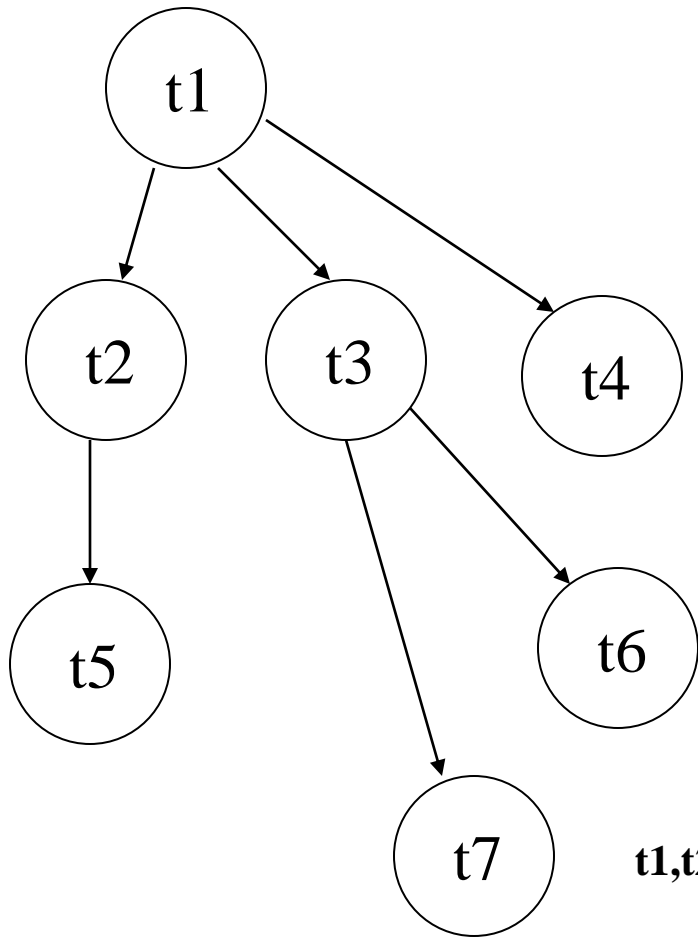


פרק - 8

עצים בינאריים

עץ הגדרה :



עץ – גרף קשור ללא מעגלים .

צמת – כל איבר בעץ

בנים של צומת A – האיברים העוקבים לצומת .

אב של צומת – האיבר הקודם לצומת.

צמתים – t1,t2,t3,t4,t5,t6,t7

עלים – t5,t7,t6,t4

שורש – t1

t3 הוא **צאצא** של t1

t3 הוא **אב** של t6
רועי רחמני

עלה – צמת ללא בנים .

שורש – צמת שאין לו אב .

מבנה הנתונים עץ בינארי :

עץ בינרי הוא:

עץ ריק או עץ שלכל צמת בו לכל היותר שני בנים .

בכל צמת נבדיל בין בן שמאלי ובן ימני כאשר כל אחד מהם הוא שורש של תת-עץ-בינארי בפני עצמו .

הגדרה :

% definition:

tree(nil).

tree(t(L,_,R)):- tree(L), tree(R).

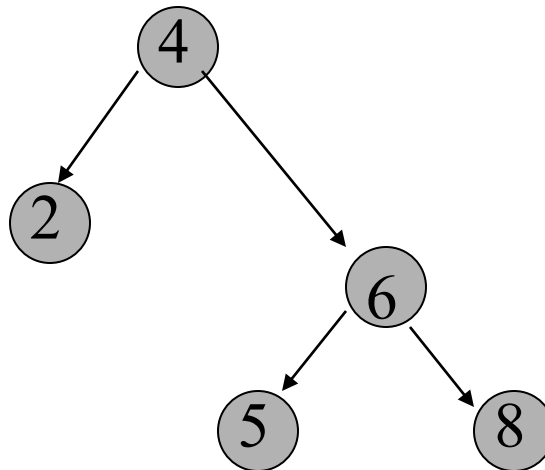
יצירת עץ חדש :

`new_tree(Val,t(nil,Val,nil)).`

? `-new_tree(10, T).`

`T = t(nil, 10, nil)`

העץ הבא :



מיוצג ע"י היחס הבא

`t(t(nil,2,nil),4,t(t(nil,5,nil),6,t(nil,8,nil)))`

יחסים פשוטים :

בדיקת שייכות :

% member check

in(X,t(_,X,_)):-!.

in(X,t(L,_,R)):-in(X,L),!;in(X,R).

2 - החזרת סכום הערכים בעץ המכיל מספרים שלמים :

sum_tree(t(L,X,R),Sum):-

sum_tree(L,Sum1), sum_tree(R,Sum2),

Sum is X+Sum1+Sum2.

sum_tree(nil,0).

3 – גובה שורש העץ :

height(t(L,_,R),H):-

height(L,H1), height(R,H2),

max(H1,H2,H3),

H is H3+1.

height(nil,-1).

4 – ספירת עלים בעץ :

countLeaf(nil,0).

countLeaf(t(nil,_,nil),1):-!.

countLeaf(t(R,_L),Count):-

countLeaf(R,Count1),

countLeaf(L,Count2),

count is Count1+Count2.

5-החזרת רשימת עלים בעץ :

```
getLeaf (nil,[]).  
getLeaf (t(nil,X,nil),[X]):-!.  
getLeaf (t(L,_,R),List):-  
    getLeaf (L,ListL),  
    getLeaf (R,ListR),  
    conc(ListL,ListR,List).
```

ללא conc עם רשימות הפרש :

```
getLeaf(Tree,List):-  
    getLeaf(Tree,List-[]),!.
```

```
getLeaf(nil,T-T).  
getLeaf(t(nil,X,nil),[X|T]-T):-!.  
getLeaf(t(R,_,L),List-T):-  
    getLeaf(L,List-List1),  
    getLeaf(R,List1-T).
```

או :

```
getLeaf(nil,T-T).  
getLeaf(t(nil,X,nil),[X|T]-T):-!.  
getLeaf(t(R,_,L),List-T):-  
    getLeaf(L,L1-T1),  
    getLeaf(R, L2-T2),  
    conc(L1-T1, L2-T2, List-T).
```

6-החזרת הצמתים בעץ שגדולים מ 10 :

count10(nil,[]).

count10(t(L,X,R),List):-X <=10 ,!,

count10 (L,ListL),

count10 (R,ListR),

conc(ListL,ListR,List).

count10(t(L,X,R),[X|List]):-

count10 (L,ListL),

count10 (R,ListR),

conc(ListL,ListR,List).

**7 – עצים איזומורפיים – שני עצים הם איזומורפיים אם
יש להם מבנה דומה וערכים מקבילים :**

isotree(nil,nil).

isotree(t(L1,X,R1),t(L2,X,R2)):-

isotree(L1,L2), isotree(R1,R2).

isotree(t(L1,X,R1),t(L2,X,R2)):-

isotree(R1,L2), isotree(L1,R2).

8 – המסלול עד לאיבר מסוים (בהנחה שקיים):

path_to_item(X,t(_,X,_),[X]).

path_to_item(X,t(L,Y,R),[Y|Path]):-

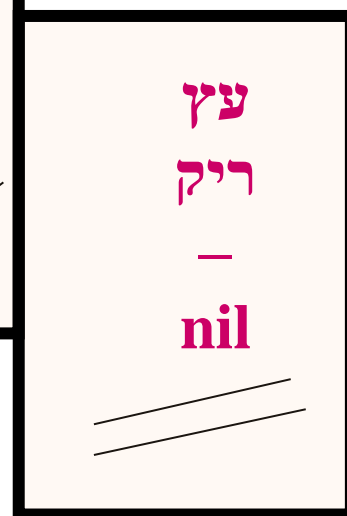
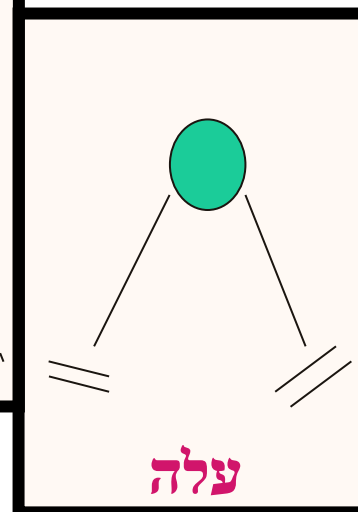
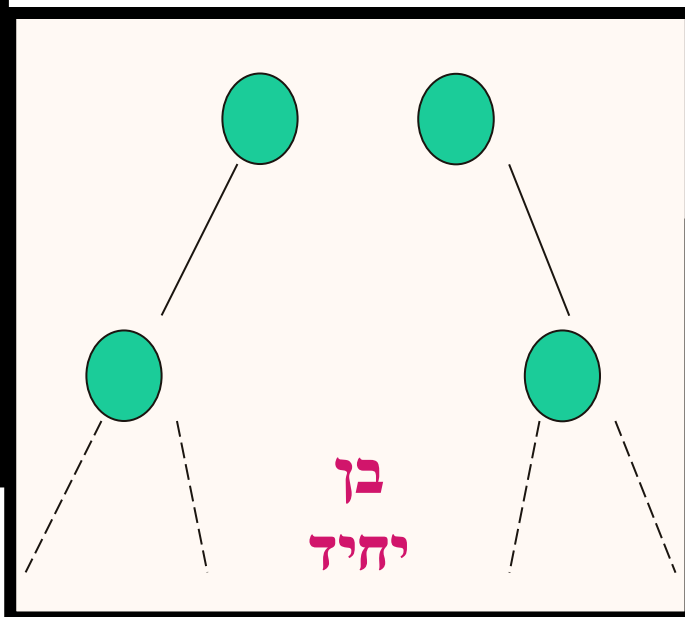
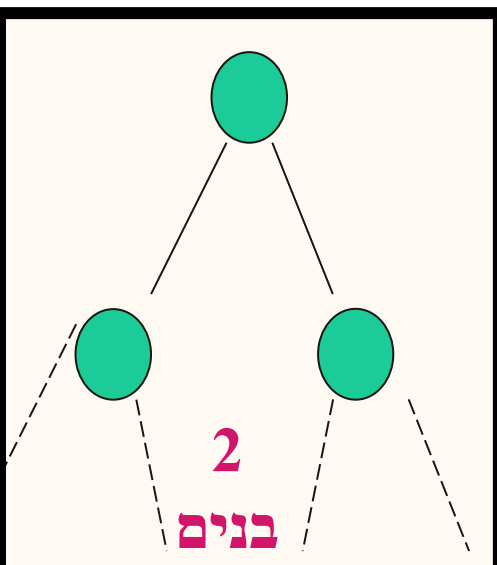
path_to_item(X,L,Path)

;

path_to_item(X,R,Path).

מקרי קצה

רוב הטעויות בפתרון בעיות בעצים בינאריים מתרחשות בגלל חוסר תשומת לב למקרי הקצה השונים בהם האלגוריתם נתקל.



9 – פרדיקט שמצליח רק אם בעץ יש רק צמתים בעלי לכל היותר בן יחיד :

one_l(t(nil,_,R)):-

one_l(R) .

one_l(t(L,_,nil)):-

one_l(L) .

one_l(nil).

**10 – פרדיקט שמצליח רק אם בעץ כל צומת בעץ גדול
מסכום ערכי בניו הישירים ובכל עלה יש ערך חיובי .**


big(nil).

big(t(L,X,R)) :-

**get(L,XL), get(R,XR) ,Temp is XL+XR , X > Temp,! ,
big(L), big(R).**

get(nil,0).

get(t(_,X,_),X).



Get יחזיר את ערך הצומת
וחוסך מאיתנו התעסקות במבנה
הצומת הנוכחי :
2 בנים בן יחיד או עלה ...

עצים בינאריים

סריקות :

- נתון עץ בינרי המכיל מספר שלם בכל אחד מצמתיו.
- יש לכתוב 3 פרדיקטים לסריקת העץ בשלושה אופנים:
בסדר תחילי (preorder),
בסדר תוכי (inorder)
ובסדר סופי (postorder).

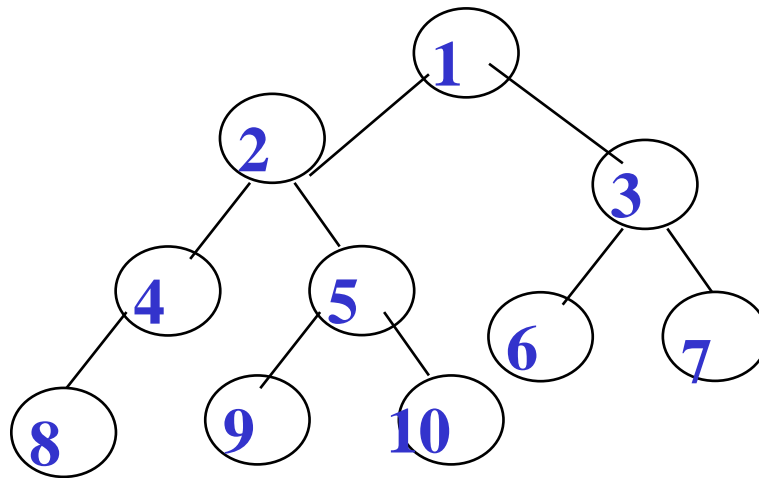
כל אחד מהפרדיקטים יצור את רשימת עלי העץ המתקבלת מאופן הסריקה המתאים.

הגדרות:

סריקה על-פי סדר תחילי: תחילה מבקרים בשורש, לאחר מכן נסרק תת-העץ השמאלי בסדר תחילי.

סריקה על-פי סדר תוכי: תחילה נסרק תת-העץ השמאלי בסדר תוכי, לאחר מכן מבקרים בשורש ולבסוף נסרק תת-העץ הימני בסדר תוכי.

סריקה על-פי סדר סופי: תחילה נסרק תת-העץ השמאלי בסדר סופי, לאחר מכן נסרק תת-העץ הימני בסדר סופי ולבסוף מבקרים בשורש. רשימות העלים שיתקבלו מהרצת התכנית על העץ הנתון:



תחילי: [1,2,4,8,5,9,10,3,6,7]

תוכי: [8,4,2,9,5,10,1,6,3,7]

סופי: [8,4,9,10,5,2,6,7,3,1]

12 - פרדיקט הסופר עבור כמה צמתים מתקיים שערך השורש שווה למספר הצמתים בתת עץ (כולל עצמו).

$eq(nil, 0).$

$eq(t(L, X, R), N):-$

$eq(L, N1) , eq(R, N2),$

$(countNodes(t(L, X, R), X) , ! , N \text{ is } N1 + N2 + 1$

$;$

$N \text{ is } N1 + N2$

$).$

12 - פרדיקט הסופר עבור כמה צמתים מתקיים שערך השורש שווה למספר הצמתים בתת עץ (כולל עצמו).

האלגוריתם : רוץ על כל תת העצים האפשריים ולכל אחד מהם חשב את גודלו אם גודל זה זהה לערך השורש הוסף 1 למונה .

`eq(nil,0,0).`

%% The second fild is for the size of tree and

%%the third fild is for count the good digit

`eq(t(L,X,R),Size,Num):-`

`eq(L,Size1,Num1),`

`eq(R,Size2,Num2),`

`Size is Size1+Size2+1 ,`

`(X=Size,!,Num is Num1+Num2+1`

`;`

`Num is Num1+Num2).`

13. כתוב פרדיקט המקבל עץ ורמה בעץ ומחזיר את רשימת הצמתים ברמה מסוימת בעץ ע"פ סידורם משמאל לימין .

nodes_of_level(t(L,X,R),N,List):-N > 0 ,

N1 is N -1 ,

nodes_of_level(L,N1,List1),

nodes_of_level(R,N1,List2),

conc(List1,List2,List).

nodes_of_level(t(L,X,R),0,[X]).

nodes_of_level(nil,_,[]).

nodes_of_level(t(L,X,R),N,List-T):-

N>0,

N1 is N-1,

nodes_of_level(L,N1,List-List2),

nodes_of_level(R,N1,List2-T).

nodes_of_level(t(L,X,R),0,[X|T]-T).

nodes_of_level(nil,_,List-List).

/*

?-nodes_of_level(t(t(nil,2,nil),4,t(t(nil,5,nil),6,t(nil,8,nil))),2,List-[]).

List = [5,8] ;

***/**

14- כתוב פרדיקט המקבל עץ ומחזיר את סכום הגבהים של צמתיו כולל של השורש.

גובה של עץ מוגדר כאורך המסלול המקסימלי מאותה צומת לעלה (שמתחתיו).

sum_heights(t(L,X,R),Sum):-

height(t(L,X,R),H),

sum_heights(L,SumL),

sum_heights(R,SumR),

Sum is SumL+SumR+H.

sum_heights(nil,0).

גרסה יעילה יותר נחשב את הסכום ואת הגובה באותו המעבר: !

sum_heights(t(L,X,R),Sum,H):-

sum_heights(L,SumL,H1),

sum_heights(R,SumR,H2),

max(H1,H2,H3),

H is H3 +1 ,

Sum is Sum1+Sum2+H.

sum_heights(nil,0,-1).

. 15

****/subtree(Subtree,Tree) is true if Subtree is
a subtree of the binary tree Tree */***

subtree(T,T).

subtree(S,tree(L, X, R)):-

subtree(S,L).

subtree(S,tree(L, X, R)):-

subtree(S,R).

```
ordered_tree(nil).
```

```
ordered_tree(t(L, X, R)):-
```

```
    bigger_then(X,L),
```

```
    smaller_then(X,R),
```

```
    ordered_tree(R),ordered_tree(L).
```

```
bigger_then(_,nil):-!.
```

```
bigger_then(X,t(L,Y,R)):- X > Y ,
```

```
    bigger_then(X,L),
```

```
    bigger_then(X,R).
```

```
smaller_then(_,nil):-!.
```

```
smaller_then(X,t(L,Y,R)):- X < Y ,
```

```
    smaller_then(X,L),
```

```
    smaller_then(X,R).
```

או בדרך אחרת :

oreder_tree(T):-

inorder(T,L-[]),

order_list(L).

inorder(t(L,X,R),Xs-T):-

inorder(L,Xs-[X|Ys]),

inorder(R,Ys-T).

inorder(nil,T-T).

order_list([]).

order_list([X]).

order_list([X,Y|Tail]):-

X<Y,

order_list([Y|Tail]) .

או בדרך אחרת
:

ordered(Tree):-

ordered(Tree,-100000,+100000).

% If needed the values of Min and Max can be

% determined by a pre-scan of the tree

ordered(t(L,X,R),Min,Max):-

X>Min, X<Max,

ordered(L,Min,X),

ordered(R,X,Max).

ordered(nil,_,_).

או בדרך אחרת
:

ordered(Tree):-

ordered(Tree,Min,Max).

ordered(t(L,X,R),Min,Max):-

(var(Min),! ; X>=Min),(var(Max),! ; X<Max),

ordered(L,Min,X),

ordered(R,X,Max).

ordered(nil,_,_).

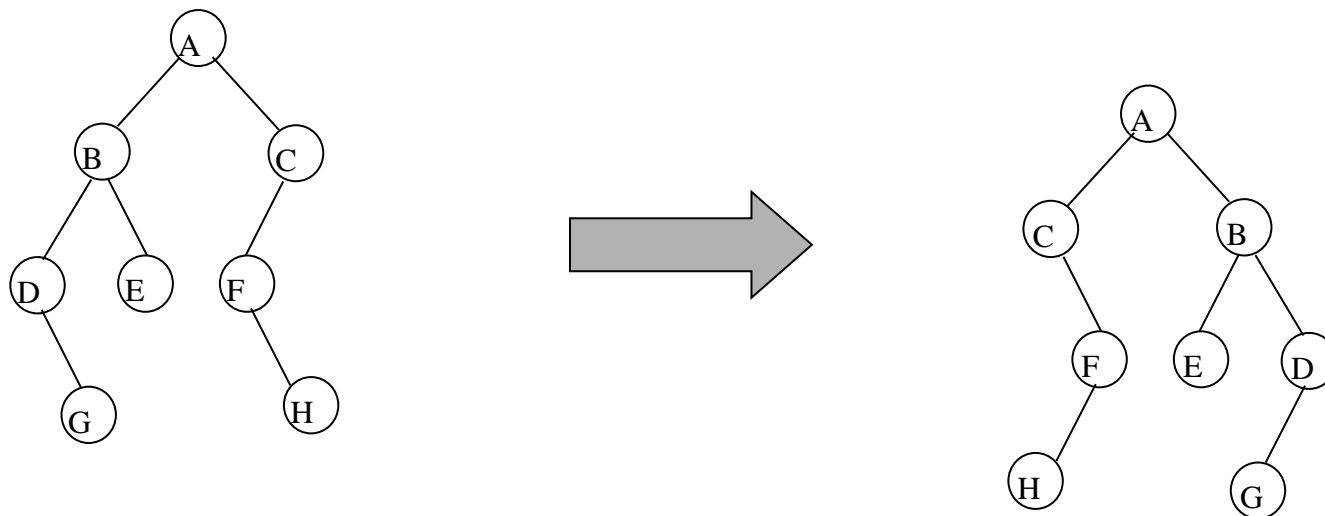
% replace every node, except the root with its father
replace(t(L,Root,R),NewTree):-
replace(t(L,Root,R),Root,NewTree).

replace(t(Left,X,Right),Father,t(Left1,Father,Right1)):-
replace(Left,X,Left1),
replace(Right,X,Right1).

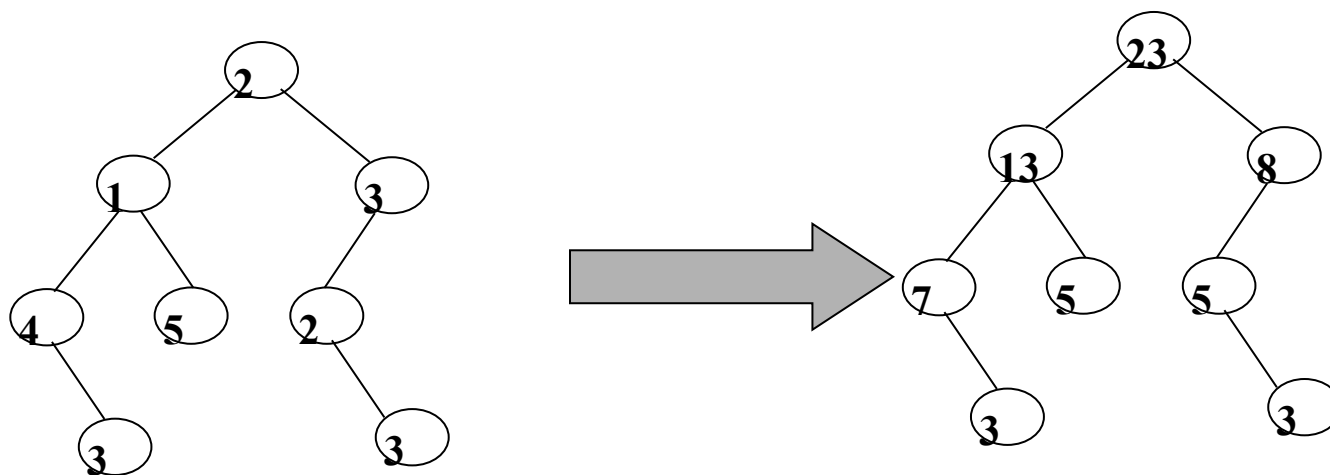
balance(t(L,X,R),H) :-
balance(L,HL),
balance(R,HR),
Diff is HL-HR,
Diff =<1,Diff>=-1,
Max(HL,HR,TempMax),
H is Max +1.

balance(nil,-1).

- כתוב פרדיקט בשם mirror המקבל כקלט עץ בינרי Tree1 ויוצר עץ בינרי Tree2 המהווה תמונת ראי של Tree1. כלומר הפרדיקט mirror מחליף את התת-עץ הימני בתת-עץ השמאלי של כל צומת בעץ הנתון.



- כתוב יחס שיקבל עץ בינארי ויחזיר אותו כשבכל צומת יעודכן סכום הערכים בתת עץ ששורשו הוא הצומת (עצמו)



- כתוב תכנית (`count_level` (Tree , Level , Nodes , List) המקבלת כקלט עץ בינרי Tree ומספר שלם אי-שלילי Level. התכנית תחזיר כפלט את רשימת הצמתים בעץ הנמצאים ברמה Level (משמאל לימין) וכן את מספר הצמתים ברמה Level.
- הערה: רמת השורש היא 0.

- ערימה היא עץ בינרי שבו הערך של כל צומת גדול מהערך של כל אחד מצאצאיו.
- כתוב יחס `heapify (Tree, Heap)` המקבל כקלט עץ בינרי המכיל מספר בכל צומת ומחזירה עץ בינרי בעל מבנה זהה המהווה ערימה, על-ידי החלפת ערכי צמתים מסוימים בעץ.
- הדרכה:
- בהינתן עץ בינרי, שנה את התת-עץ הימני ואת התת-עץ השמאלי כך שכל אחד מהם יהווה ערימה ולאחר מכן קבע את מקומו של שורש העץ כראוי.
- שים לב, התכנית אינה משנה את מספר הבנים של כל צומת בעץ.

לתיקונים והערות :

royrachmany@gmail.com