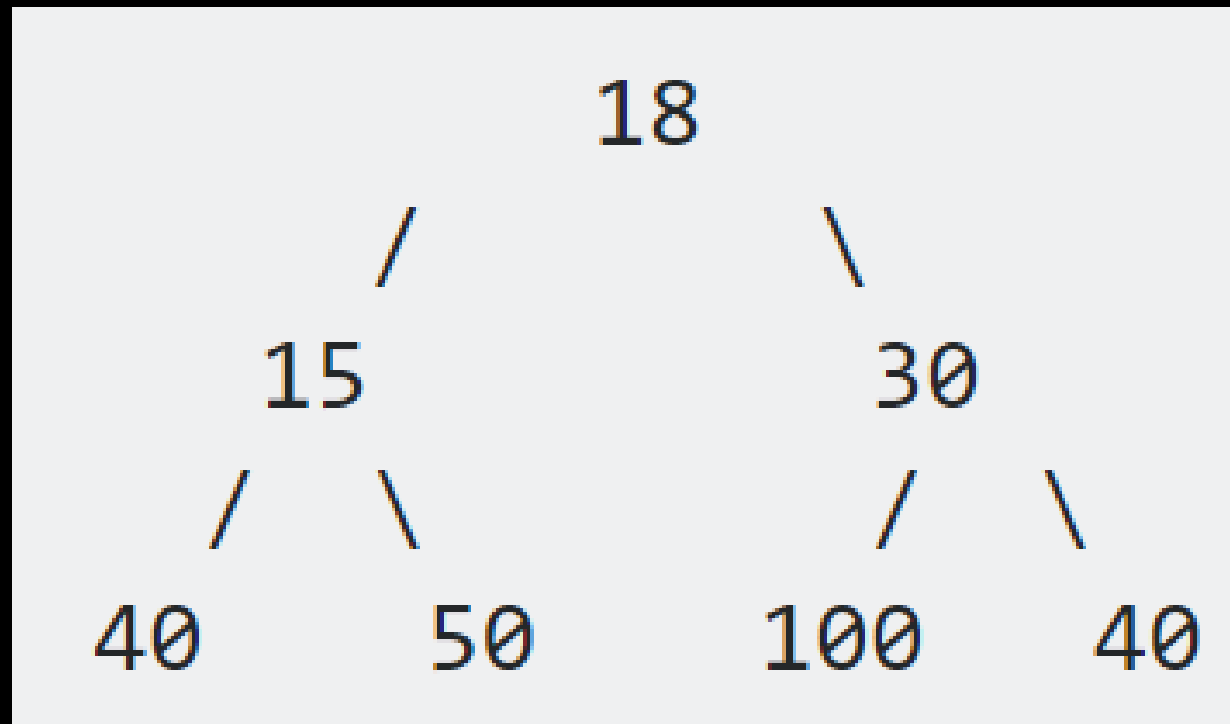
עץ בינארי הוא עץ בינארי שלם אם כל הרמות מלאים בעלים חוץ מהרמה האחרונה כאשר כל העלים ברמה האחרונה הם מצד שמאל ככל היותר

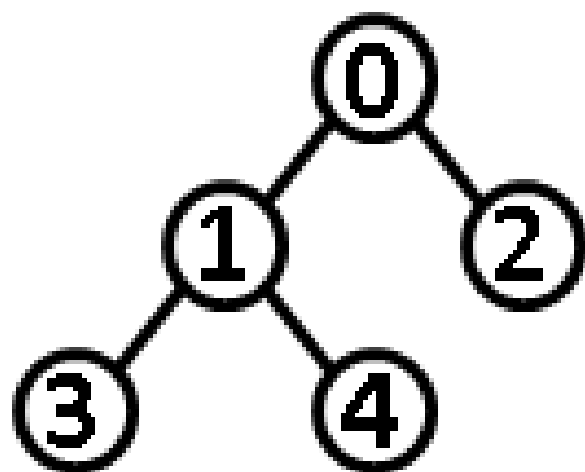


עץ בינארי מושלם

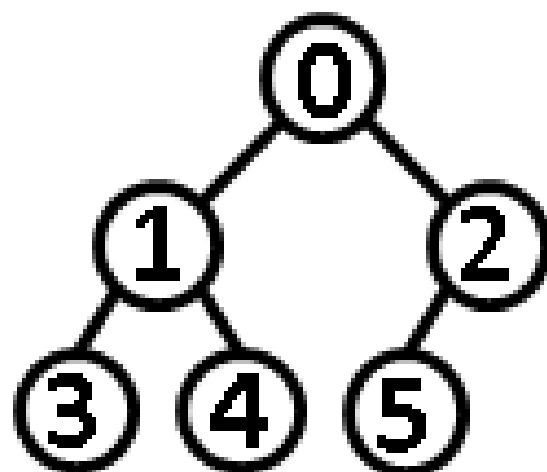
עץ בינארי מושלם (Perfect Binary Tree)

עץ בינארי מושלם הוא עץ בינארי כך שכל העלים נמצאים באותו מרחק מהשורש וכל הקודקודים הפנמיים בעלי 2 ילדים

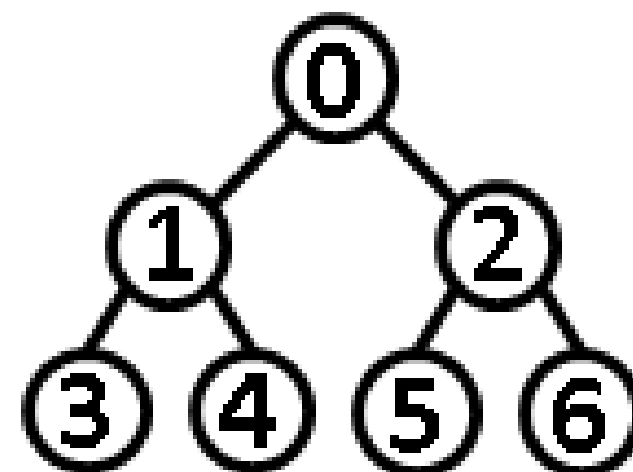




**full
binary tree**



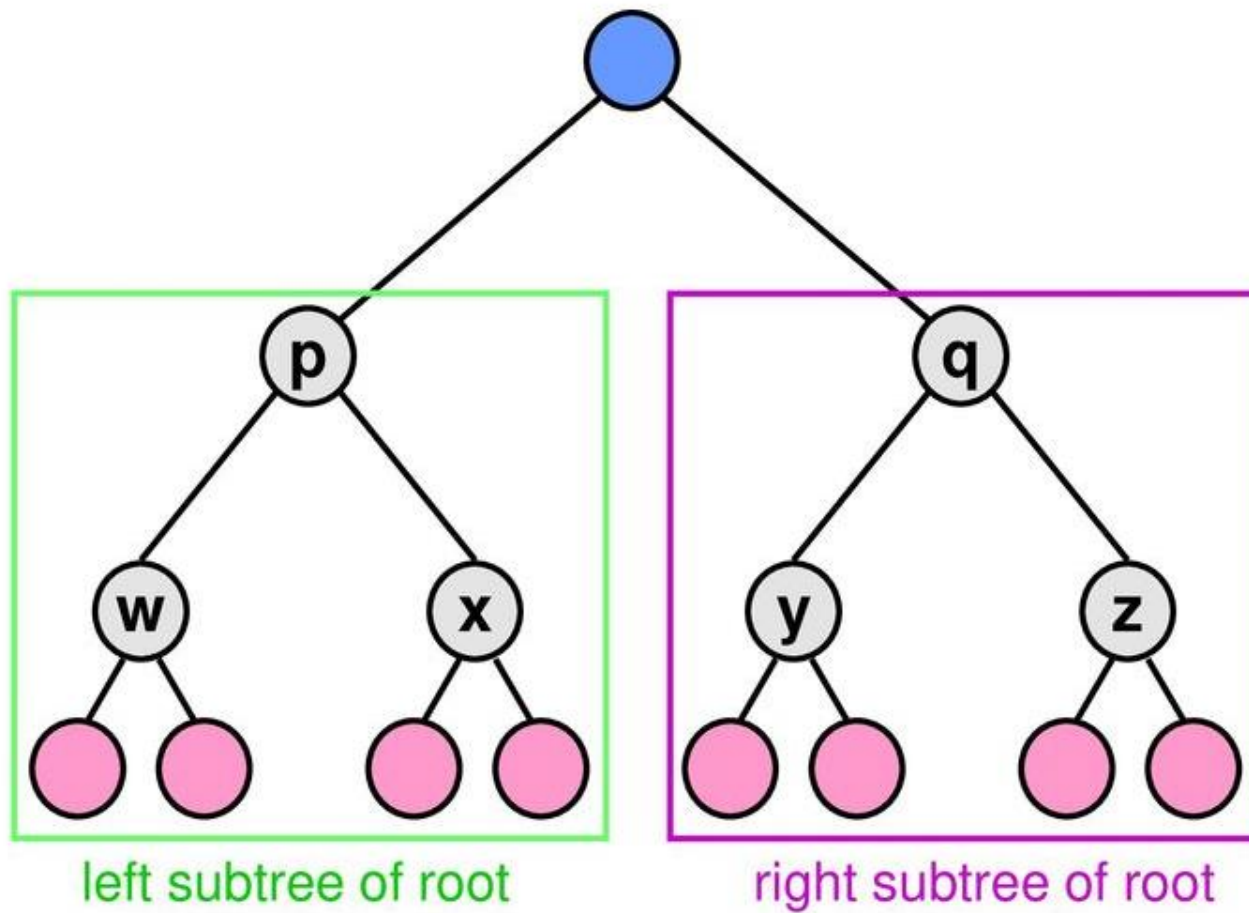
**complete
binary tree**



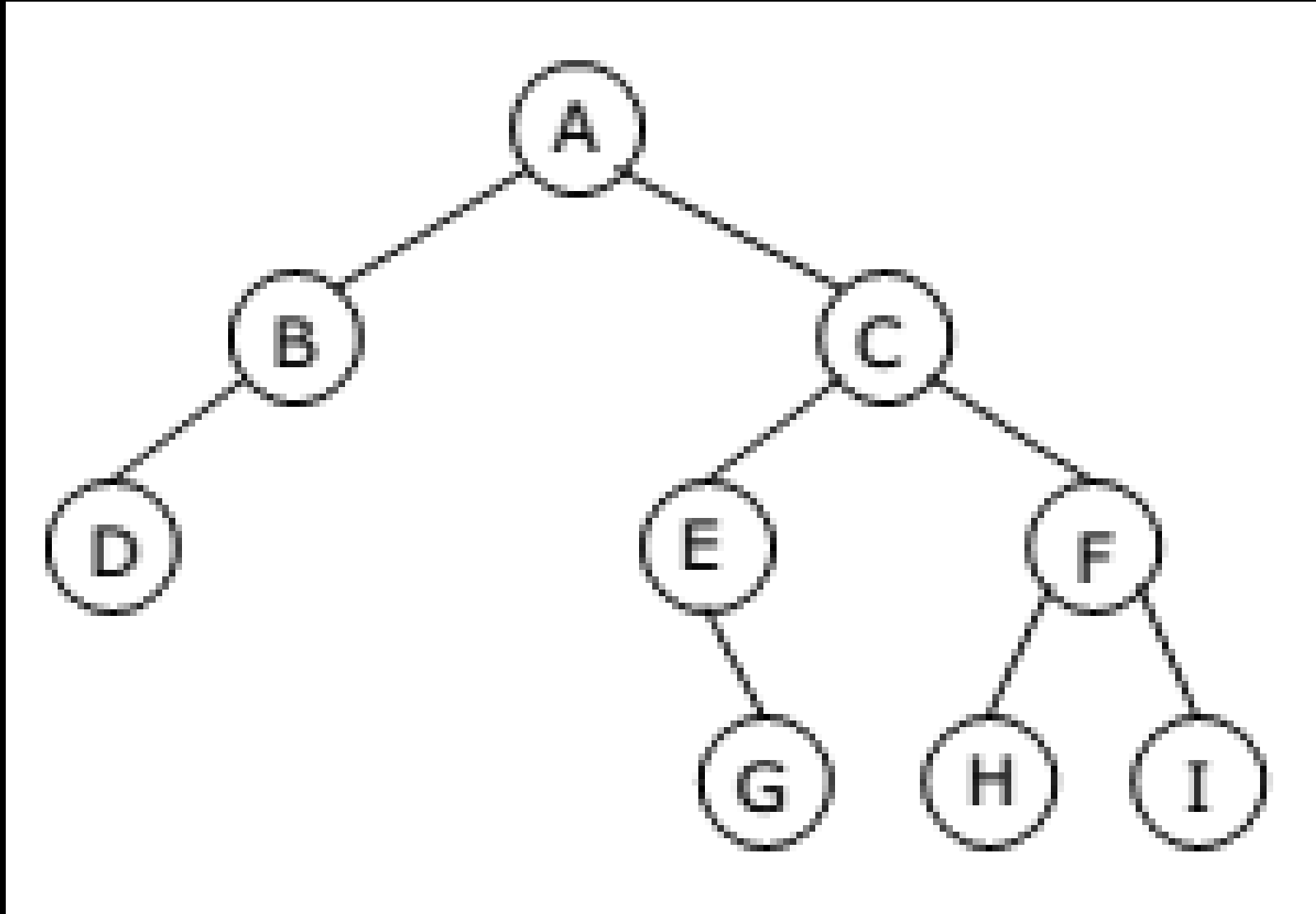
**perfect
binary tree**

עץ בינארי

כל עץ ניתן לפרוק ל**שורש**, תת-עץ **שמאלי** ותת-עץ **ימני**



שיטות מעבר על עץ בינארי tree traversal



שיטות מעבר על עץ בינארי tree traversal

- Preorder

עבור כל צומת יש להדפיס תחילה את ערכו, לאחר מכן לעבור לבנו השמאלי ולבסוף לבן הימני.

preorder tree traversal

```
public void PreOrder() // (V,L,R)
{
    PreOrder(root);
}
```

```
public void PreOrder(Node current) {
    if(current != null)
    {
        System.out.print(current.data + " ");
        PreOrder(current.left);
        PreOrder(current.right);
    }
}
```

שיטות מעבר על עץ בינארי tree traversal

- Preorder

עבור כל צומת יש להדפיס תחילה את ערכו, לאחר מכן לעבור לבנו השמאלי ולבסוף לבן הימני.

- Inorder

עבור כל צומת יש להדפיס תחילה את בן השמאלי, לאחר מכן לעבור לערכו שלו ואז הבן הימני

Inorder tree traversal

```
public void InOrder() // (L,V,R)
{
    InOrder(root);
}
```

```
public void InOrder(Node current) {
    if(current != null)
    {
        InOrder(current.left);
        System.out.print(current.data + " ");
        InOrder(current.right);
    }
}
```

שיטות מעבר על עץ בינארי tree traversal

- Preorder

עבור כל צומת יש להדפיס תחילה את ערכו, לאחר מכן לעבור לבנו השמאלי ולבסוף לבן הימני.

- Inorder

עבור כל צומת יש להדפיס תחילה את בן השמאלי, לאחר מכן לעבור לערכו שלו ואז הבן הימני

- Postorder

עבור כל צומת יש להדפיס תחילה את בנו השמאלי, אחר כך את בנו הימני ולבסוף

PostOrder tree traversal

```
public void PostOrder() // (L,R,V)
{
    PostOrder(root);
}
```

```
public void PostOrder(Node current) {
    if(current != null)
    {
        PostOrder(current.left);
        PostOrder(current.right);
        System.out.print(current.data + " " );
    }
}
```

שיטות מעבר על עץ בינארי tree traversal

- Preorder

עבור כל צומת יש להדפיס תחילה את ערכו, לאחר מכן לעבור לבנו השמאלי ולבסוף לבן הימני.

- Inorder

עבור כל צומת יש להדפיס תחילה את בן השמאלי, לאחר מכן לעבור לערכו שלו ואז הבן הימני

- Postorder

עבור כל צומת יש להדפיס תחילה את בנו השמאלי, אחר כך את בנו הימני ולבסוף

...

שיטות מעבר על עץ בינארי tree traversal

- Preorder

עבור כל צומת יש להדפיס תחילה את ערכו, לאחר מכן לעבור לבנו השמאלי ולבסוף לבן הימני.

- Inorder

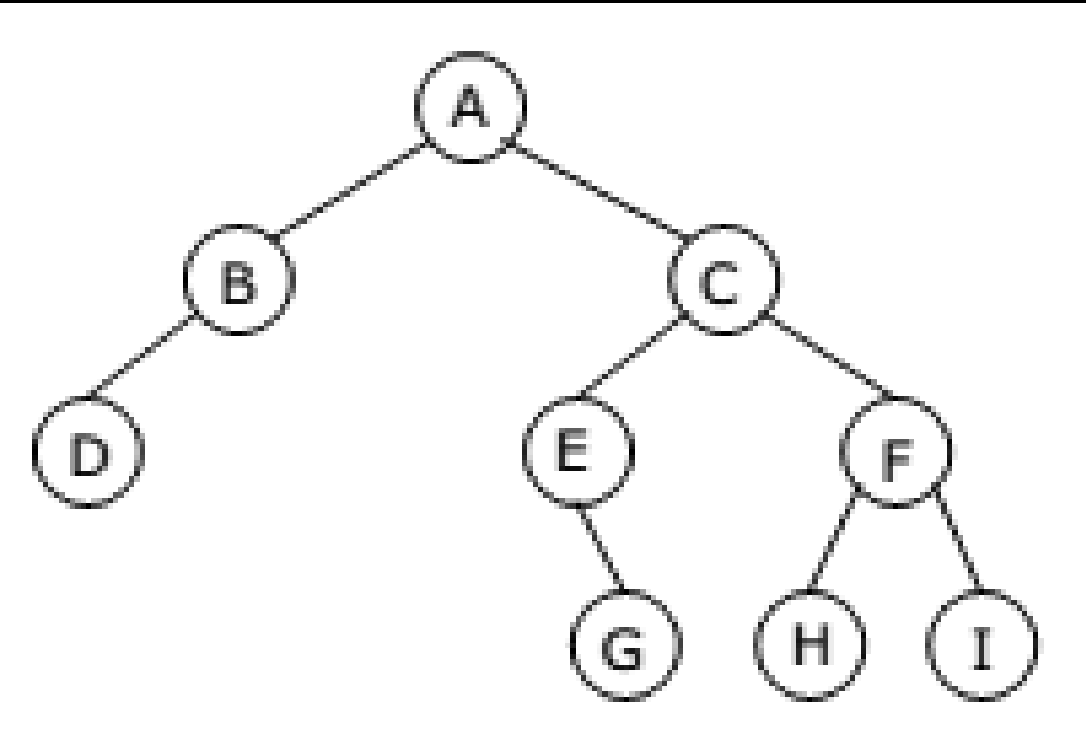
עבור כל צומת יש להדפיס תחילה את בן השמאלי, לאחר מכן לעבור לערכו שלו ואז הבן הימני

- Postorder

עבור כל צומת יש להדפיס תחילה את בנו השמאלי, אחר כך את בנו הימני ולבסוף

...

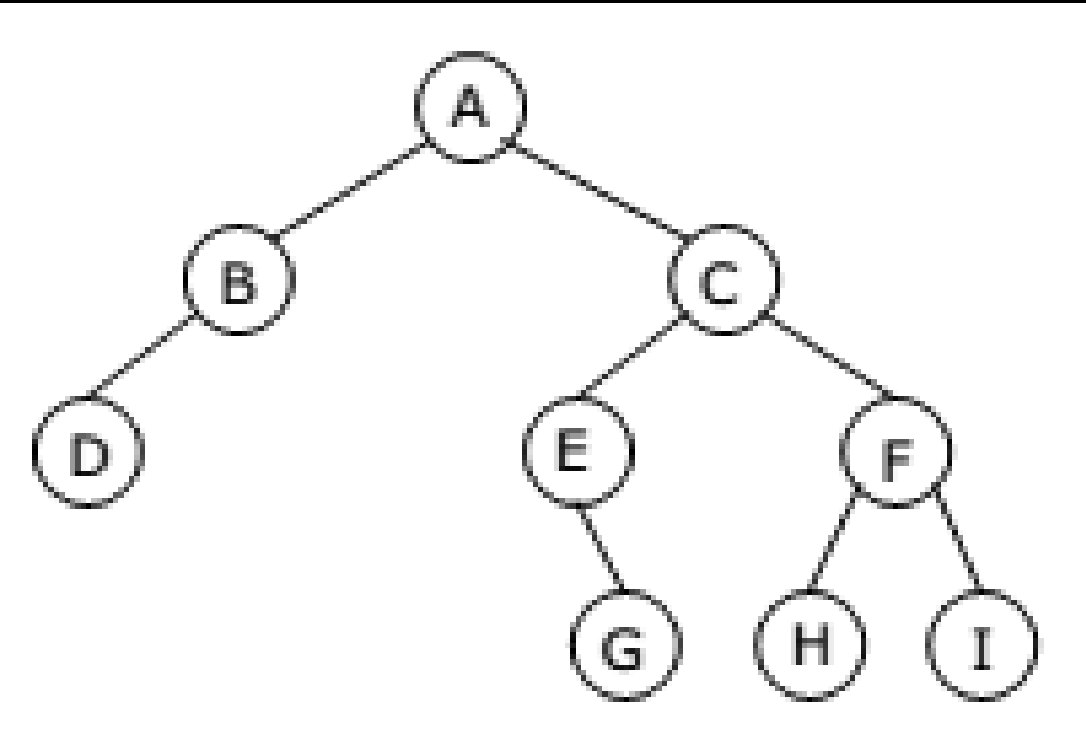
PreOrder



- Preorder traversal yields:
A, B, D, C, E, G, F, H, I

```
public void PreOrder(Node current) {  
    if(current != null)  
    {  
        System.out.print(current.data + " ");  
        PreOrder(current.left);  
        PreOrder(current.right);  
    }  
}
```

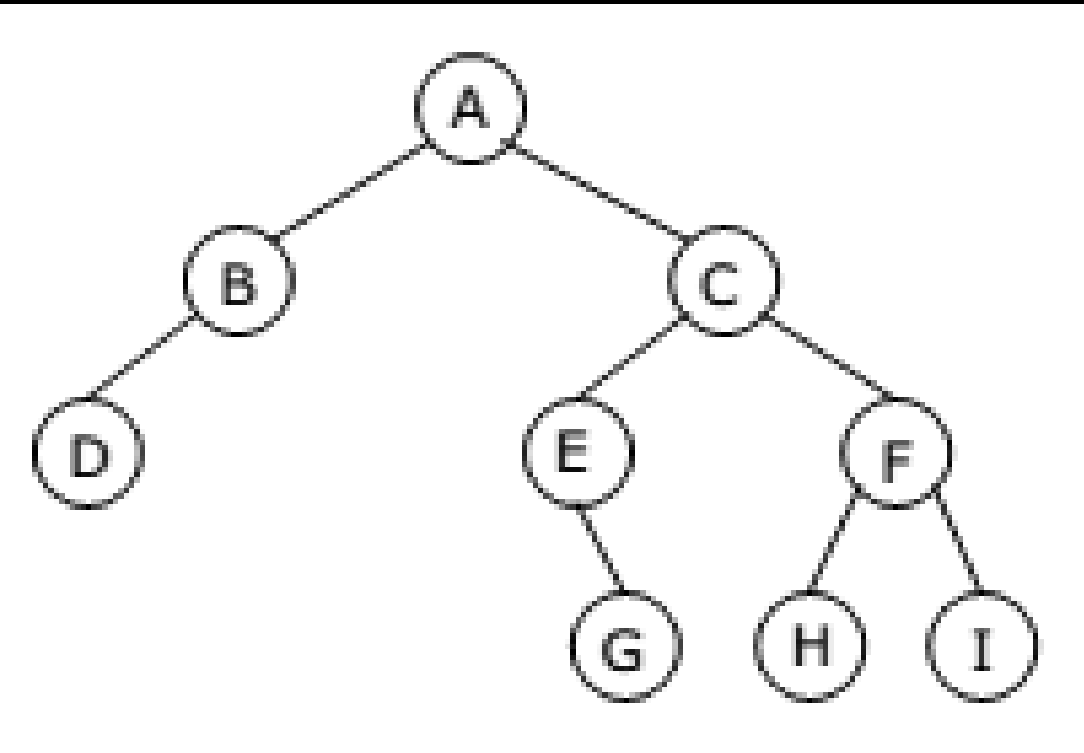

PostOrder



- Preorder traversal yields:
A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
D, B, G, E, H, I, F, C, A

```
public void PostOrder(Node current) {  
    if(current != null)  
    {  
        PostOrder(current.left);  
        PostOrder(current.right);  
        System.out.print(current.data + " ");  
    }  
}
```

InOrder



- Preorder traversal yields: A, B, D, C, E, G, F, H, I
- Postorder traversal yields: D, B, G, E, H, I, F, C, A
- Inorder traversal yields: D, B, A, E, G, C, H, F, I

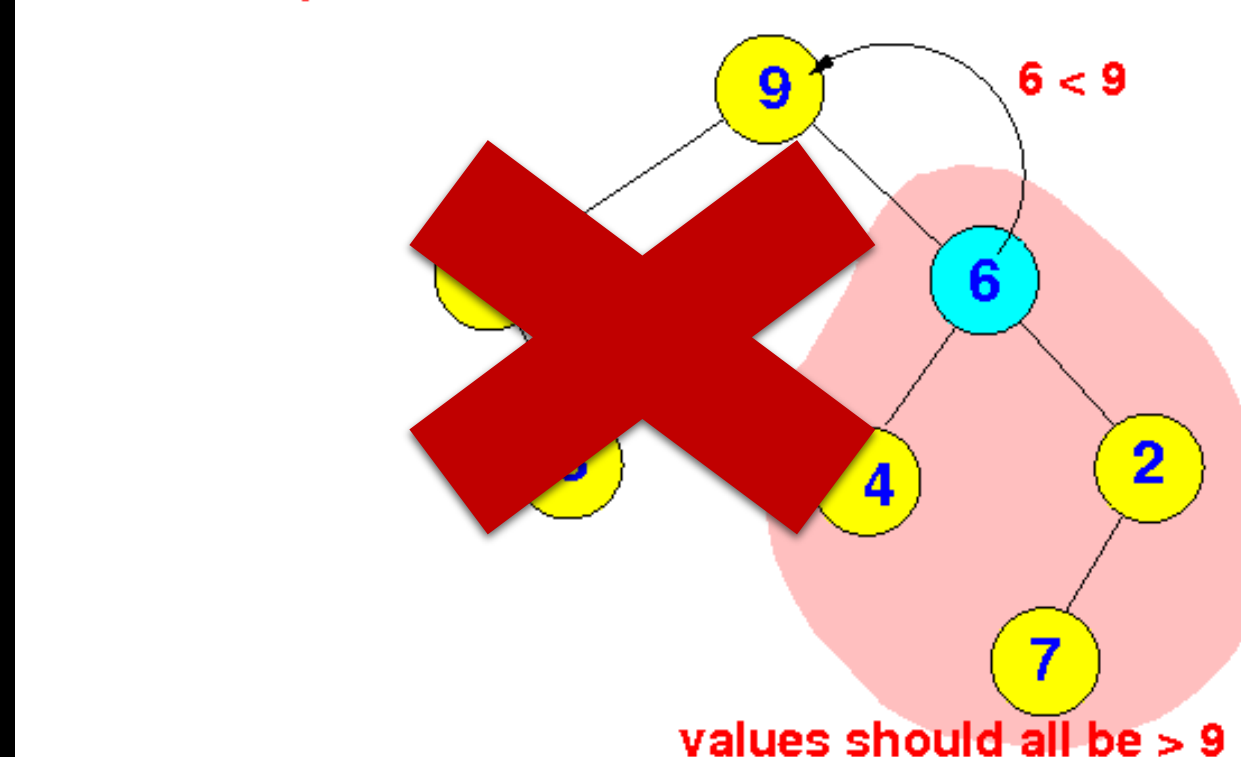
```
public void InOrder(Node current) {  
    if(current != null)  
    {  
        InOrder(current.left);  
        System.out.print(current.data + " ");  
        InOrder(current.right);  
    }  
}
```

עץ חיפוש בינארי (BST)

עץ חיפוש בינארי הוא עץ בינארי כך שעבור כל קודקוד x מתקיים כי :

- הערכים של כל הקודקודים בתת עץ השמאלי של x קטנים ממש מ- x
- הערכים של כל הקודקודים בתת עץ הימני של x גדולים ממש מ- x

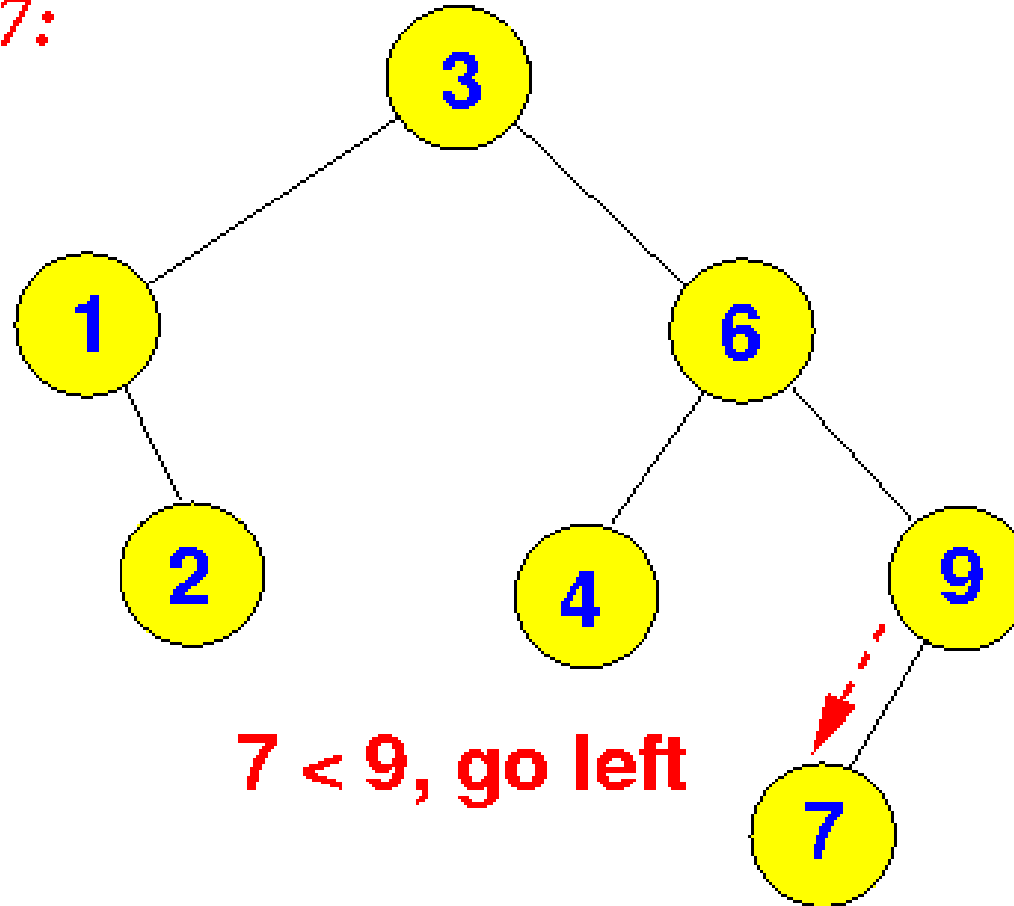
Not a Binary Search tree:



חיפוש Search

נרצה לחפש את 7 בעץ חיפוש בינארי

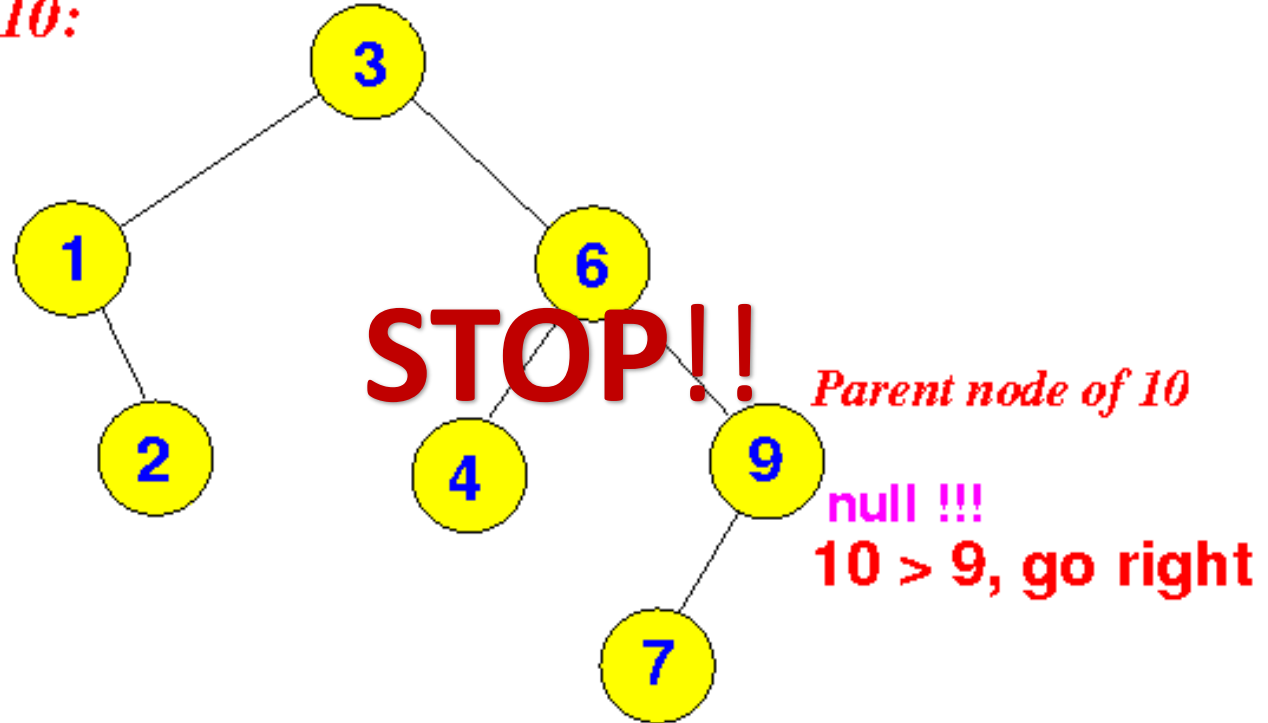
Find 7:



חיפוש Search

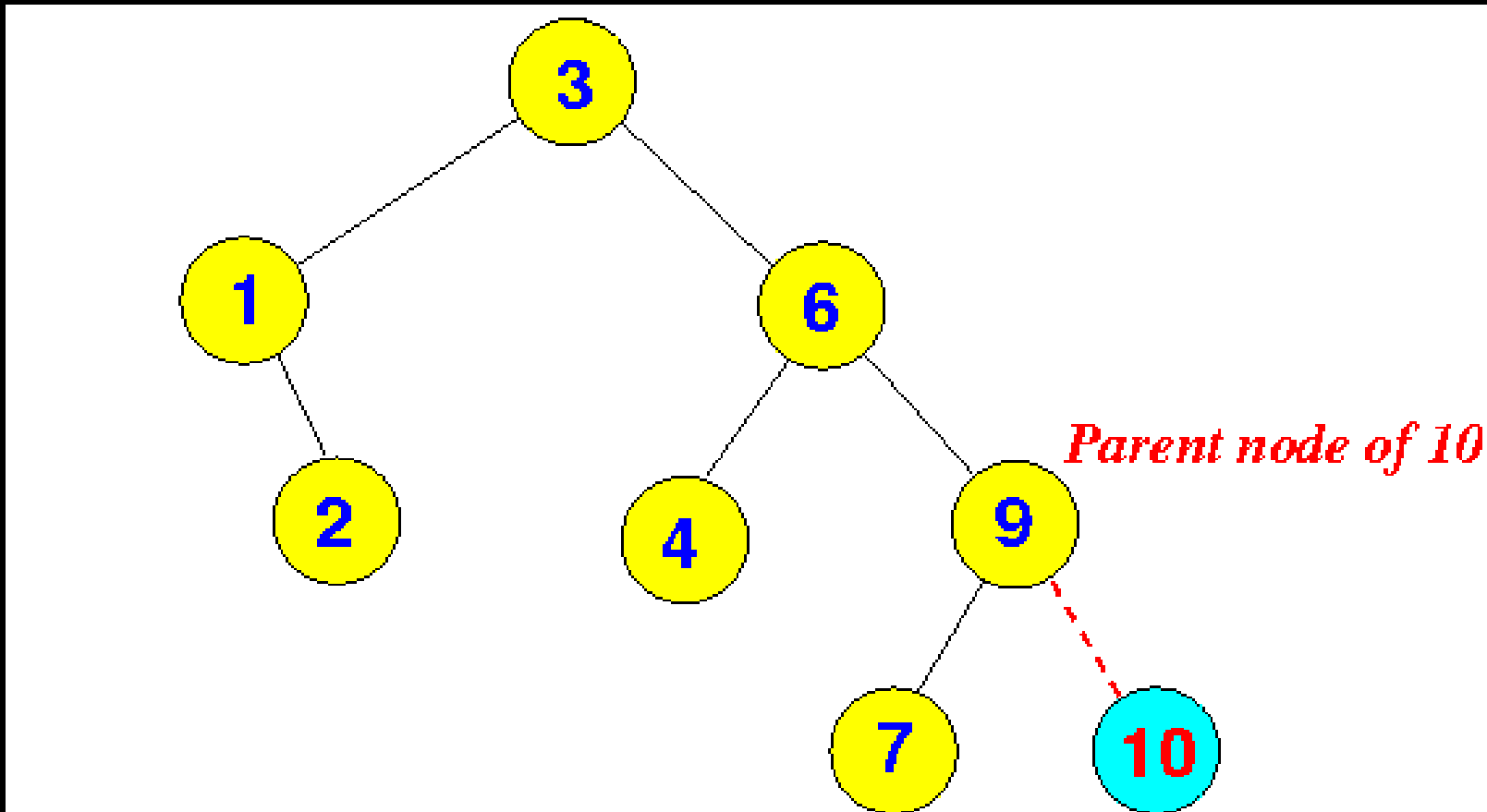
נרצה לחפש את 10 בעץ חיפוש בינארי

Find 10:



הכנסה Insert

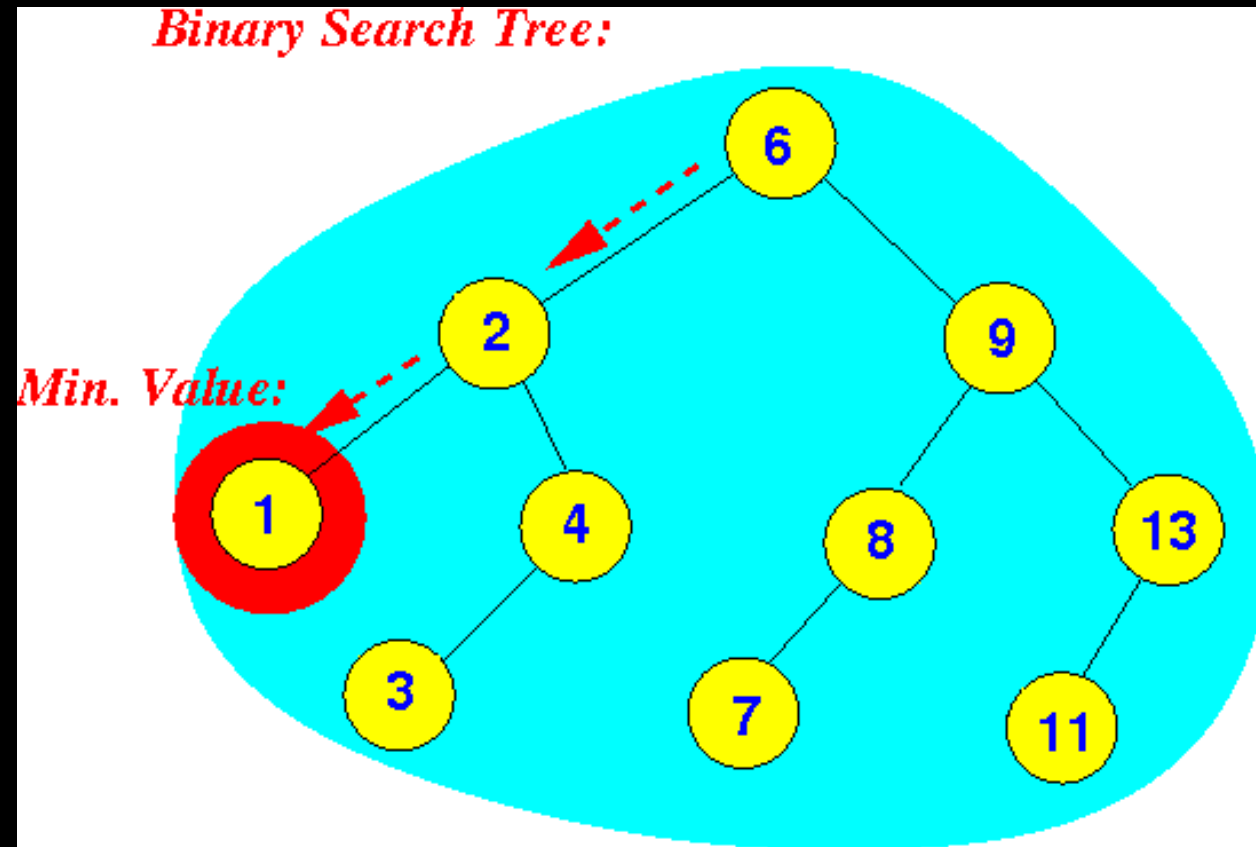
נרצה להכניס את 10 בעץ חיפוש בינארי



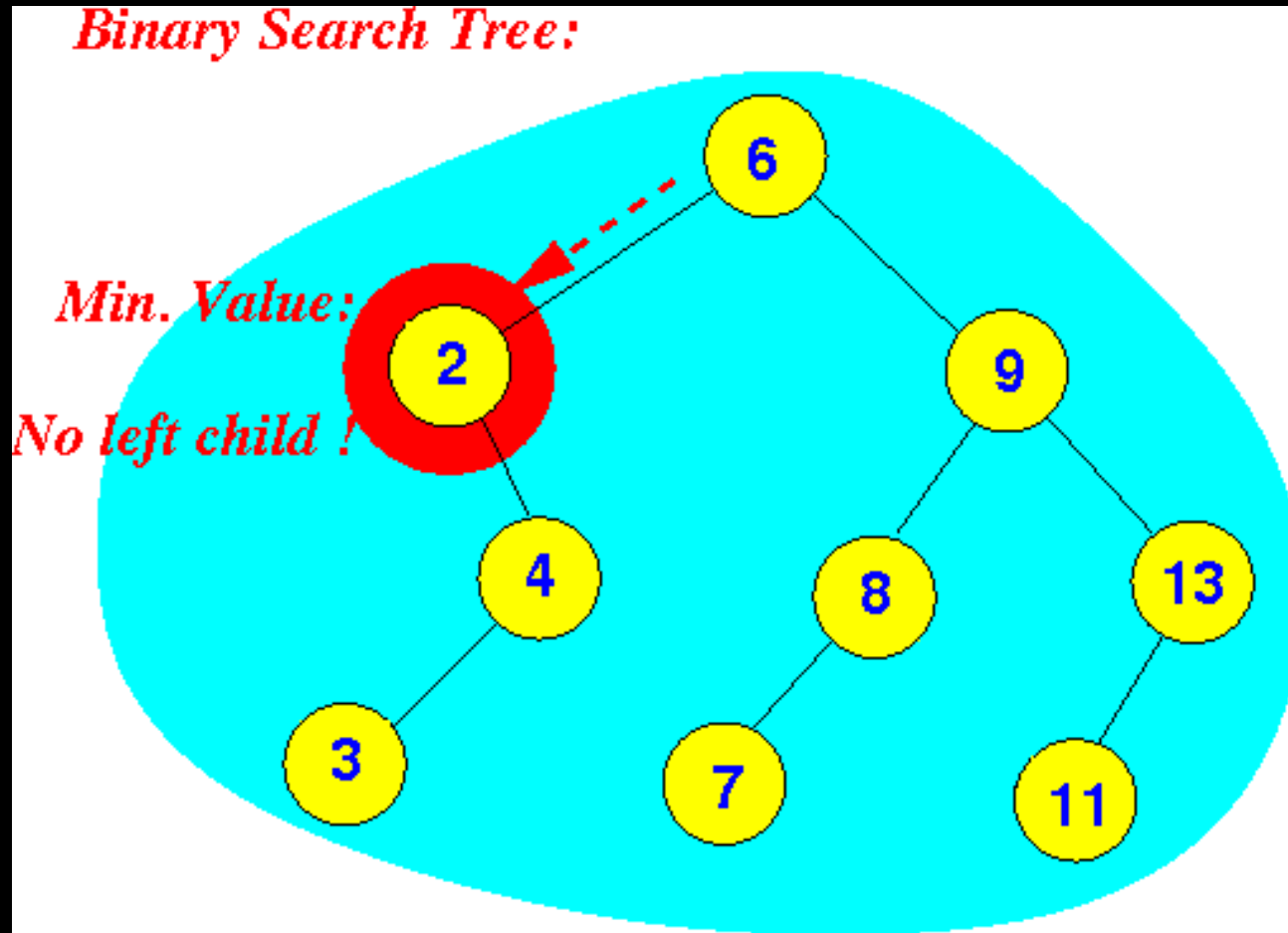
הכנסה Insert

www.mathwarehouse.com

איבר מינמלי

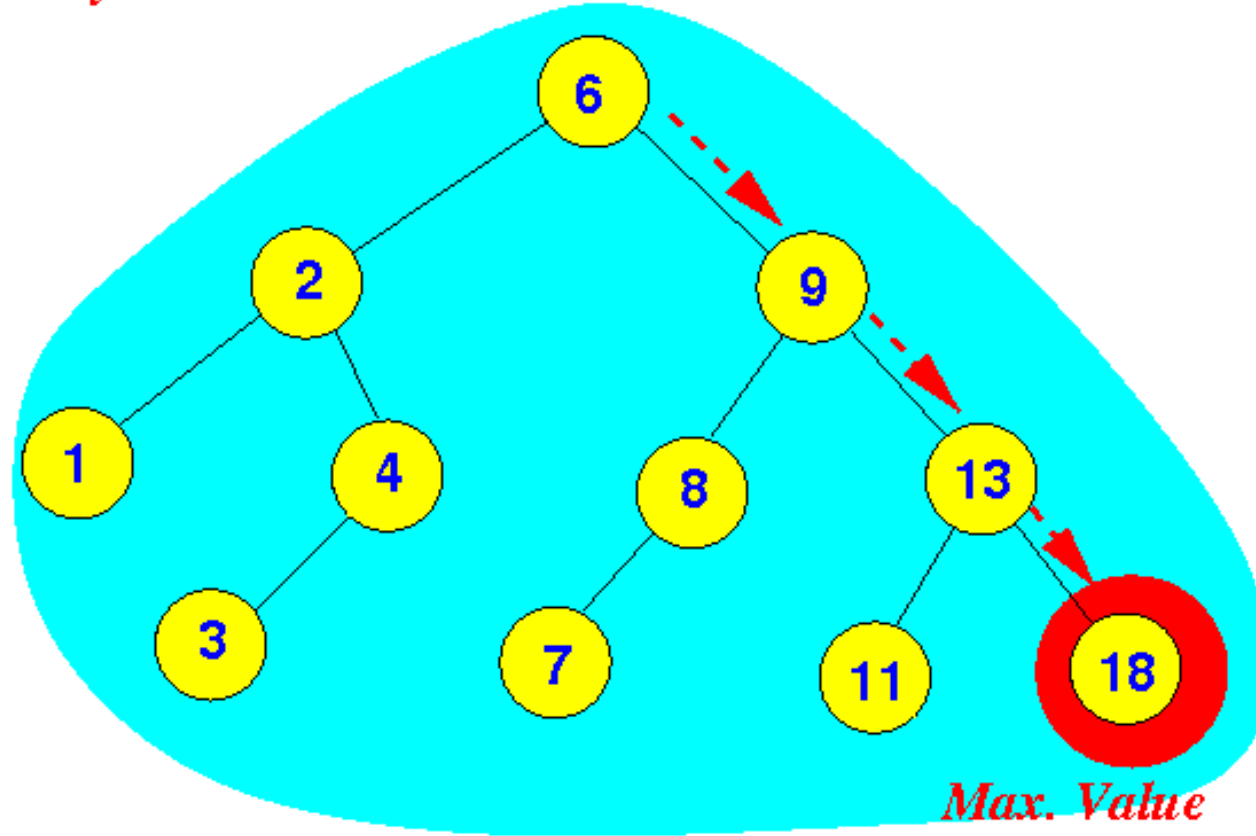


איבר מינמלי



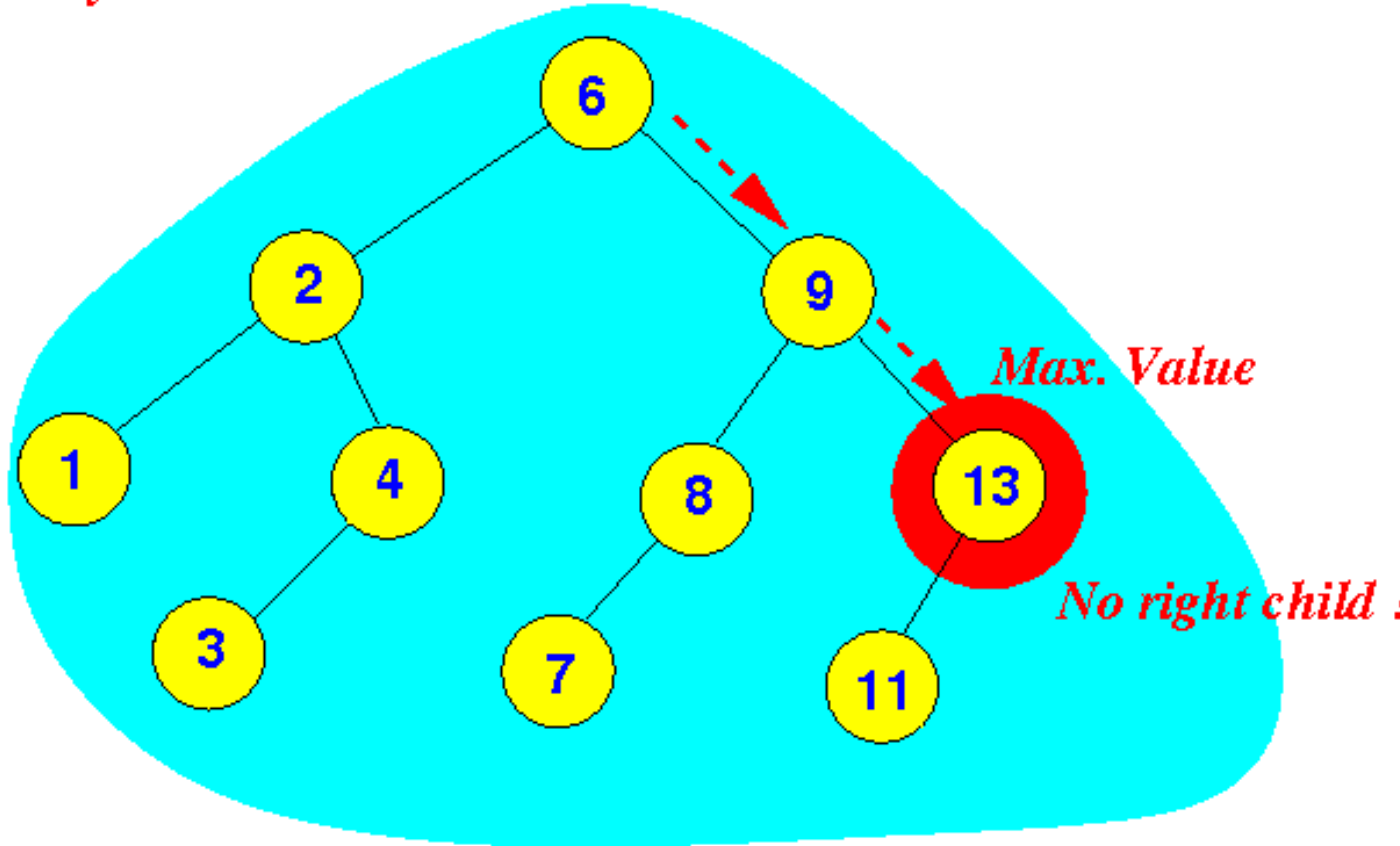
איבר מקסמלי

Binary Search Tree:

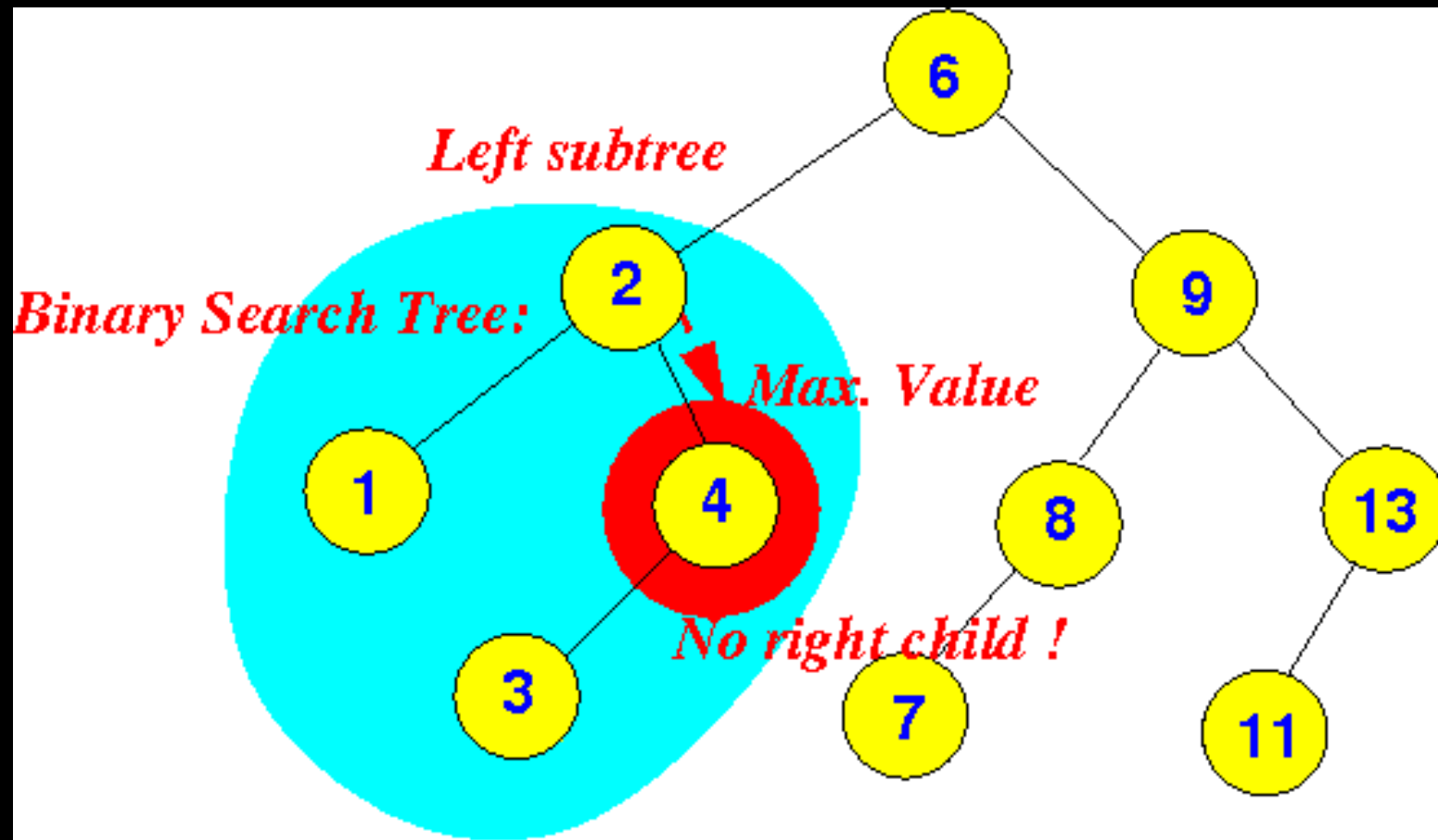


איבר מקסמלי

Binary Search Tree:

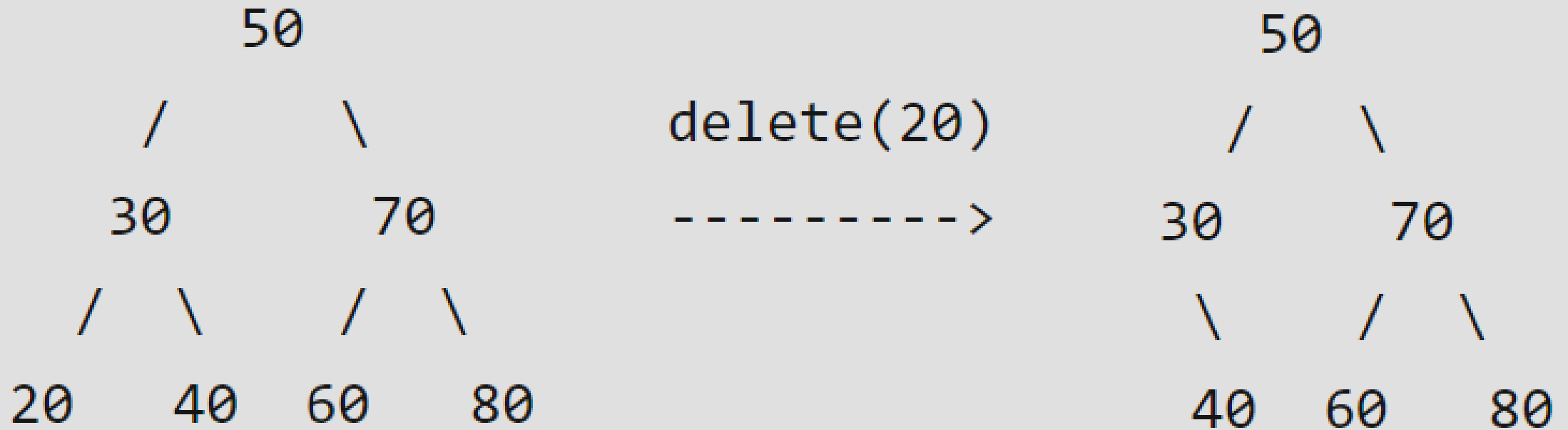


איבר מקסמלי בתת עץ



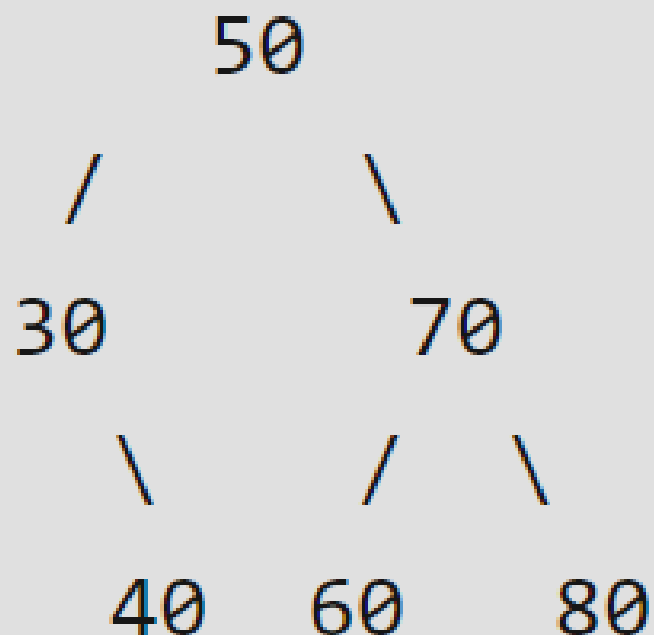
מחיקה remove

מקרה 1: לקודקוד אין בנים



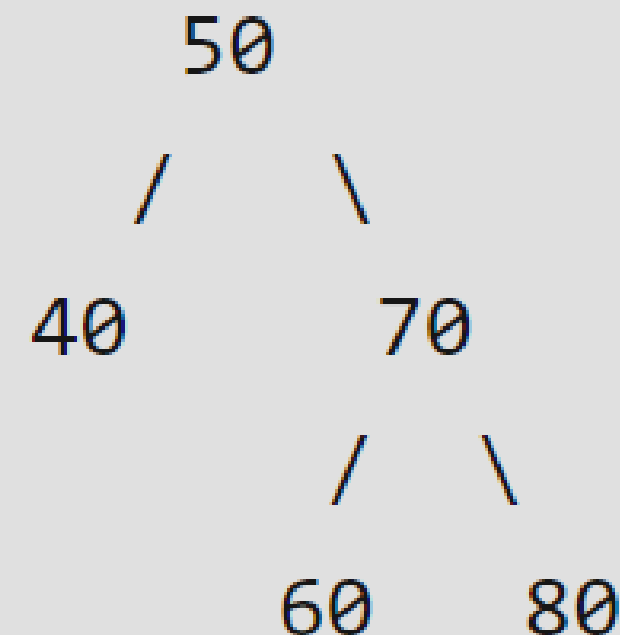
מחיקה remove

מקרה 2: לקודקוד בעל בן יחיד



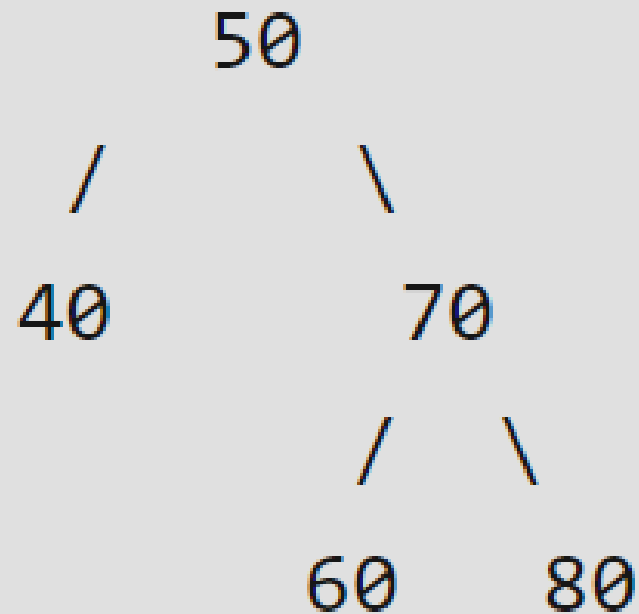
delete(30)

----->

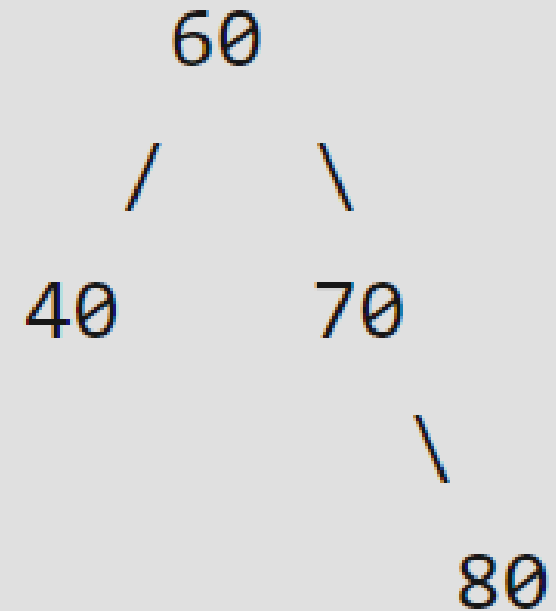


מחיקה remove

מקרה 3: לקודקוד 2 בנים



delete(50)
----->



תכונות של עץ חיפוש בינארי

תכונות של עץ חיפוש בינארי

מספר הכולל של הקודקודים בעץ בינארי מושלם הינו $n = 2^{h+1} - 1$
כאשר h הוא גובה העץ

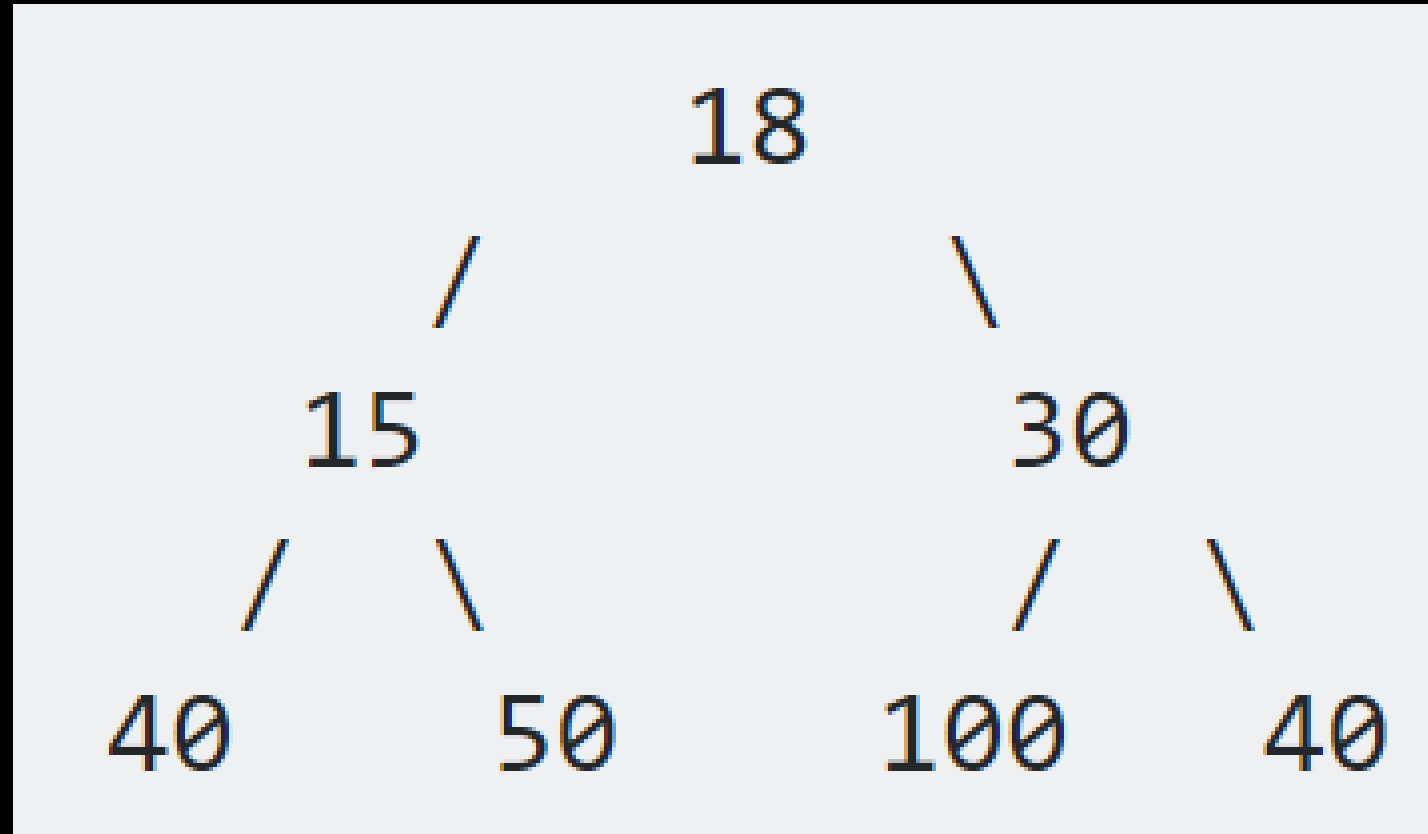
18

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h = \frac{1(2^{h+1} - 1)}{2 - 1} = 2^{h+1} - 1$$

/ \ / \
40 50 100 40

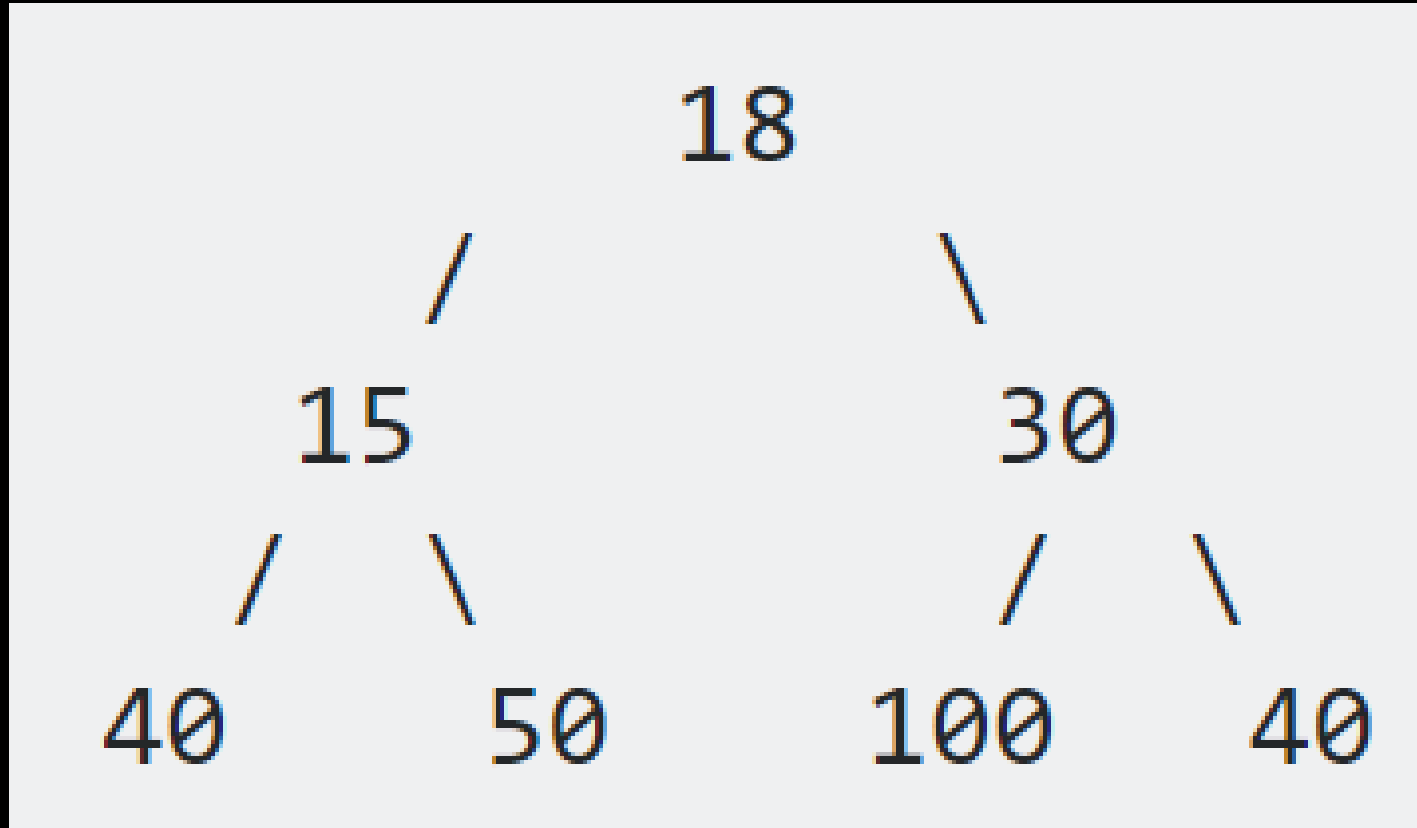
תכונות של עץ חיפוש בינארי

מספר העלים בעץ בינארי מושלם (Perfect) הינו 2^h



תכונות של עץ חיפוש בינארי

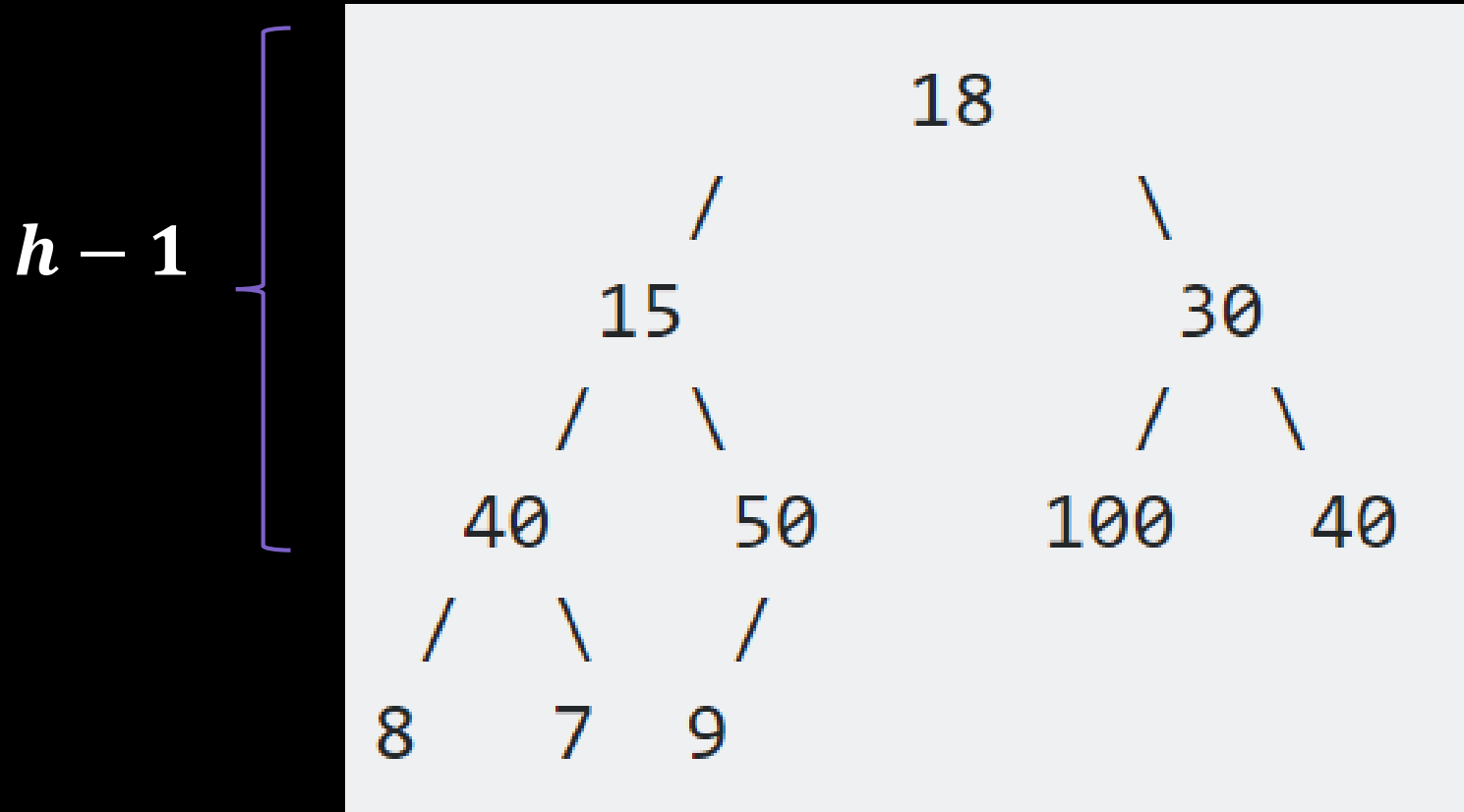
מספר הצמתים הפנמיים בעץ בינארי מושלם (Perfect) הינו $2^h - 1$



תכונות של עץ חיפוש בינארי

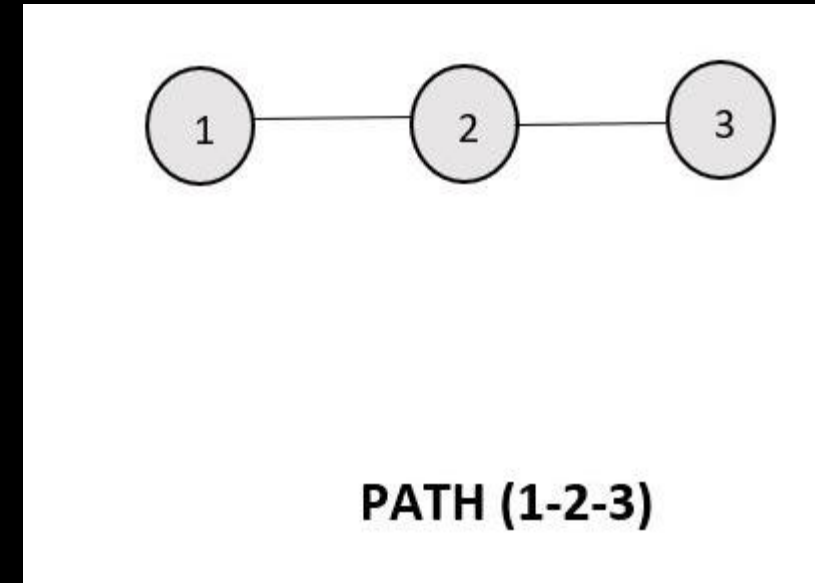
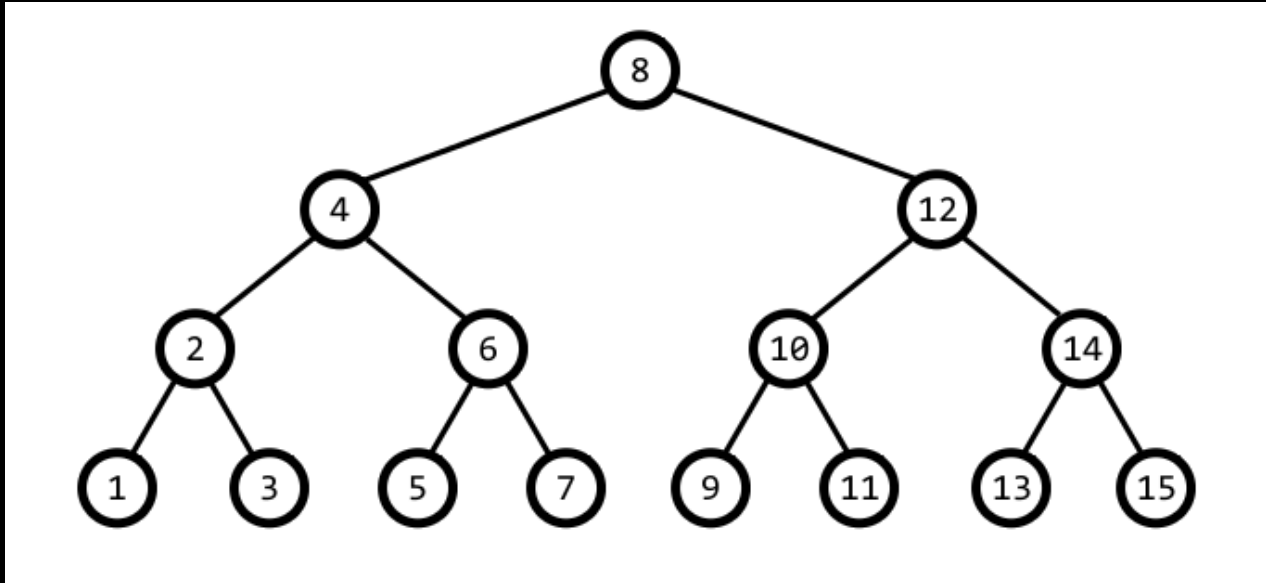
מספר הצמתים הכולל n בעץ בינארי שלם (Complete)

הוא בין 2^h לבין $2^{h+1} - 1$



תכונות של עץ חיפוש בינארי

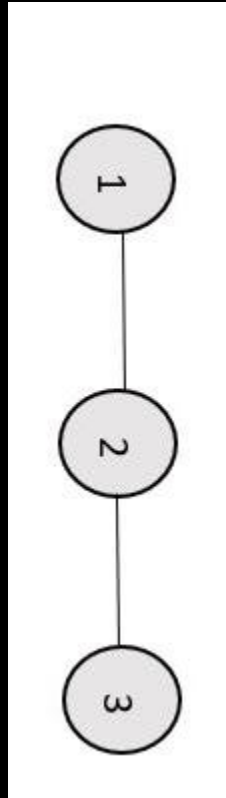
גובה של עץ חיפוש בינארי הינו במקרה הטוב $O(\log_2(n))$
ובמקרה הרע $O(n)$



תכונות של עץ חיפוש בינארי

מספר הקודקודים המינמלי בעץ בינארי עם גובה h

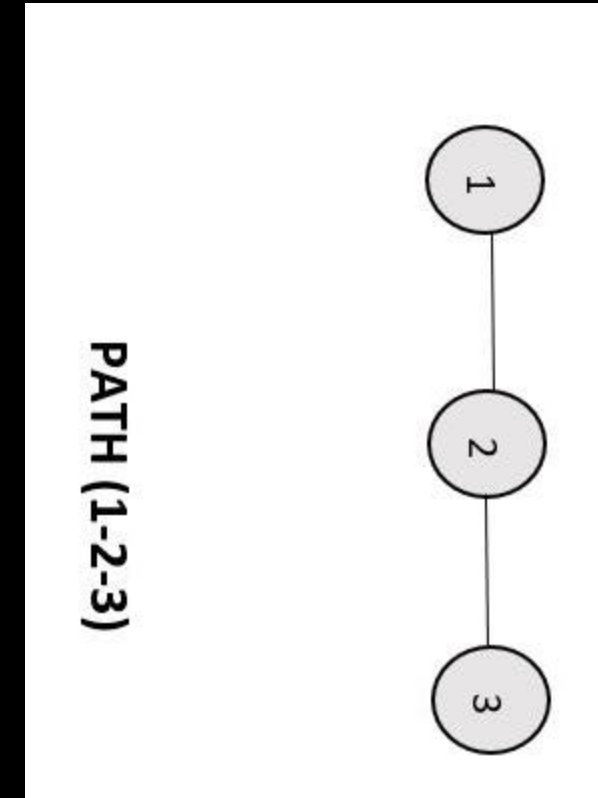
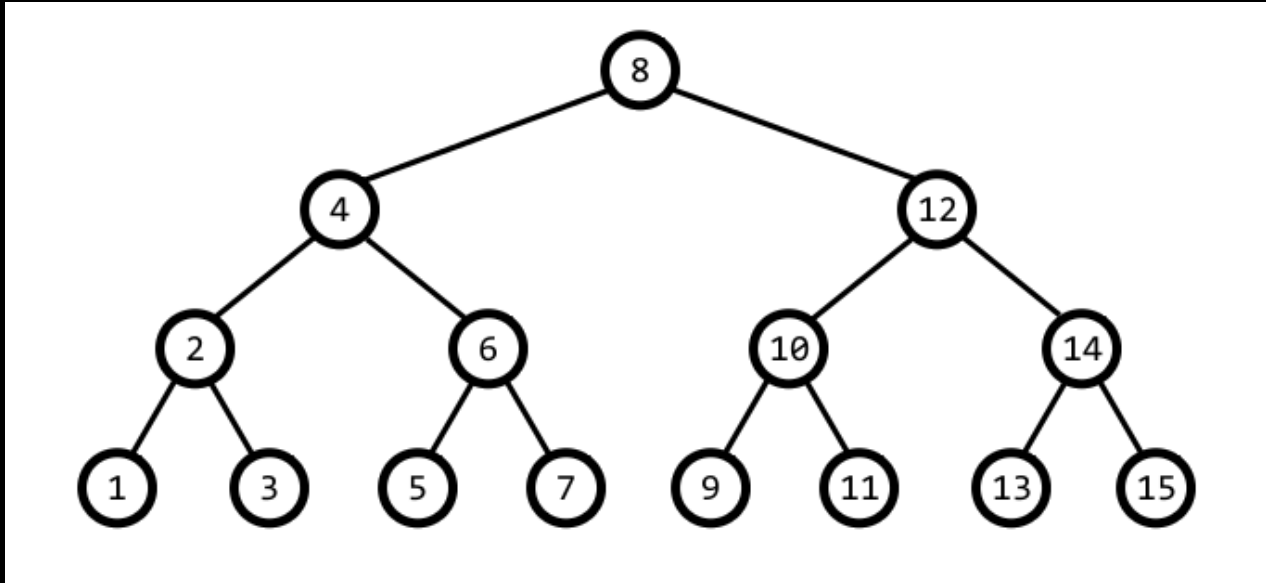
הינו $h + 1$



תכונות של עץ חיפוש בינארי

כמות הצמתים הפנמיים בעץ בינארי

$$\text{בקטע } [h, 2^h - 1]$$



סיבוכיות BST

Data structure	Access /peek	Search	Insert /push	Delete /pop	Traverse
Linear					
Array	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Ordered array	$O(1)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$
Linked list	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Ordered linked list	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Matrix	$O(1)$	$O(n^2)$	$O(1)$	$O(n^2)$	$O(n^2)$
Stack	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Queue	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Non-Linear					
Tree (worst case)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Tree (balanced)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$

Improving a BST can be done by making it balanced - like [AVL](#) or [red-black-trees](#).

visualization

Binary Search Tree



Animation Completed

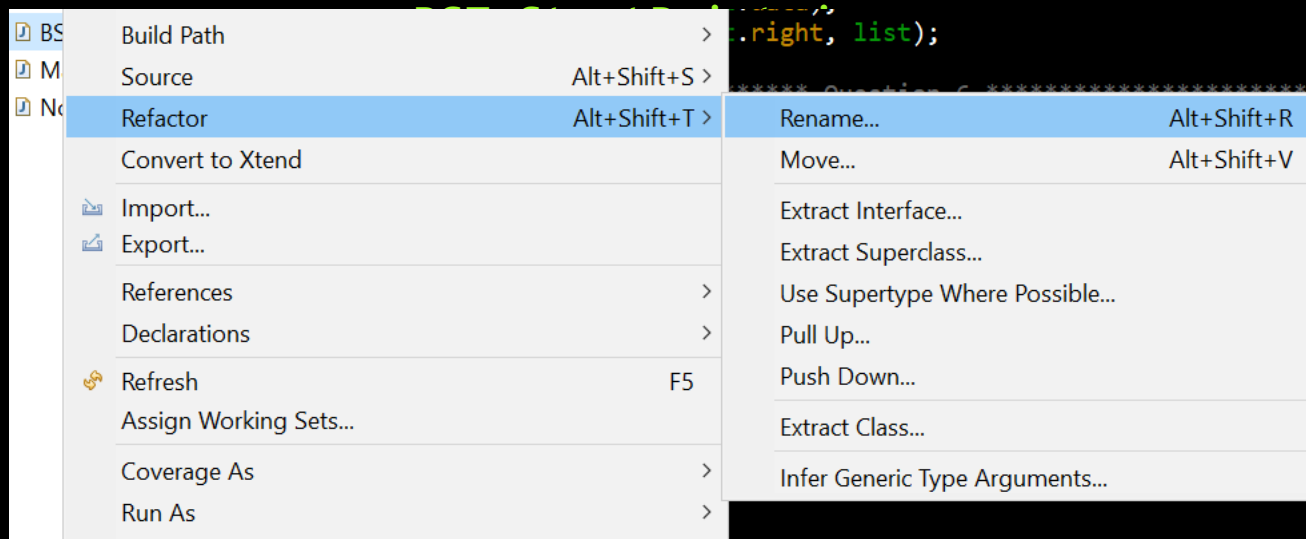
w: h:

Animation Speed

Algorithm Visualizations

<https://www.cs.usfca.edu/~galles/visualization/BST.html>

מימוש BST



Node.java חימוש

```
/* Class containing left and right  
child of current node and key value*/  
public class Node {  
    int data;  
    Node right;  
    Node left;  
    public Node(int data) {  
        this.data = data;  
        left = right = null;  
    }  
}
```

מימוש Main.java

```
public class Main {  
    public static void main(String[] args) {
```

```
        /----- 9  
5  
        \----- 4  
            |           /----- 2  
            \----- 1
```

```
    }
```

תרגילים מהמודל

תרגיל 10 לעבודה עצמית עץ חיפוש בינארי



שאלה 6

הוסיפו למחלקת `BinarySearchTree` שיטה **רקורסיבית** המחשבת מספר צמתי העץ:

```
public int size(){ . . . }
```

שאלה 7

כתוב פונקציה שמקבלת נתון הנמצא בצומת ומחזירה את מספר הבניו:

```
public int numOfChilds(Object data){ . . . }
```

שאלה 8

כתוב פונקציה `toString()` שמחזירה מחרוזת המכילה את כל איברי העץ:

```
public String toString(){ . . . }
```

שאלה 9

כתוב פונקציה `numOfLeaves()` שמחזירה מספר עלים בעץ:

```
public int numOfLeaves(){ . . . }
```

שאלה 10

כתוב פונקציה `numOfParents()` שמחזירה מספר צמתים שיש להם לפחות בן אחד:

```
public int numOfParents(){ . . . }
```

שאלה 11

כתוב בנאי מעתיק לעץ חיפוש בינארי

```
public BinarySearchTree (BinarySearchTree bst){ . . . }
```

תרגיל 9 לעבודה עצמית עצים בינאריים



שאלה 2

הוסיפו למחלקת `BinaryTree` שיטה שמדפיסה צמתי העץ בצורת **Inorder**

```
public void printInorder(){ . . . }
```

שאלה 3

הוסיפו למחלקת `BinaryTree` שיטה שמדפיסה צמתי העץ בצורת **Postorder**

```
public void printPostorder (){ . . . }
```

שאלה 4

הוסיפו למחלקת `BinaryTree` שיטה המחשבת מספר צמתי העץ:

```
public int size(){ . . . }
```

<https://moodlelearn.ariel.ac.il/course/view.php?id=70393>

מימוש BST.java

```
public class BST  
{
```

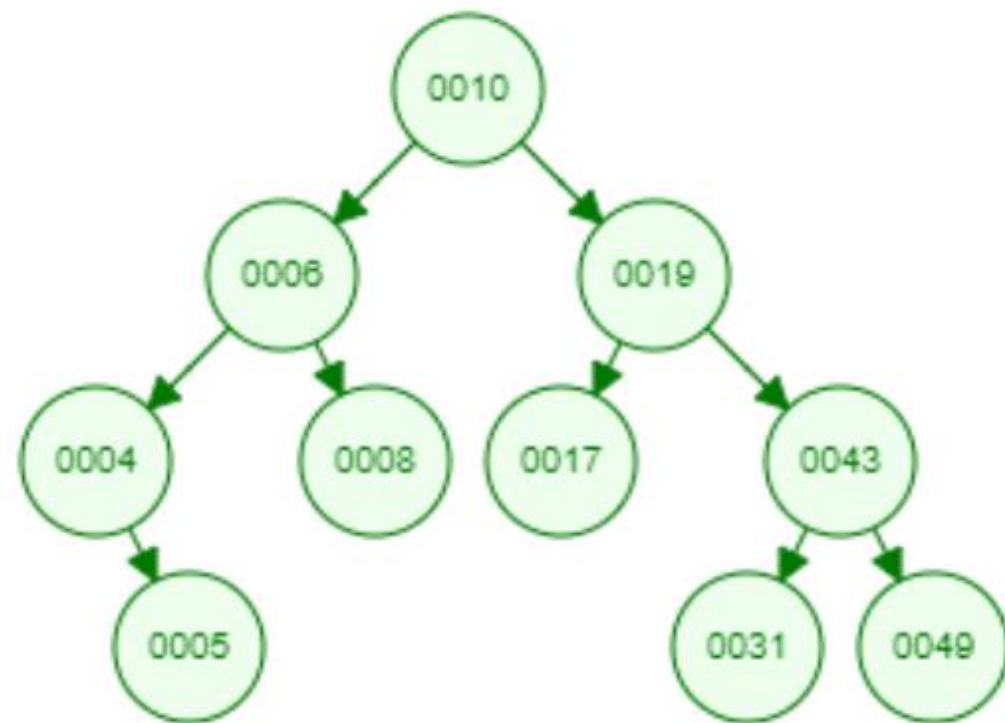
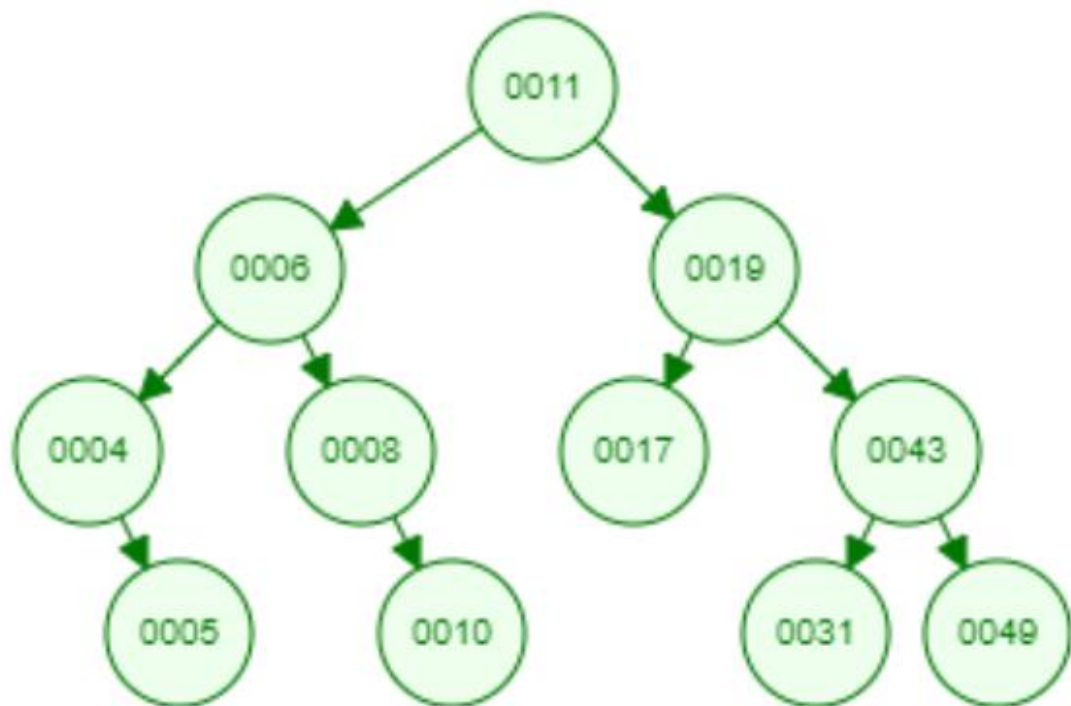


```
...  
}
```

שאלות

שאלה 5

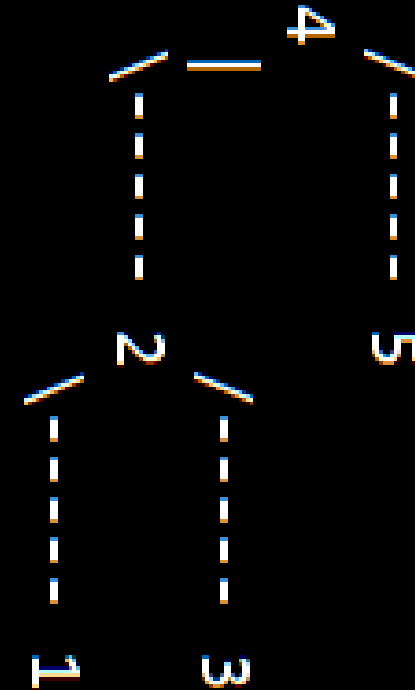
נתונה סדרת המספרים: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31. צייר עץ חיפוש בינארי כתוצאה של הוספת המספרים לעץ חיפוש בינארי ריק. צייר שני עצים כתוצאת המחיקה של מספר 11.



כתבו תוכנית אשר בודקת אם עץ בינארי הוא BST או לא

```
boolean b = tree.isBST();
```

```
public static void main(String[] args) {  
    BST tree = new BST();  
    tree.root = new Node(4);  
    tree.root.left = new Node(2);  
    tree.root.right = new Node(5);  
    tree.root.left.left = new Node(1);  
    tree.root.left.right = new Node(3);  
}
```



```
boolean b = tree.isBST();  
System.out.println( (b == true) ? "V" : "X" );
```

```
}
```

כתבו תוכנית אשר בודקת אם עץ בינארי הוא BST או לא

הצעה:

```
public boolean isBST() {  
    return isBST(root);  
}
```

```
private boolean isBST(Node curr) {  
    if ( curr == null ) return true;
```

```
    /* false if left is > than node */
```

```
    if(curr.left != null && curr.left.data > curr.data)  
        return false;
```

```
    /* false if right is < than node */
```

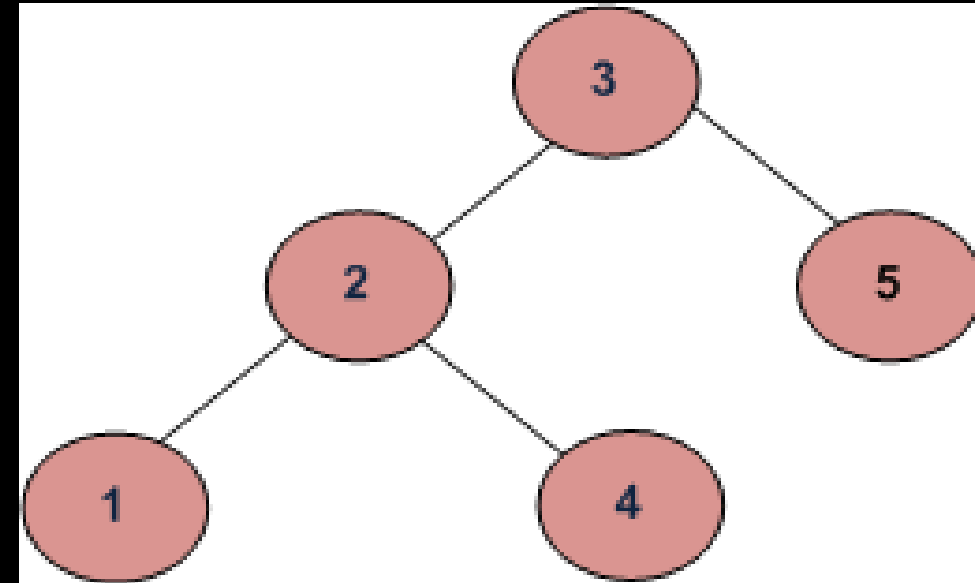
```
    if(curr.right != null && curr.right.data < curr.data)  
        return false;
```

```
    /* false if, recursively, the left or right is not a BST */
```

```
    if(!isBST(curr.left) || !isBST(curr.right)) return false;  
    else return true;
```

```
}
```

wrong



כתבו תוכנית אשר בודקת אם עץ בינארי הוא BST או לא

הצעה 2:

```
public boolean isBST() {  
    return isBST(root);  
}
```

```
private boolean isBST(Node curr) {  
    if ( curr == null ) return true;
```

```
    /* false if left is > than node */  
    if( curr.left != null && maxValue(curr.left) > curr.data)  
        return false;
```

```
    /* false if right is < than node */  
    if( curr.right != null && minValue(curr.right) < curr.data)  
        return false;
```

```
    /* false if, recursively, the left or right is not a BST */  
    if(!isBST(curr.left) || !isBST(curr.right)) return false;  
    else return true;
```

```
}
```

Correct but
not efficient

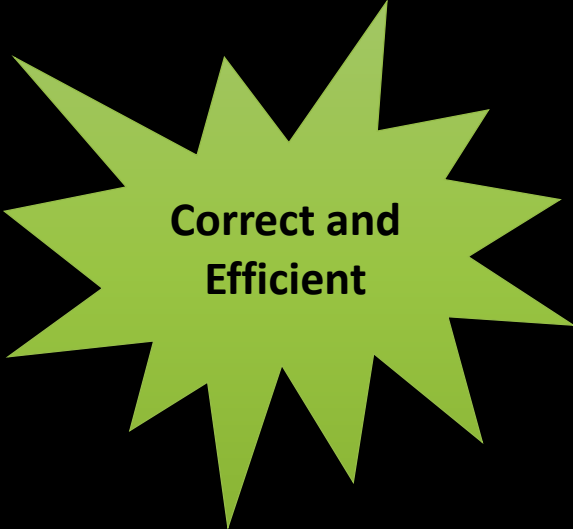
```
private int minValue(Node current) {  
    while (current.left != null)  
        current = current.left;  
    return current.data;  
}
```

```
private int maxValue(Node current) {  
    while (current.right != null)  
        current = current.right;  
    return current.data;  
}
```

כתבו תוכנית אשר בודקת אם עץ בינארי הוא BST או לא

הצעה 3:

```
public boolean isBST() {  
    return isBST(root,Integer.MIN_VALUE, Integer.MAX_VALUE);  
}  
private boolean isBST(Node curr, int min, int max) {  
    /* an empty tree is BST */  
    if (curr == null)  
        return true;  
  
    /* false if this node violates the min/max constraints */  
    if (curr.data < min || curr.data > max)  
        return false;  
  
    /* otherwise check the subtrees recursively  
    tightening the min/max constraints */  
    // Allow only distinct values  
    return (isBST(curr.left, min, curr.data-1) &&  
            isBST(curr.right, curr.data+1, max));  
}
```



Correct and
Efficient

Time Complexity: $O(n)$

Auxiliary Space : $O(1)$

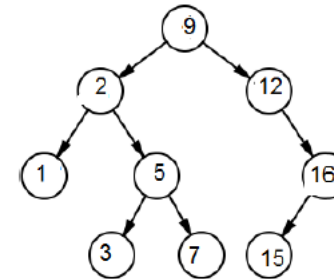
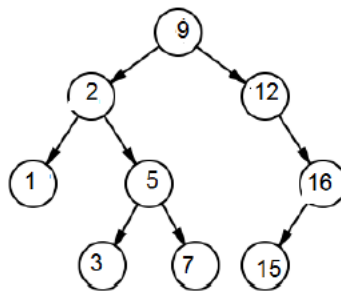
if Function Call Stack size is not considered,
otherwise $O(n)$

כתבו תוכנית שבודקת אם שני עצים בינאריים זהים

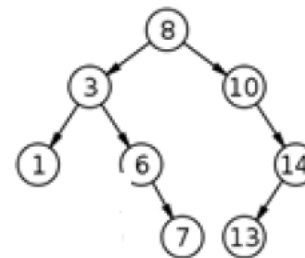
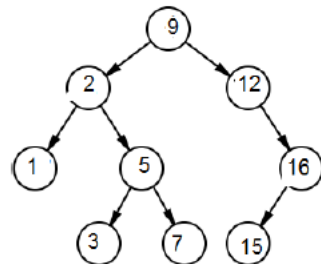
שאלה 4 (20 נקודות)

שני עצים חיפוש בינאריים הם שווים (equals) אם רק ערכי המפתח שלהם שווים ויש להם אותה הצורה בדיוק.

דוגמא 1: שני עצים חיפוש בינאריים הבאים שווים:



דוגמא 2: שני עצים חיפוש בינאריים הבאים שונים:



כתוב אלגוריתם (קוד) שמקבלת שני עצים חיפוש בינאריים ומחזיר true אם הם שווים, אחרת הוא מחזיר false

עבודה עצמית (10 דקות)

כתבו תוכנית שבודקת אם שני עצים בינאריים זהים

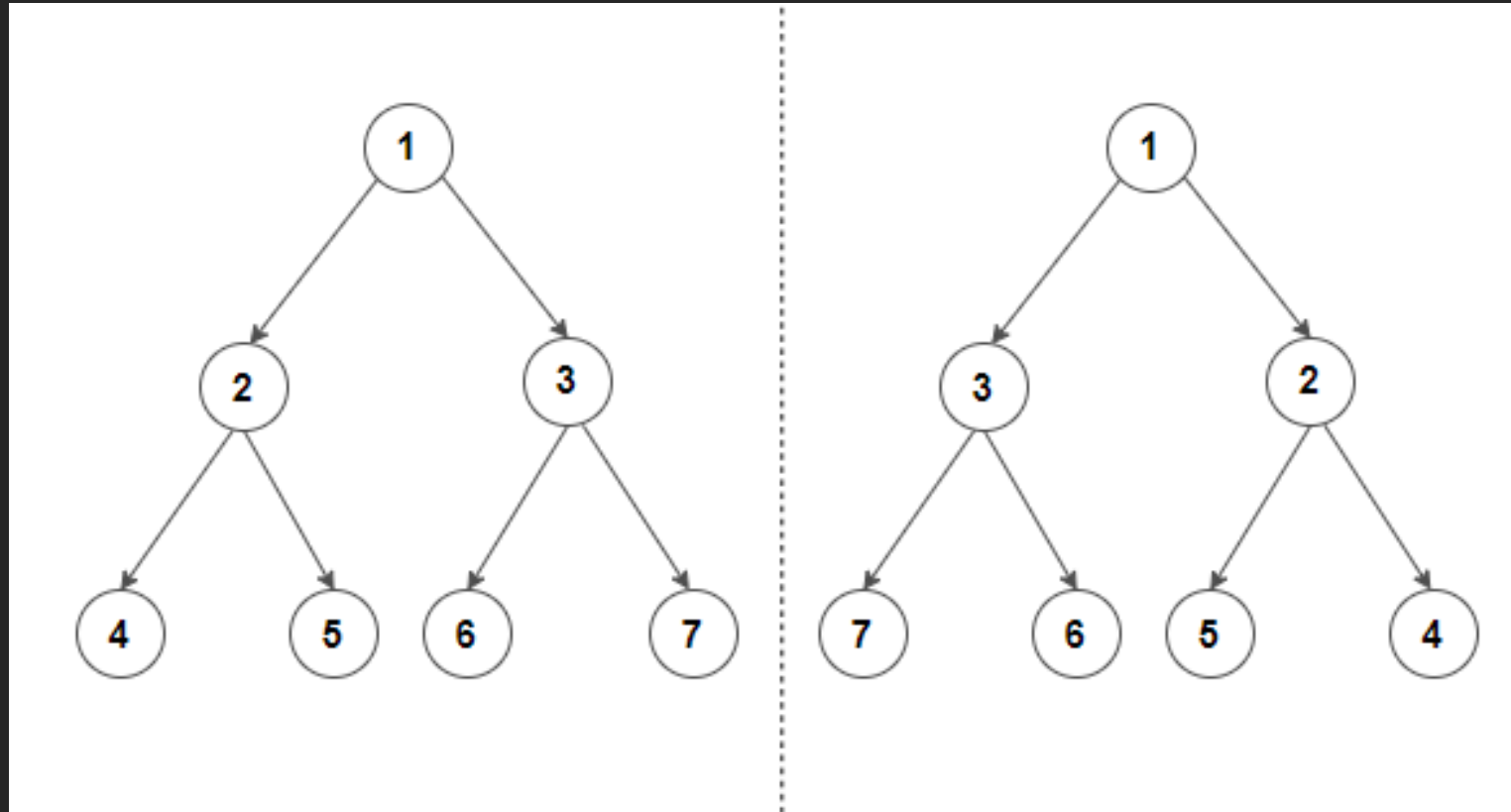
```
// ***** Question 2 ***** //
public static boolean isIdentical(Node x, Node y) {
    /*1. both empty */
    if (x == null && y == null)
        return true;

    /* 2. both non-empty -> compare them */
    if (x != null && y != null)
        return (x.data == y.data
                && isIdentical(x.left, x.left)
                && isIdentical(x.right, y.right));

    /* 3. one empty, one not -> false */
    return false;
}
```

```
System.out.println(BST.isIdentical(tree.root, tree2.root));
```

כתבו תוכנית אשר בהינתן עץ בינארי תהפוך אותו לעץ בינארי מראה



עבודה עצמית (10 דקות)

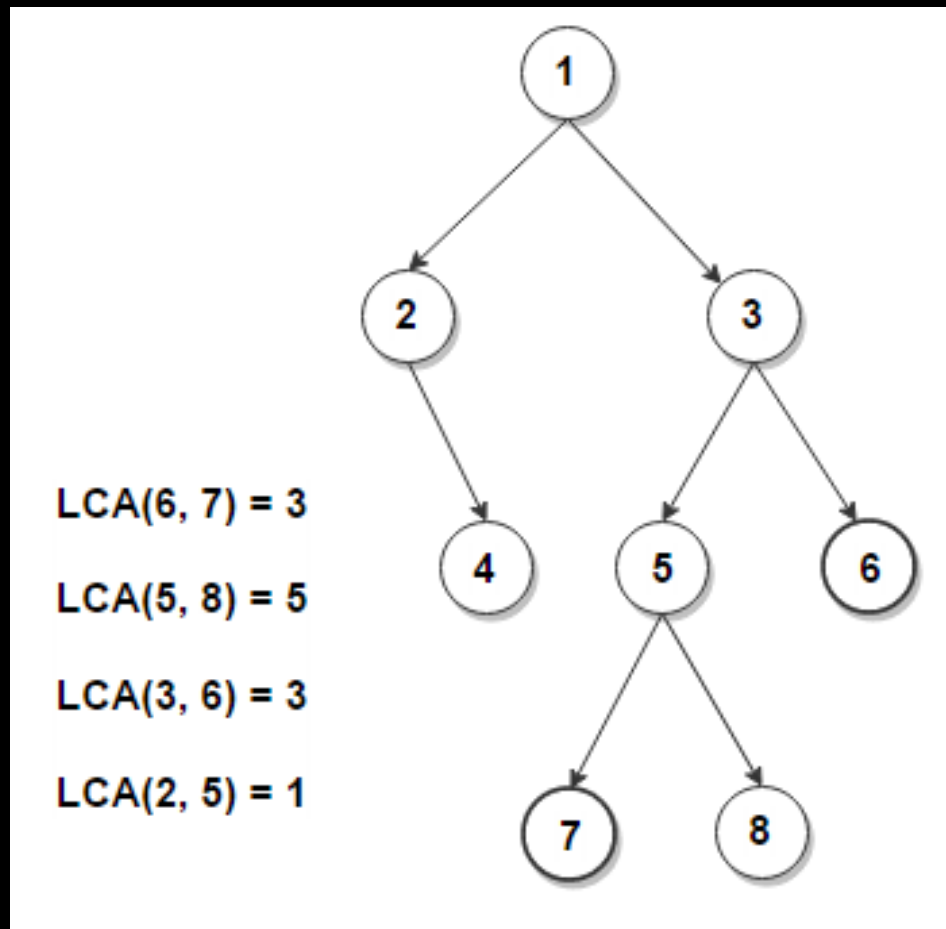
כתבו תוכנית אשר בהינתן עץ בינארי תהפוך אותו לעץ בינארי מראה

```
// ***** Question 3 *****
// Utility function to swap left subtree with right subtree
public static void swap(Node root) {
    if (root == null) { return; }
    Node temp = root.left;
    root.left = root.right;
    root.right = temp;
}
// Function to convert given binary Tree to its mirror
public static void convertToMirror(Node root)
{
    // base case: if tree is empty
    if (root == null) { return; }
    // convert left subtree
    convertToMirror(root.left);
    // convert right subtree
    convertToMirror(root.right);
    // swap left subtree with right subtree
    swap(root);
}
```


כתבו תוכנית אשר מחזירה את ה-LCA (Lowest Common Ancestor) של שני קודקודים בעץ בינארי

בהינתן עץ בינארי ו-2 קודקודים x ו-y

כתבו תוכנית אשר מוצאת את האב המשותף הנמוך ביותר של x ו-y



Method 1 (By Storing root to n1 and root to n2 paths):

- 1) Find path from root to n1 and store it in a vector or array.
- 2) Find path from root to n2 and store it in another vector or array.
- 3) Traverse both paths till the values in arrays are same.
Return the common element just before the mismatch.

כתבו תוכנית אשר מחזירה את ה-LCA (Lowest Common Ancestor) של שני קודקודים בעץ בינארי

// ***** Question 4 ***** //

פתרון 1:

```
public int findLCA(int n1, int n2) {
    if(search(n1) && search(n2)) {
        List<Node> path1 = findPath(n1);
        List<Node> path2 = findPath(n2);
        int i = 0;
        for(i=0; i<path1.size() && i<path2.size(); i++) {
            if(path1.get(i) != path2.get(i)) break;
        }
        return path1.get(i-1).data;
    }
    return Integer.MIN_VALUE;
}

private List<Node> findPath(int key) {
    List<Node> path = new ArrayList<Node>();
    return findPath(root, key, path);
}

private List<Node> findPath(Node curr, int key, List<Node> path) {
    if(curr == null ) return path;
    path.add(curr);
    if(curr.data < key)
        findPath(curr.right, key, path);
    else
        findPath(curr.left, key, path);
    return path; // curr.data == key
}
```

plus extra
spaces

כתבו תוכנית אשר מחזירה את ה-LCA (Lowest Common Ancestor) של שני קודקודים בעץ בינארי

הצעה: לעבור בעץ עד שמגיעים לפיצול – עבודה עצמית (10 דקות)

```
// ***** Question 4 *****  
public int findLCA(int n1, int n2) {  
    return findLCA(Node(n1), Node(n2)).data; }  
}
```

```
// This function returns pointer to the LCA  
// values n1 and n2. This function  
// n1 and n2 are present in Binary Tree
```

```
public Node findLCA(Node node1, Node node2)  
{
```

```
    if (node1 == null || node2 == null)  
        return null;
```

```
    // If both n1 and n2 are present in left
```

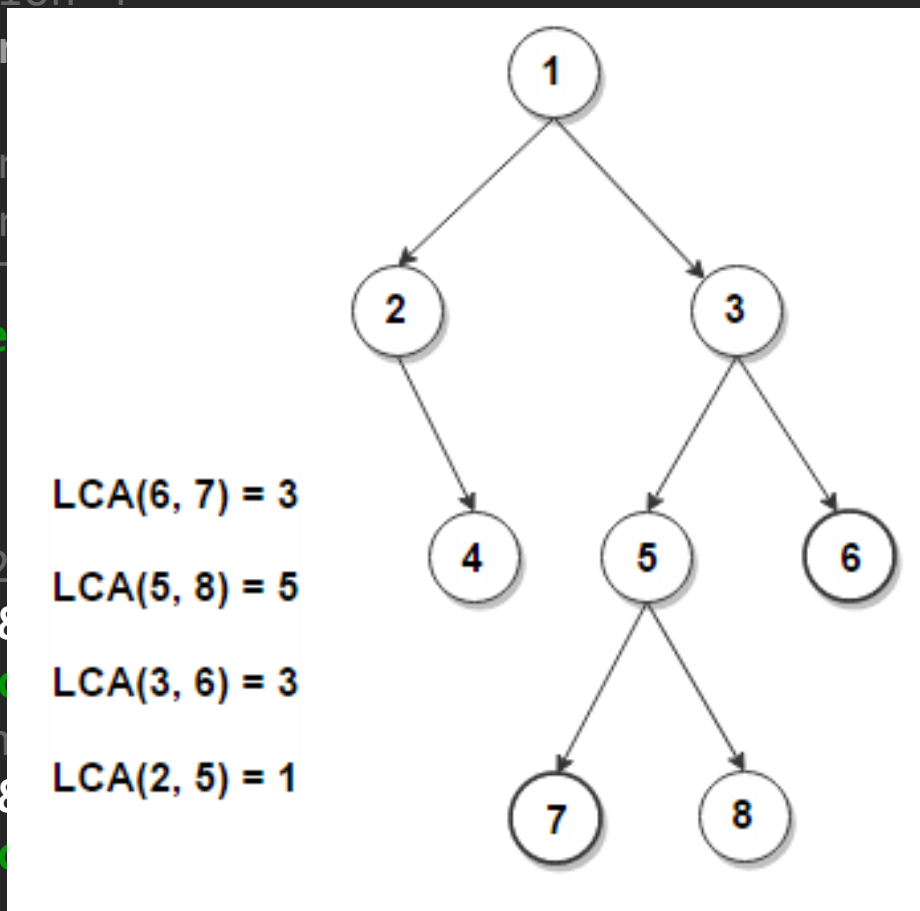
```
    if (node1.data > n1 && node2.data > n1 &&  
        node1.data > n2 && node2.data > n2)  
        return findLCA(node1.left, node2.left);
```

```
    // If both n1 and n2 are present in right
```

```
    if (node1.data < n1 && node2.data < n1 &&  
        node1.data < n2 && node2.data < n2)  
        return findLCA(node1.right, node2.right);
```

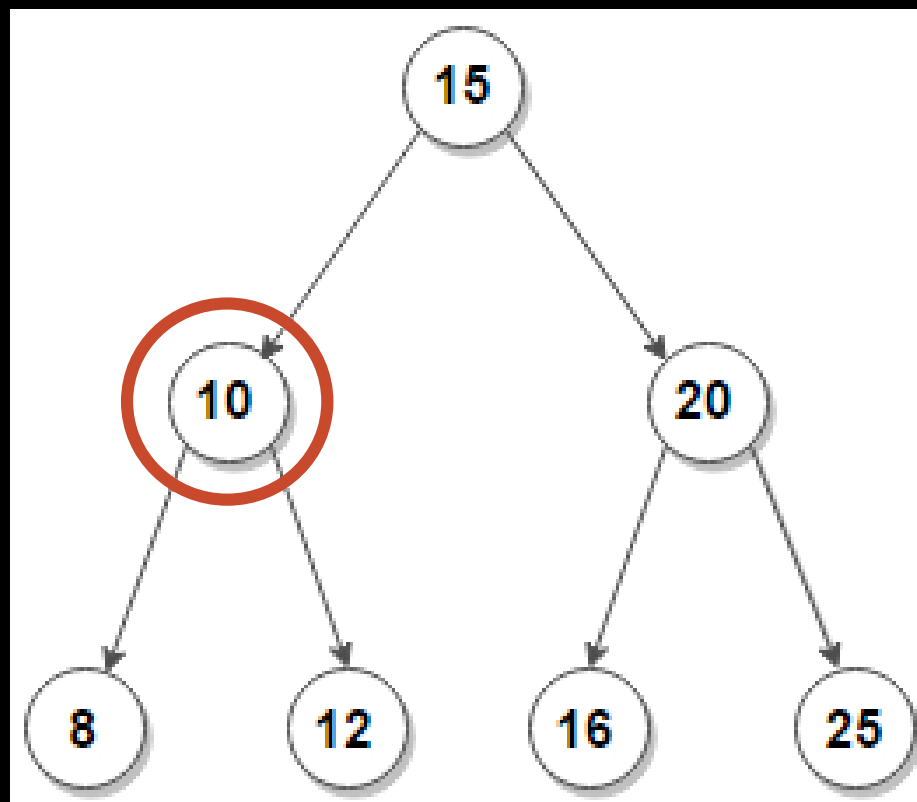
```
    return node1;
```

```
}
```



כתבו תוכנית אשר מוצאת את האיבר ה-K הקטן ביותר בעץ חיפוש בינארי

K=2



כתבו תוכנית אשר מוצאת את האיבר ה- k הקטן ביותר בעץ חיפוש בינארי

פתרון:

ידוע כי In-Order traversal של עץ בינארי מחזיר את האיברים בסדר עולה ולכן על מנת למצוא את האיבר ה- k בגודלו נבצע `InOrder()` ונאחסן את התוצאות במערך, לאחר מכן נחזיר את האיבר ה- k בגודלו.

```
public int kthSmallest(Node root, int k) {
    if( k > size() ) return Integer.MAX_VALUE;
    List<Integer> list = new ArrayList<>();
    preOrder(root, list);
    return list.get(k - 1);
}

private void preOrder(Node root, List<Integer> list) {
    if (root == null) return;
    preOrder(root.left, list);
    list.add(root.data);
    preOrder(root.right, list);
}
```

פתרון זה לא יעיל מבחינת מקום

ניתן לשמור את כמות הקריאות הרקורסיביות במשתנה עד שמגיעים ל- k (תרגיל בית)

שאלה 6

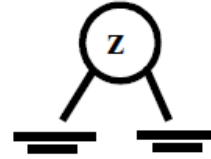
כתוב פונקציה שמקבלת נתון הנמצא בצומת ובודקת אם הצומת הוא עלה. הפונקציה מחזירה מחרוזת :
אם הנתון אינו נמצא באחת מהצמתים הפונקציה מחזירה מחרוזת "not a vertex".
אם הצומת הוא אינו עלה הפונקציה מחזירה מחרוזת "not a leaf".
אם הצומת הוא עלה הפונקציה מחזירה מחרוזת "a leaf".

```
public String isLeaf( int data){ . . . }
```

 תרגיל 9 לעבודה עצמית עצים בינאריים

עבודה עצמית (10 דקות)

```
// ***** Question 6 *****  
public String isLeaf(int data){  
    Node current = root;  
    while(current != null) {  
        if( data == current.data)  
            break;  
        else if( data > current.data )  
            current = current.right;  
        else  
            current = current.left;  
    }  
  
    if(current == null) return "not a vertex";  
    else if ( current.right != null || current.left != null )  
        return "not a leaf";  
    else  
        return "leaf";  
}
```

נתון עץ חיפוש בינארי T , וצומת בודד z כמתואר בציור. נניח כי אנו שומרים לכל צומת ב- T את הערך שלו $data$, מצביע לבן השמאלי שלו $left$, מצביע לבן הימני של $right$, ומספר הצמתים בתת העץ שהוא מהווה שורש שלו – num . בצומת z קיים ערך $data$, $num=1$ ו- $left(z)=right(z)=null$.

א. כתוב אלגוריתם (פסאודו קוד) $add(node, z)$ להוספת צומת z לעץ בינארי T בעל שורש $root$ ע"י הקריאה הראשונה $add(root, z)$, כך שלאחר ההוספה, כל צומת ב- T שומר את מספר הצמתים בתת העץ שלו. זמן הריצה הדרוש $O(L)$ כש- L הוא מספר הרמות ב- T .

ב. כתוב אלגוריתם (פסאודו קוד) $less(node, x)$ כך שבהינתן מספר x , ו- $node=root(T)$, השורש של T , מחזיר כמה איברים ב- T קטנים ממש מ- x . זמן ריצה יהיה $O(L)$ כש- L הוא מספר הרמות ב- T .

עבודה עצמית (20 דקות)

מבחן לדוגמא 2016

```

public class Node {
    int data;
    Node right;
    Node left;
    public int num;
    public Node(int data)
    {
        this.data = data;
        left = right = null;
        num = 1;
    }
}

```

נתון עץ חיפוש בינארי T, וצומת בודד z כמתואר בציור. נניח כי אנו שומרים לכל צומת ב-T את הערך שלו data, מצביע לבן השמאלי שלו left, מצביע לבן הימני של right, ומספר הצמתים בתת העץ שהוא מהווה שורש שלו – num. בצומת z קיים ערך data, num=1 ו-
`left(z)=right(z)=null`

```

public void add (Node node, Node z) {
    if(node == null)
        root = z;
    else {
        boolean exit = false;
        while(!exit) {
            if(z.data < node.data) {
                if(node.left == null)
                {
                    // Put Here
                    node.num ++ ;
                    node.left = z;
                    exit = true;
                }
            }
            else {
                node.num ++ ;
                node = node.left;
            }
        }
        // z.data < node.data
        ... Symmetric
    }
} // End Else
}

```

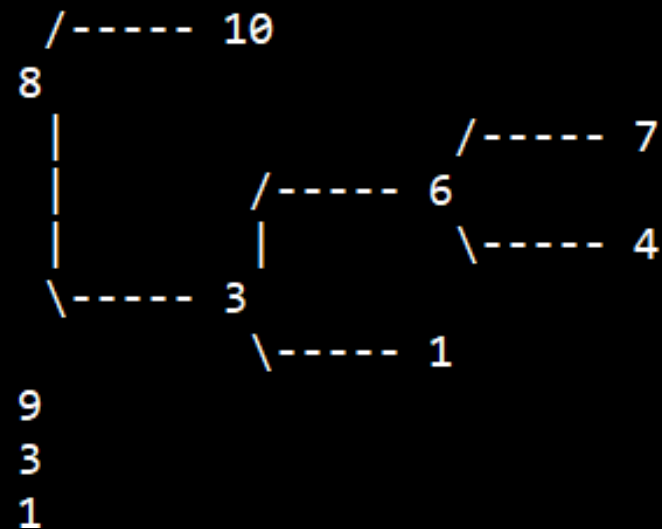
א. כתוב אלגוריתם (פסאודו קוד) `add(node, z)` להוספת צומת `z` לעץ בינרי `T` בעל שורש `root` ע"י הקריאה הראשונה `add(root, z)`, כך שלאחר ההוספה, כל צומת ב-`T` שומר את מספר הצמתים בתת העץ שלו. זמן הריצה הדרוש $O(L)$ כש-`L` הוא מספר הרמות ב-`T`.

```

public static void main(String[] args) {
    BST tree = new BST();
    int[] keys = {8,3,1,6,4,7,10,14,13};
    for(int i : keys)
        tree.add(tree.root,new Node(i));

    tree.printTree();
    System.out.println(tree.root.num);
    System.out.println(tree.root.right.num);
    System.out.println(tree.root.left.left.num);
}

```



ב. כתוב אלגוריתם (פסאודו קוד) $\text{less}(\text{node}, x)$ כך שבהינתן מספר x , ו-

$\text{node} = \text{root}(T)$, השורש של T , מחזיר כמה איברים ב- T קטנים ממש מ- x . זמן

ריצה יהיה $O(L)$ כש- L הוא מספר הרמות ב- T .

// B:

```
public int less(Node node, int x) {
```

```
    // Base Case:
```

```
    if ( node == null ) return 0;
```

```
    else if ( x == node.data ) {
```

```
        if(node.left != null)
```

```
            return node.left.num;
```

```
        else
```

```
            return 0;
```

```
    }
```

```
    else if( x > node.data ) {
```

```
        if(node.left != null)
```

```
            return node.left.num + 1 + less(node.right,x);
```

```
        else
```

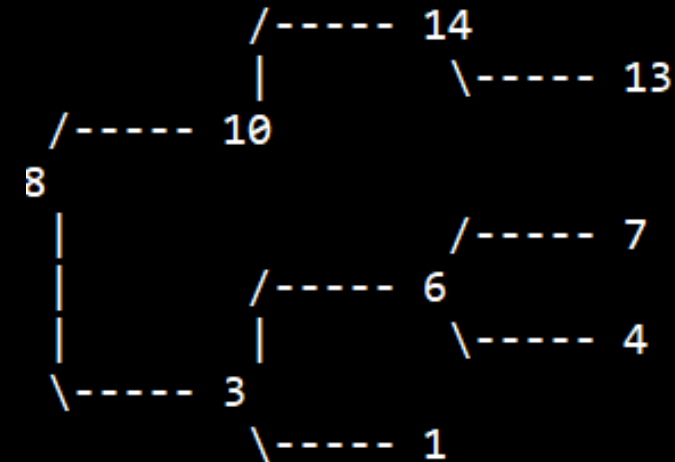
```
            return 1 + less(node.right,x);
```

```
    }
```

```
    else // x < node.data
```

```
        return less(node.left,x);
```

```
}
```



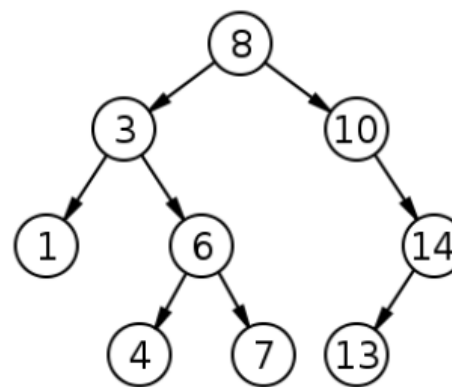
שאלה 2 (35 נק') הדפסת עץ חיפוש בינארי לפי רמות.

יש לכתוב מחלקה בשם Question2.

בתוך המחלקה יש לכתוב פונקציה סטטית שמקבלת עץ חיפוש בינארי ומדפיסה את הקדקודים שלו לפי רמות. הקוד שמממש עץ חיפוש בינארי מצורף למבחן.

```
public static void printTreeByLevels(BST tree)
```

דוגמה : קלט :



פלט :

```
8
3 10
1, 6, 14
4, 7, 13
```

מבנה נתונים מבחן קיץ 2015 מועד ב
עבודה עצמית (10 דקות)

```
// ***** Question 8 ***** //
public static void printTreeByLevels(BST tree) {
    // Base Case
    if(tree.root == null)
        return;
    // Create an empty queue for level order traversal
    Queue<Node> q = new LinkedList<Node>();
    // Enqueue Root and initialize height
    q.add(tree.root);
    while(!q.isEmpty()) {
        int NumberOfNodesAtCurrentLevel = q.size();
        while(NumberOfNodesAtCurrentLevel > 0 ) {
            Node top = q.poll();
            System.out.print(top.data + " ");
            if(top.left != null)
                q.add(top.left);
            if(top.right != null)
                q.add(top.right);
            NumberOfNodesAtCurrentLevel--;
        }
        System.out.println();
    }
}
```

כתבו פונקציה שמקבלת עץ בינארי ומחזירה אמת אם הוא עץ שלם.

עבודה עצמית (10 דקות)

```
// ***** Question 9 *****  
public boolean isFullTree(Node node) {  
    // if empty tree  
    if(node == null)  
        return true;  
  
    // if leaf node  
    if(node.left == null && node.right == null )  
        return true;  
  
    // if both left and right subtrees are not null  
    // Return Recur on Left subtree and right Subtree  
    if((node.left != null) && (node.right != null))  
        return (isFullTree(node.left) && isFullTree(node.right));  
  
    // if none work  
    return false;  
}
```