
מבני נתונים

הרצאה מספר 7

עץ בינארי

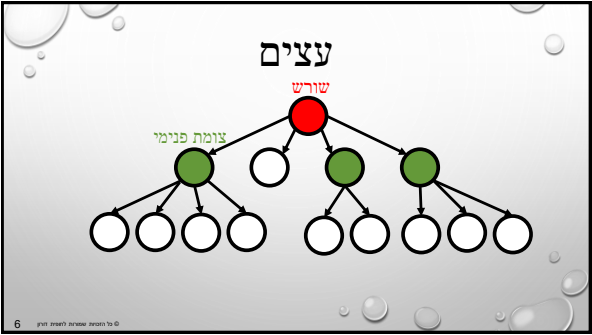
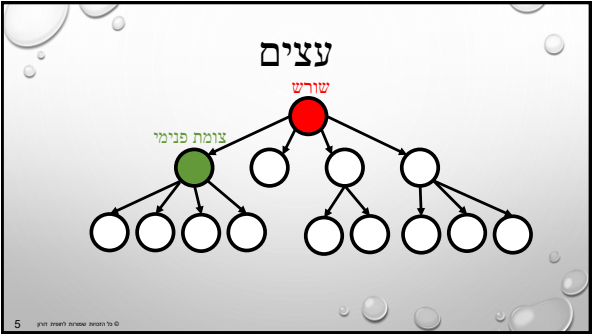
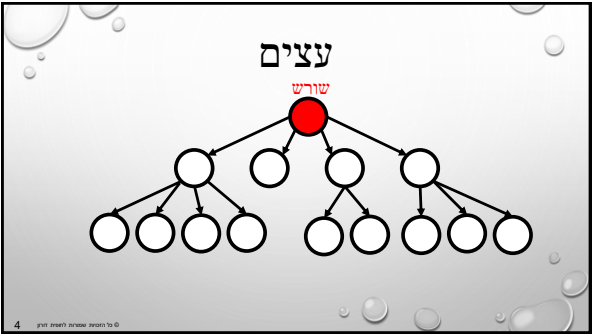
מרצה: חופית דורון

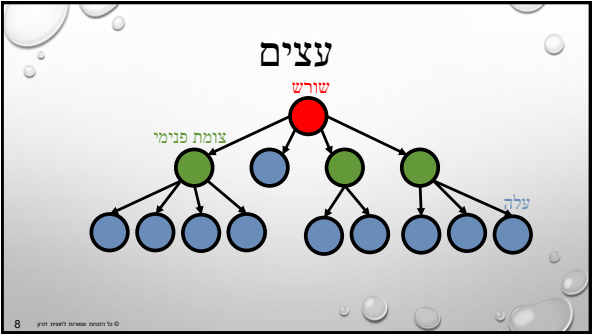
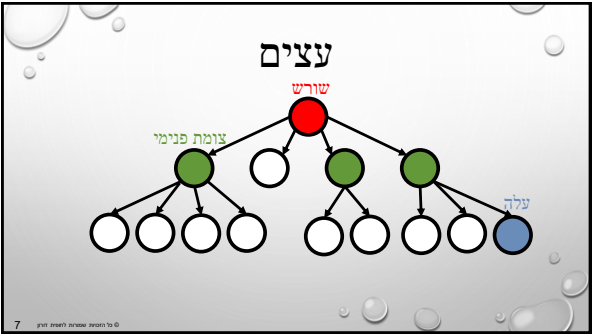
עצים

- עץ הוא מבנה נתונים היררכי חסר מעגלים.
- הקשרים בין הצמתים בעץ הם קשרי אב-בן.
- שורש העץ - צומת ללא אב שממנו ניתן להגיע לכל שאר הצמתים בעץ.
- צומת פנימי - צומת שיש לה אבא יחיד ויש לה בן אחד או יותר.
- עלה - צומת שיש לה אבא יחיד אך אין לה בנים.

עצים

```
graph TD; A(( )) --- B(( )); A --- C(( )); B --- D(( )); B --- E(( )); B --- F(( )); C --- G(( )); C --- H(( )); D --- I(( )); D --- J(( )); E --- K(( )); E --- L(( )); F --- M(( )); F --- N(( ))
```






עצים

- עומק/רמה – מספר הקשתות שיש לעבור בין שורש העץ לצומת.
- גובה העץ – המרחק המקסימלי בין השורש לצומת הנמצאת ברמה הגבוהה ביותר בעץ.
- דרגה של צומת – מספר הבנים שיש לצומת.
- צאצא – צומת A הוא צאצא של צומת B במידה וישנו מסלול בין B ל- A שבכל שלב במסלול ההתקדמות היא מאב לבן בלבד.
- אב קדמון – צומת A הוא אב קדמון של צומת B במידה וישנו מסלול בין A ל- B שבכל שלב במסלול ההתקדמות היא מאב לבן בלבד.

9

עץ בינארי

- עץ בינארי הוא עץ שלכל צמתיו יש 2 בנים לכל היותר.
- עץ בינארי מלא – עץ שלכל צומת פנימי שבו יש 2 בנים בדיוק.



- עץ בינארי שלם – עץ בינארי מלא שבו כל העלים בעלי אותו עומק.

© כל הזכויות שמורות לרשתות מחשבים תש"פ 10

עץ בינארי

- מבנה צומת בעץ בינארי:
- מפתח.
- DATA - המידע שעלינו לשמור (INT, FLOAT, STRUCT,...).
- מצביע לבן הימני.
- מצביע לבן השמאלי.
- מצביע לאב (לא בכל עץ בינארי יופי מצביע לאב).
- לעיתים המפתח הוא גם ה-DATA.

© כל הזכויות שמורות לרשתות מחשבים תש"פ 11

עץ בינארי

- מבנה צומת בעץ בינארי:
- מפתח.
- DATA - המידע שעלינו לשמור (INT, FLOAT, STRUCT,...).
- מצביע לבן הימני.
- מצביע לבן השמאלי.
- מצביע לאב (לא בכל עץ בינארי יופי מצביע לאב).
- לעיתים המפתח הוא גם ה-DATA.

© כל הזכויות שמורות לרשתות מחשבים תש"פ 12

עץ בינארי

➤ מבנה צומת בעץ בינארי:

```
typedef struct node
{
    int key;
    int data;
    struct node* right;
    struct node* left;
}Node;
```

13

© כל הזכויות שמורות לרשתות ערוץ

עץ חיפוש בינארי

- עץ חיפוש בינארי הוא עץ בינארי המקיים את הכלל הבא:
- כל צומת שערכה גדול מצומת האב – תהיה בתת העץ הימני.
- כל צומת שערכה קטן מצומת האב – תהיה בתת העץ השמאלי.
- עץ חיפוש בינארי מאפשר לבצע חיפוש \ הוספת של צומת לעץ מבלי לעבור על כל צמתיו.
- באמצעות עץ חיפוש בינארי נצטרך במקרה הגרוע לעבור על גובה העץ.
- כאשר עץ החיפוש הבינארי כמעט שלם\מאוזן גובה העץ יהיה $O(\log N)$.

14

© כל הזכויות שמורות לרשתות ערוץ

תרגיל

- כתבו פונקציה המקבלת מספר ומכניסה אותו לעץ החיפוש הבינארי.
- כתבו פונקציה המקבלת מספר ובודקת האם הוא נמצא בעץ.
- כתבו תוכנית המאפשרת בחירת פעולה בתפריט:
- הוספת מספר לעץ
- חיפוש מספר בעץ
- יציאה

15

© כל הזכויות שמורות לרשתות ערוץ

חיפוש צומת בעץ בינארי

- חיפוש ערך בעץ בינארי יתבצע על ידי השוואת הערך שאותו אנו מחפשים לערך הנמצא בשורש העץ, במידה והערך שווה נוכל להחזיר את הצומת \ להחזיר ערך TRUE.
- במידה והערך אותו אנו מחפשים גדול מהערך שנמצא בשורש נמשיך לבצע חיפוש בתת העץ הימני.
- במידה והערך אותו אנו מחפשים קטן מהערך הנמצא בשורש נמשיך לבצע חיפוש בתת העץ השמאלי.

16

© כל הזכויות שמורות לרשתות תורת

חיפוש צומת בעץ בינארי

- נמשיך בפעולת החיפוש על ידי התייחסות לתת העץ כעץ ששורשו הוא הבן הימני \ השמאלי.
- פעולת החיפוש תסתיים באחת מ-2 האפשרויות:
 - נמצא את האיבר שחיפשנו בעת ביצוע ההשוואה
 - נגיע לתת עץ ששורשו שווה ל NULL – משמעותו שהערך שחיפשנו איננו קיים בעץ.
- סיבוכיות הפעולה $O(N)$
- כאשר מדובר בעץ כמעט שלם/מאוזן הסיבוכיות היא $O(\log N)$.

17

© כל הזכויות שמורות לרשתות תורת

חיפוש צומת בעץ בינארי

```
Node* search(Node* head, int num)
{
    while (head!= NULL)
    {
        if (head->data == num)
            return head;
        if (num > head->data)
            head = head->right;
        else
            head = head->left;
    }
    return head;
}
```

18

© כל הזכויות שמורות לרשתות תורת

חיפוש צומת בעץ בינארי

```
bool search(Node* head, int num)
{
    while (head!= NULL)
    {
        if (head->data == num)
            return true;
        if (num > head->data)
            head = head->right;
        else
            head = head->left;
    }
    return false;
}
```

19

© כל הזכויות שמורות לרשתות תוכן

הוספת צומת בעץ בינארי

➤ על מנת לבצע הוספה של צומת לעץ בינארי, תחילה נבצע שלבי חיפוש של הצומת בעץ על מנת לאתר את המיקום המתאים להוספת הצומת.

➤ בכל שלב נשווה את הערך אותו נרצה להוסיף לערך הנמצא בשורש העץ.

➤ במידה והערך אותו נרצה להוסיף גדול מהערך שנמצא בשורש נמשיך להתקדם לתת העץ הימני.

➤ במידה והערך אותו נרצה להוסיף קטן מהערך שנמצא בשורש נמשיך להתקדם לתת העץ השמאלי.

20

© כל הזכויות שמורות לרשתות תוכן

הוספת צומת בעץ בינארי

➤ נמשיך בפעולה זו עד שנגיע לתת עץ שערכו NULL.

➤ ברגע שנגיע ל-NULL נמצא את המיקום אליו נבצע הוספה של הצומת.

➤ סיבוכיות הפעולה $O(N)$

➤ כאשר מדובר בעץ כמעט שלם\מאוזן הסיבוכיות היא $O(\log N)$.

21

© כל הזכויות שמורות לרשתות תוכן

הוספת צומת בעץ בינארי

```
Node* insert(Node* head, int num)
{
    Node* curr = head;
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = num;
    temp->right = NULL;
    temp->left = NULL;
    if (head == NULL)
    {
        head = temp;
        return head;
    }
    while((curr->data < num && curr->right!= NULL) || (curr->data >= num && curr->left!= NULL))
    {
        if (curr->data < num)
            curr = curr->right;
        else
            curr = curr->left;
    }
    if (curr->data < num)
        curr->right = temp;
    else
        curr->left = temp;
    return head;
}
```

22

הדפסת עץ בינארי

ישנן מספר אפשרויות להדפסת עץ בינארי:

- **PREORDER** – הדפסה של שורש העץ, לאחר מכן תת עץ שמאלי ובסוף תת עץ ימני (ההדפסה מתבצעת בצורה רקורסיבית).
- **INORDER** – הדפסה של תת העץ השמאלי, השורש, תת העץ הימני. הדפסה זו נותנת לנו הדפסה של כל הצמתי בעץ בצורה ממויינת.
- **POSTORDER** – הדפסה של תת העץ השמאלי, תת העץ הימני ולבסוף השורש.

23

PREORDER

```
void preorder(Node* head)
{
    if (head == NULL)
        return;
    printf("%d ", head->data);
    preorder(head->left);
    preorder(head->right);
}
```

24

INORDER

```
void inorder(Node* head)
{
    if (head == NULL)
        return;
    inorder(head->left);
    printf("%d ",head->data);
    inorder(head->right);
}
```

25

POSTORDER

```
void postorder(Node* head)
{
    if (head == NULL)
        return;
    postorder(head->left);
    postorder(head->right);
    printf("%d ",head->data);
}
```

26

מחיקת צומת בעץ בינארי

- על מנת לבצע מחיקה של צומת תחילה יש לחפשו.
- ישנם 3 מקרים למחיקה של צומת:
- הצומת הוא עלה (אין לו בנים).
- לצומת יש בן אחד.
- לצומת יש 2 בנים.

27

מחיקת צומת בעץ בינארי

מקרה 1 - הצומת הוא עלה:
➤ יש לעדכן את צומת האב שתצביע על NULL במקום על העלה.

מקרה 2 - לצומת יש בן אחד:
➤ יש לעדכן את צומת האב שתצביע לצומת הבן של הצומת שאותה אנו רוצים למחוק, במקום לצומת שאותה נרצה למחוק.

28 © כל הזכויות שמורות לרשתות ערוץ

מחיקת צומת בעץ בינארי

מקרה 3 - לצומת יש 2 בנים:
➤ אפשרות א':
➤ יש למצוא תחילה את הצומת העוקב לצומת שאותה נרצה למחוק.
מכיוון שמדובר בצומת בעלת 2 בנים - צומת העוקב בוודאות ימצא בתת עץ הימני, כלומר עלינו למצוא את הצומת המינימלי בתת העץ הימני.
➤ נבצע החלפה בין הצומת העוקב לצומת שאותה אנו רוצים למחוק.
➤ כעת נבצע מחיקה של הצומת.
(שימו לב - כאשר מדובר בצומת שאין לה תת עץ ימני העוקב יכול להיות אחד מהאבות הקדמונים של הצומת.)

29 © כל הזכויות שמורות לרשתות ערוץ

מחיקת צומת בעץ בינארי

מקרה 3 - לצומת יש 2 בנים:
➤ אפשרות ב':
➤ יש למצוא תחילה את הצומת הקודם לצומת שאותה נרצה למחוק.
מכיוון שמדובר בצומת בעלת 2 בנים - צומת הקודם בוודאות ימצא בתת עץ השמאלי, כלומר עלינו למצוא את הצומת המקסימלי בתת העץ השמאלי.
➤ נבצע החלפה בין הצומת הקודם לצומת שאותה אנו רוצים למחוק.
➤ כעת נבצע מחיקה של הצומת.
(שימו לב - כאשר מדובר בצומת שאין לה תת עץ שמאלי הקודם יכול להיות אחד מהאבות הקדמונים של הצומת.)

30 © כל הזכויות שמורות לרשתות ערוץ

מציאת מינימום ומקסימום

- הצומת המינימלי בעץ בינארי תמיד יהיה האיבר הכי שמאלי בעץ.
- כלומר אם נתקדם מהשורש בכל פעם לתת העץ השמאלי (שמכיל ערכים הקטנים מהשורש) עד שנגיע לצומת שהבן השמאלי שלו הוא NULL צומת זה הוא הצומת המינימלי בעץ.
- באופן סימטרי הצומת המקסימלי בעץ בינארי תמיד יהיה האיבר הכי ימני בעץ.
- כלומר אם נתקדם מהשורש בכל פעם לתת העץ הימני (שמכיל ערכים הגדולים מהשורש) עד שנגיע לצומת שהבן הימני שלו הוא NULL צומת זה הוא הצומת המקסימלי בעץ.

31

© כל הזכויות שמורות לרשתות תוכן

מציאת מינימום

```
Node* min(Node* head)
{
    if (head==NULL)
        return NULL;
    while (head->left!=NULL)
        head = head->left;
    return head;
}
```

32

© כל הזכויות שמורות לרשתות תוכן

מציאת מקסימום

```
Node* max(Node* head)
{
    if (head==NULL)
        return NULL;
    while (head->right!=NULL)
        head = head->right;
    return head;
}
```

33

© כל הזכויות שמורות לרשתות תוכן

מחיקת צומת בעץ בינארי

```
Node* deleteLeaf(Node* head, Node* prev, Node* curr)
{
    if (head == curr)
    {
        free(curr);
        return NULL;
    }
    if (prev->left == curr)
        prev->left = NULL;
    else
        prev->right = NULL;
    free(curr);
    return head;
}
```

34 © כל הזכויות שמורות לידידי

מחיקת צומת בעץ בינארי

```
Node* deleteOneSubtree(Node* head, Node* prev, Node* curr)
{
    if (head == curr)
    {
        if (head->left != NULL)
            head = head->left;
        else
            head = head->right;
    }
    else if (prev->left == curr)
    {
        if (curr->right == NULL)
            prev->left = curr->left;
        else
            prev->left = curr->right;
    }
    else
    {
        if (curr->right == NULL)
            prev->right = curr->left;
        else
            prev->right = curr->right;
    }
    free(curr);
    return head;
}
```

35 © כל הזכויות שמורות לידידי

מחיקת צומת בעץ בינארי

```
Node* deleteTwoSubtrees(Node* head, Node* curr)
{
    Node* temp;
    int templ;
    Node* prev = head;
    temp = min(curr->right);
    while (prev->right != temp && prev->left != temp)
    {
        if (prev->data < temp->data)
            prev = prev->right;
        else
            prev = prev->left;
    }
    templ = temp->data;
    if (temp->left == NULL && temp->right == NULL)
        head = deleteLeaf(head, prev, temp);
    else
        head = deleteOneSubtree(head, prev, temp);
    curr->data = templ;
    return head;
}
```

36 © כל הזכויות שמורות לידידי

מחיקת צומת בעץ בינארי

```
Node* delete(Node* head, int num)
{
    Node* curr,temp1,temp2;
    int tempNum;
    Node* prev = head;
    curr = searchNode(head,num);
    if (curr == NULL)
        return head;
    if (curr->left!=NULL && curr->right!=NULL)
    {
        head = deleteTwoSubtrees(head,curr);
        return head;
    }
}
```

37

מחיקת צומת בעץ בינארי

```
while(prev->right!=curr && prev->left!=curr)
{
    if (prev->data < num)
        prev = prev->right;
    else
        prev = prev->left;
}
if (curr == head)
    prev = NULL;
if (curr->left==NULL && curr->right==NULL)
    head = deleteLeaf(head,prev,curr);
else
    head = deleteOneSubtree(head,prev,curr);
return head;
```

38

תרגילים

ציינו נכון/לא נכון במידה והטענה נכונה הוכיחו אותה אחרת הביאו דוגמא נגדית.

➤ סדר מחיקת איברים בעץ חיפוש בינארי לא משפיעה על העץ הסופי. כלומר אם מוחקים קודם את A ואחר כך את B, מקבלים אותו עץ כאשר מוחקים את B ואחר כך את A.

➤ במעבר PREORDER על עץ חיפוש בינארי, האיבר הראשון תמיד הערך הכי קטן.

➤ כל עץ חיפוש בינארי הינו מאוזן ? (עץ שבו ההפרש בין הגובה של שני תתי-עצים של אותו הצומת לעולם אינו גדול מאחד)

39

תרגילים

➤ כתבו פונקציה המקבלת 2 פרמטרים: עץ חיפוש בינארי ומספר שלם K התוכנית תחזיר את האיבר ה- K בגודלו בעץ.

שימו לב סיבוכיות החזרת האיבר ה- K בגודלו תהיה $O(N)$.

➤ כתבו פונקציה המקבלת כפרמטר עץ חיפוש בינארי ומחזירה עץ שכל צומת שבו תכיל את סכום כל הצמתים שקטנים ממנה.

© כל הזכויות שמורות לרשת 40