

## מבני נתונים

### פתרון 5 - עצים

1. בהינתן עץ בינרי (לא עץ חיפוש) המכיל מספרים ששורשו root, כתבו פונקציה שמחזירה את מספר הצמתים

שערכם גדול מהערך של אביהם.

**פתרון** – לכל צומת, הפונקציה בודקת האם הערך של הבנים שלו גדולים משלו, וכן מחשבת באופן רקורסיבי בכל אחד מתתי העצים שלו את מספר הצמתים שערכם גדול משל אביהם.

```
int BigChild(Node_type* root)
{
    if (root == null)
        return 0;
    int count = 0;
    if ((root->left) && (root->left->data > root->data)) count++;
    if ((root->right) && (root->right->data > root->data)) count++;
    return count + BigChild (root->left) + BigChild (root->right);
}
```

### נכונות –

תנאי העצירה מתייחס למצב שהצומת לא קיים ולכן מוחזר 0.

לכל אחד מהצמתים בעץ, הפונקציה מבצעת בדיקה מפורשת לגבי בנוי, וממשיכה רקורסיבית לבדוק בכל אחד מתתי העצים שלו. בכל שלב מוחזר מספר הצמתים שמקיימים את התנאי הדרוש בכל התת-עץ.

### סיבוכיות –

לכל צומת בעץ מתבצעות 2 קריאות רקורסיביות (ועוד כמה פעולות קבועות) ולכן זמן הריצה הוא  $\Theta(n)$ .

## שאלה 2

נתון עץ בינרי ששורשו root. כתבו פונקציה היוצרת רשימה מקושרת אשר מחזיקה את איברי העץ לפי סדר postorder.

### תשובה

הפונקציה treeToLinkedList מקבלת את שורש העץ, יוצרת רשימה מקושרת ריקה ומפעילה את PostList שהיא פונקציה רקורסיבית שמכניסה את צמתי העץ לרשימה בסדר סופי. הרשימה שנוצרת היא רשימה מקושרת עם זקיף, לכן בסיום כל הקריאות הרקורסיביות treeToLinkedList מוחקת את האיבר הראשון, ואז מחזירה את ראש הרשימה המקושרת.

הפונקציה PostList מקבלת צומת v בעץ וחוליה curr ברשימה המקושרת (האחרונה ברשימה שנבנתה עד כה) ומכניסה רקורסיבית את תת העץ ששורשו v אל הרשימה המקושרת החל מהחוליה curr. הפונקציה מחזירה מצביע לאיבר האחרון ברשימה המקושרת.

כדי לבנות מהעץ רשימה מקושרת בסדר סופי post-order המעבר על צמתי העץ מתבצע בסדר הבא: קודם מעבר על תת העץ השמאלי, תת העץ הימני, ואז מעבר על השורש, כאשר בכל פעם הצומת מתווסף בסוף הרשימה המקושרת. לשם כך המצביע לרשימה המקושרת שעובר בקריאות הרקורסיביות הוא המצביע לאיבר **האחרון** ברשימה. האיבר הראשון ברשימה נשמר בפונקציה treeToLinkedList, ומוחזר בסוף התהליך.

נגדיר (כדי שנוכל להבדיל בין node של עץ לזה של הרשימה -> בהנחה שהם שונים):

```
typedef struct {
    int value;
    Tree_Node_Type *left;
    Tree_Node_Type *right;
} TreeNode;
```

```
typedef struct {
    int value;
    List_Node_type *next;
} ListNode;
```

```
ListNode *treeToLinkedList(TreeNode *root)
```

```
{
    if (root == NULL)
        return NULL;

    ListNode *head = MakeNode(); // we will use this node as guard (only as a pointer to the list)
    ListNode *curr = head;       //temp pointer running all over the list
    PostList(root, curr);

    curr = head;                  // =====
    head = head->next;            // Remove guard from list
    delete(curr);               // =====

    return head;
}
```

```

ListNode *PostList(TreeNode *v, ListNode *curr);
{
    If (v->left)
        curr = PostList(v->left, curr);

    If (v->right)
        curr = PostList(v->right, curr);

    curr->next = MakeNode(v->info);    // Create Next node with current tree node value
    return curr-> next;                // Return next Node (if we will not do it, we will return null
                                      and we will lose our entire list)
}

```

נכונות : עלינו לוודא כי מבקרים בכל צמתי העץ : הפונקציה פועלת על השורש, ובכל פעם מתבצעת קריאה רקורסיבית על תת העץ הימני ותת העץ השמאלי של השורש. מכיוון שכל צומת בעץ נמצא בתת העץ השמאלי או הימני של השורש, נקבל באופן אינדוקטיבי כי מבקרים בכל צומת באחת מן הקריאות הרקורסיביות.

סיבוכיות הזמן : כל צומת קורא לפונקציה הרקורסיבית לכל היותר פעמיים – עבור שני בניו, רק במידה והם קיימים כלומר – מספר הקריאות הרקורסיביות הן כמספר צמתי העץ. העלות של ביקור בצומת הוא  $\Theta(1)$  (רק הגדרה של מצביעים), ולכן אם יש  $n$  צמתים בעץ, סיבוכיות הזמן הכוללת היא  $\Theta(n)$ .

דוגמת הרצה : (אני לא יודעת כמה זמן הלינק הזה יחזיק...) <https://onlinegdb.com/SJxr5LrRr>

---

3. בהינתן עץ כלשהו המיוצג בשיטת בן-שמאלי אחים-ימניים, כתבו פונקציה המדפיסה את כל צמתי העץ, לפי סדר הרמות בעץ. (ראשון יודפס השורש, אחריו בניו, אח"כ בני בניו וכו').  
**רמז:** היעזרו בשני תורים.

## פתרון

נשתמש בשני תורים : Brother עבור האחים של הצומת ו-Kids עבור בניו.  
 עבור כל צומת אליו הפונקציה מגיעה, מודפסים נתוני הצומת, ואז מכניסים לתור Brother את האח הבא ברשימת האחים של הצומת, ולתור Kids את הבן הראשון של הצומת. בשיטה זו אנו מגיעים לכל צומת (או מאביו או מאחיו), כי כל צומת מכניס לתורים את שני הצמתים שהוא מצביע אליהם.  
 כדי שההדפסה תתבצע לפי רמות העץ, ההוצאה מן התורים תהיה בסדר הזה :  
**כל עוד התור Brother לא ריק** – הרי שיש בו אחים של הצומת הנוכחי, כלומר צמתים ברמה הנוכחית – **לכן נוציא ממנו את הצומת הבא ונטפל בו**.  
**כאשר התור Brother מתרוקן** – סימן שכל האחים שבהם טיפלנו הכניסו את כל הבנים הראשונים שלהם לתור הבנים – **לכן נוציא צומת מהתור Kids ונטפל בו**. במידה ויש לצומת זה אח – הוא יכניס את אחיו לתור Brother – ולכן באיטרציה הבאה נוציא את האח מתור Brother (שאינו כבר לא יהיה ריק). אם אין לו אח – נוציא את הצומת הבא **בתור Kids**.

```
void printTree(Node_type* root)
{
    if (root==null)
        printf("Empty tree");
    Queue_type *Brother, *Kids ;
    Node_type *v;

    printf(root->info);                                // visit tree root.
    if (root->child)
        AddQueue(root->child, Kids);

    while (!(Empty(Brother)) || !(Empty (Kids)))
    {
        if !(Empty(Brother))
        {
            DeleteQueue(v, Brother);
            printf(v->info);                            // visit current brother.
            if (v->brothers)
                AddQueue(v->brothers, Brother);
            if (v->child)
                AddQueue(v->child, Kids);
        }
    }
}
```

```

if (Empty(Brother) && !(Empty (Kids)))
{
    DeleteQueue(v, Kids);
    printf(v->info);           // visit current child.
    if (v->brothers)
        AddQueue(v->brothers, Brother);
    if (v->child)
        AddQueue(v->child, Kids);
}
}
}

```

**נכונות :** עלינו לוודא כי [1] מבקרים בכל צמתי העץ, [2] ולפי סדר הרמות שלו :

[1] הפונקציה מתחילה עם השורש, שמכניס את הבן הראשון שלו לתור הבנים [בלבד, שהרי לשורש אין אחים]. החל מהרמה הבאה בעץ, כאשר הפונקציה פועלת על צומת, היא קודם כל מדפיסה את הצומת עצמו. ואז מכניסה את שני הצמתים שהוא מצביע אליהם – האח הימני שלו והבן השמאלי שלו – כל אחד לתור המתאים לו. מכיוון שאל כל צומת בעץ יש מצביע או מאביו (אם הוא הבן השמאלי ביותר) או מאחיו שמשמאלו (בתוך הרשימה המקושרת של האחים), נקבל כי מכניסים כל צומת לאחד התורים.

[2] **נותר להוכיח – כל הצמתים מרמה L הודפסו משמאל לימין לפני שהוצא מהתור Kids בן ששייך לרמה L+1.**

**הוכחה** – באינדוקציה על הרמה L בעץ.

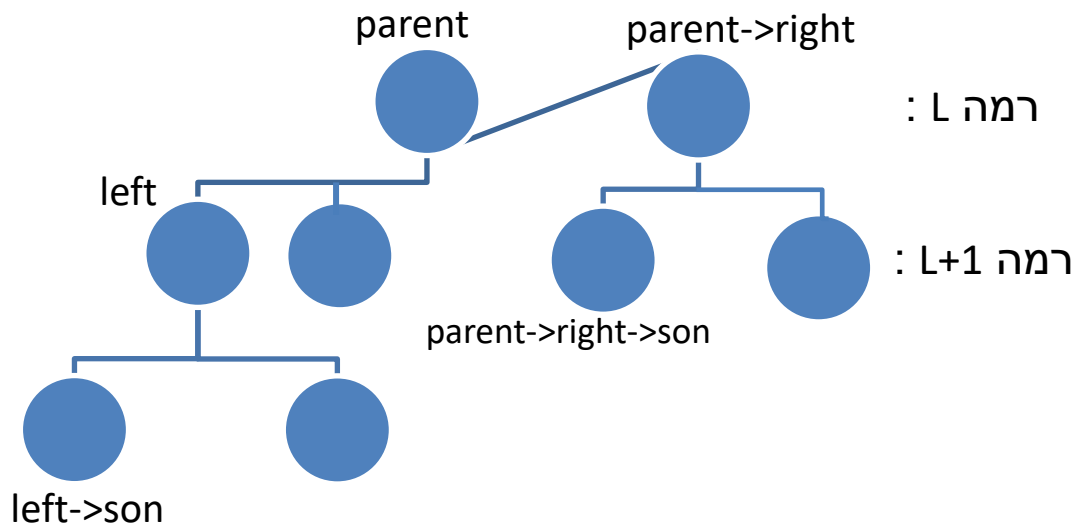
**בסיס** – ברמה  $L=0$  : השורש מודפס ראשון (לפני שמוציאים צומת מתור כלשהו) ואז מכניס את בנו השמאלי לתור Kids.

ברמה  $L=1$  : כל הצמתים ברמה 1 הם בנים של השורש. לאחר שהשורש הכניס את בנו השמאלי לתור Kids, בן זה מוצא מהתור ומכניס את האח שמימינו (במידה וישנו) לתור Brother. מאחר שהתור Brother אינו ריק, נטפל באח זה – שיכניס גם הוא את אחיו – וכך נמשיך עד שנסיים את כל האחים שיוכנסו ל-Brother, כלומר את כל הבנים של השורש. לסיכום – לא נוציא צומת מהתור Kids [שהוא כבר יהיה שייך לרמה 2], עד שנדפיס את כל הצמתים שברמה 1. ניתן לראות כי באופן זה הצמתים יודפסו לפי סדרם – משמאל לימין [שהרי משתמשים בתור – FIFO – ולכן הצמתים ייכנסו וייצאו לפי סדרם ברשימה המקושרת].

צעד האינדוקציה – נניח כי כל הצמתים מרמה L הודפסו משמאל לימין לפני שהוצא מהתור Kids בן ששייך לרמה L+1. בסיום ההדפסה, התור Brother ריק – כי אין עוד צמתים שהודפסו והאחים שלהם רק הוכנסו לתור ולא יצאו עדיין, שהרי כל הצמתים באותה רמה הודפסו. לעומת זאת, התור Kids אינו ריק, שהרי כל הצמתים מרמה L הכניסו אליו את בניהם השמאליים.

מאחר ו**כל הצמתים מרמה L הודפסו משמאל לימין** [הנחת האינדוקציה], הרי שהם גם הכניסו את בניהם השמאליים לתור Kids לפי סדרם משמאל לימין. לכן הבן הראשון בתור הוא הצומת השמאלי ביותר ברמה L+1. נסמן צומת זה ב-left. נסמן את אביו ב-parent, את האח שצמוד לאב מימין [במידה וישנו] ב-parent->right, ונסמן את הבן השמאלי של כל אחד מהם [במידה וישנו] ב-left->son, parent->right->son. **[ראו תרשים בעמוד הבא.]** כך, left יהיה הראשון מרמה L+1 שיודפס [ולכן גם הראשון שיכניס את בנו השמאלי left->son לתור Kids]. אם ל-left יש אחים – הוא יכניס את האח הימני שלו לתור Brother, ואז תור זה לא יהיה ריק, ולכן כל אחיו של left יטופלו לפני שייצא צומת נוסף מהתור Kids. לאחר הטיפול בכל האחים של left [במידה וישנם], התור Brother יתרוקן [לאח הימני ביותר אין אח ימני להכניס לתור זה]. לכן אז נוציא את הצומת הבא מהתור Kids, שהוא הבן השמאלי של האח הבא של parent שיש לו בנים, כלומר parent->right->son [ראו תרשים].

באופן זה, כל הצמתים מרמה  $L+1$  יודפסו משמאל לימין, לפני שנתחיל לטפל בצמתים מרמה  $L+2$ .  
מ.ש.ל.



סיבוכיות זמן : נעשה מעבר על כל צמתי העץ ובכל מעבר מתבצעות מספר סופי של פעולות, ולכן העלות הכוללת היא  $\Theta(n)$ .