

פרק 10 עץ בינרי

1. הזמן הנדרש

12 שעות.

2. מטרות

1. הכרת מבנה נתונים חדש (היררכי) – עץ בינרי.
2. פעולות רקורסיביות על עץ בינרי. מעברים לעומק ולרוחב.
3. שימושים בעץ בינרי – עץ ביטוי.
4. טיפוס נתונים מופשט – עץ חיפוש בינרי.

3. מבנה הפרק

מבוא כללי

הקו, שהחל בפרקים 6–7 עם הצגת החוליה ומבנים משורשרים הנבנים על בסיס החוליה, ממשיך בפרק זה עם הצגת הרעיון של מבנה נתונים היררכי. מבנה שכזה הוזכר לראשונה בפרק 6 כאשר הוצג מבנה ארגוני על בסיס המחלקה Employee, אך בפרק זה אנו מגדירים את המבנה הכללי ביותר של עץ ומתעמקים בו. אחר כך אנו מגבילים את הדיון לעץ בינרי, ומראים כיצד ניתן לייצגו כמבנה מבוסס חוליות. איננו בונים מחלקה עוטפת לצורך הגדרת עץ בינרי. מכאן שגם העץ הבינרי הוא מבנה נתונים המייצג אוסף ואינו טיפוס נתונים מופשט, בדיוק כפי שהייתה שרשרת החוליות. בהמשך אנו מציגים את הארגון המיוחד של עץ-חיפוש-בינרי. בתרגילים נציע מחלקה עוטפת שתגדיר טיפוס נתונים מופשט לעץ-חיפוש-בינרי.

א – עצים

- במבוא ובסעיף א מוצג מושג העץ, כמבטא מבנה היררכי מוכר מחיי היום יום, ונעשית הכרת המושגים הראשונים הקשורים לעץ. בעבר נהגנו לכנות את הצמתים בעץ במונחים אבות ובנים. בפרק הנוכחי אנו מקבלים את הגישה המקובלת כיום ומכנים את ה"אב" בשם הכללי "הורה" ואת ה"בן" בשם הכללי "ילד".
- אנו מגדירים שורש כצומת שאין לו הורה. אך ניתן להתייחס לכל צומת בעץ כשורש של תת-עץ, על ידי ההתעלמות מכל הצמתים שאינם בתת-עץ, ובפרט מההורה שלו.

ב – עצים בינריים

- סעיף זה מתמקד בעץ הבינרי ומרחיב בהכרת המושגים הרלוונטיים. הרעיון המרכזי בסעיף זה הוא הבהרת התחושה שמדובר במבנה רקורסיבי מעצם הגדרתו.

- מוזכר בסעיף שקיימת שיטה לייצוג עצים כלליים בעזרת עצים בינריים. אם התלמידים יסתקרנו לגבי השיטה, ניתן להסביר בקצרה כי: בשיטה זו כל צומת מחזיק הפניה לילדו השמאלי בלבד, ולאח שלו עצמו.

ג – חוליה בינרית

- החוליה הבינרית היא הכללה פשוטה של החוליה האונרית, ואף השימוש והמימוש שלהן דומה מאוד. החוליה הבינרית זהה כמעט לחלוטין לחוליה עם שתי הפניות שבעזרתה יצרנו שרשרת דו-כיוונית. אנו מגדירים כאן מחלקה של חוליות, כאשר כל חוליה מכילה שדה נתונים ושני מצביעים לחוליות אחרות מאותו טיפוס (או ל-null). ההבדל היחיד הוא בשמות התכונות (left ו-right, לעומת next ו-prev).
- ניתן להשתמש בחוליות כאלה לבניית עצים בינריים, כפי שאנו עושים בפרק זה, לבניית רשימות משורשרות דו-כיוונית, או לבניית מגוון גדול של מבנים אחרים שלא זכו לשמות מפורשים. שימו לב כי העובדה שקראנו למחלקה ולשדותיה בשמות הלקוחים מעולם העצים אינה קובעת דבר לגבי מהות המחלקה והשימושים האפשריים בה. מכאן ששימוש בעצמים ממחלקה זו לבניית עצים בינריים היא החלטה שלנו כמשתמשים במחלקה. והאחריות שהמבנים שנבנה יהיו עצים, היא שלנו בלבד. כך, הפרק מציג בניית מבני עצים בינריים על ידי שימוש בעצמים ממחלקה זו ומימוש פעולות עליהם. הדגש הוא על הבנת המבנה של עץ בינרי, ועל פיתוח יכולת הפתרון של בעיות על עצים כאלה.
- בשני המקרים (בחוליה אונרית ובחוליה בינרית), הממשק מכיל רק פעולות על צומת אחד (חוליה אחת). את הסיבה לכך כבר הזכרנו: חוליה כשלעצמה אינה שרשרת ואינה עץ. לכן הפעולות החוליה מתייחסות רק אליה. פעולות המתייחסות למבנים הבנויים מחוליות אין מקומן בתוך מחלקת החוליה.
- מימוש החוליה הבינרית. התלמידים צריכים להחזיק בידיהם את המחלקה BinTreeNode לצורך מימושים ופתרונות מתקדמים בהמשך. קוד המחלקה אמנם מופיע בספר הלימוד, והמחלקה המקומפלת נמצאת ב-Unit4 / unit4 שבידי התלמידים, אך כדאי וראוי שהתלמידים יממשו את המחלקה כולה בעצמם. כך יוכלו בהמשך להיכנס לקוד המחלקה ולשנותו על פי שאלות ותרגילים שיעסקו בייצוג ובשיפורי יעילות של העץ הבינרי.

ד – עץ חוליות בינרי

- סעיף זה מדגיש את השימוש בהפניות כבסיס להגדרת המבנה (דומה לרשימה, אך מורכב ממנה).
- כפי שבנינו שרשרת חוליות באמצעות חוליה אונרית, כך אנו בונים עץ חוליות בינרי באמצעות חוליה בינרית. בשרשרת, התנאי לכך שזו שרשרת (ולא מעגל למשל) הוא שהחוליה העוקבת לחוליה היא תמיד חוליה חדשה, שלא הופיעה קודם בשרשרת. תנאי הנכונות לעץ בינרי מכליל דרישה זו. בניית עץ וטיפול במבנהו הם לגמרי באחריות התוכנית היוצרת את המבנה.

- בשני המקרים (בשרשרת החוליות ובעץ) לא בנינו מחלקה עבור האוסף אלא בנינו מבנים באמצעות החוליות. אנו מכלילים כאן את הרעיון שעל ידי שימוש בחוליות ניתן לבנות מבנים משורשרים. הדיון בעץ בינרי כמבנה נתונים הבנוי מחוליות מאפשר להתרכז בגישה רקורסיבית לפעולות על העץ. גם בשרשרת חוליות דנו מעט בגישה רקורסיבית לתכנות פעולות. בעץ בינרי המבנה הרקורסיבי בולט אף יותר, ולכן הרחבנו והדגשנו אותו כאן כגישה עיקרית לתכנות פעולות על עץ. כמובן שיכולנו לעשות זאת גם עבור שרשרת חוליות, אך אנו העדפנו לעסוק בכל אחד מהמבנים בגישה שונה, וכך להציג בפני התלמידים הן גישה איטרטיבית והן גישה רקורסיבית לתכנות של מבנים שכאלה.
- יש להדגיש מדי פעם במהלך ההוראה, שכל חוליה בעץ היא שורש: שורש העץ כולו או שורש מקומי (של תת-עץ). כל אלה הם מסוג החוליה הבסיסית BinTreeNode. כיוון שאיננו עוטפים את העץ בשום מעטפת חיצונית, אלא מטפלים ישירות בחוליות, יכול להיות שיעלו שאלות בכיתה שעיקרן הזהות בין תת-עץ לחוליה. יש מקום לאותן שאלות גם בטיפול בשרשרת חוליות, וייתכן שחלקן עלו כבר אז. התשובה כאן ושם היא שחוליה בינרית משמשת אותנו ככניסה לטיפול בתת-עץ שהיא שורשו.
- העץ הריק. כמו בשרשרת חוליות, גם מבנה חוליות המייצג עץ בינרי אינו יכול להיות ריק. העץ המינימלי מכיל צומת אחת לפחות אילו היינו משתמשים במעטפת, היינו יכולים לייצג עץ ריק, אך המחיר היה, כפי שצוין לעיל, סיבוך של הפעולות הרקורסיביות. אנו מדברים בהמשך על הפנייה מטיפוס עץ שערכה null (זה יהיה תנאי בדיקה ועצירה לפעולות שונות. הפניה כזו אינה מייצגת "עץ ריק". המושג אינו קיים במבנה החוליות שבחרנו. גם בפעולות החיצוניות המקבלות עץ תתקיים ההנחה הסמויה המקובלת ביחידה שלא ניתן לשלוח פרמטר שערכו null במקום שבו הפרמטר מייצג עץ בינרי.
- כמובן שבתכנות מונחה עצמים, מבנים שיש בהם שימוש מעניין צריכים להיות ממומשים במחלקה, עם פעולותיהם. למשל, שרשרת חוליות שימשה כמבנה לייצוג מחסנית, תור ורשימה. איננו מציגים בפרק מחלקה עוטפת לעץ, אנלוגית למחלקה רשימה אף שניתן לעשות זאת. כמו הרשימה, גם מחלקה כזו לא תהיה טיפוס נתונים מופשט (כמוסבר להלן). אין בהצגת מחלקה זו רעיון מעניין חדש. העדפנו להתרכז ברעיון המעניין והשימושי יותר של עץ-חיפוש-בינרי.
- נניח שנבנה מחלקת עץ העוטפת שרשרת חוליות (אנלוגית למחלקה רשימה) – האם היא תגדיר טיפוס נתונים מופשט? אם הייתה מעטפת כזו, אז פעולות כמו `getLeft() / GetLeft()`, `getRight() / GetRight()` היו צריכות להחזיר הפניה למקום בעץ, כלומר הפניה לחוליה בינרית בעץ. מכאן שהייצוג הפנימי היה חשוף. לכן, מתכנת יכול לשנות את מבנה העץ על ידי שימוש בפעולות ה- `set / Set` שבמחלקה חוליה, ולפגוע בשגגה במבנה התקין של העץ. למשל, אפשר לשנות את ההפניה השמאלית של חוליה בעץ, כך שתפנה לשורש העץ, וכך יתקבל מבנה מעגלי. ניתן גם להציב את התת-עץ הימני במקום השמאלי, וכך לקבל מבנה שבו לצומת יש שני תתי-עץ שהם אותו העץ. היכולת לפגוע במבנה העץ פירושה שאין הוא מהווה טיפוס נתונים מופשט אלא רק מבנה נתונים. ניתן היה "להגן" על מבנה העץ התקין על ידי השמטת הפעולות `set / Set` מממשק החוליה הבינרית, אך הנזק הנלווה היה חוסר יכולת לבנות עצים באמצעות

המחלקה. נזכיר: המחלקה רשימה העוטפת את השרשרת אינה טיפוס נתונים מופשט, כיון שהיא כוללת את כל הפעולות על שרשרת, ללא הגבלה. לעומת זאת, מחלקות המאפשרות רק גישה מוגבלת, המחסנית והתור, הן כן טיפוסיות נתונים מופשטים. כך גם בעצים בינריים: מחלקה עוטפת שתכיל את כל הפעולות בהכרח חושפת את ייצוג המקום, ולא תהיה טיפוס נתונים מופשט. מחלקה העוטפת עץ-חיפוש-בינרי מאפשרת רק גישה מוגבלת לעץ שבו משתמשים לייצוגה, אינה חושפת את ייצוג המקום והיא אמנם מגדירה טיפוס נתונים מופשט. אין אנו מציגים בפרק מחלקה כזו, אך בפרק הבא 11 – מפה, אנו למעשה עוסקים, בין השאר גם בה. יש להתעכב ולוודא שהרעיון הבא מובן לחלוטין: כל מה שאנו מגדירים ומבצעים על עץ בשלב הראשון, כאשר אנו מציגים את הממשק, הן למעשה פעולות מקומיות על הצומת הנוכחי. איננו דנים בפעולות המטפלות בכלל המבנה. לכן גם יעילות כל אחת מהפעולות היא $O(1)$ שכן היא מתבצעת במקום.

תשובה לשאלת המחשבה המופיעה בסעיף זה:

? עץ עלה הוא העץ הקטן ביותר שיכול להתקיים. פעמים רבות ניעזר בבדיקה האם עץ מסוים הוא עץ עלה. כתבו קטע קוד הבודק האם חוליה בינרית נתונה היא עץ עלה.

תשובה:

בג'אווה:

```
public static boolean isLeaf(BinTreeNode<Integer> bt)
{
    return (bt.getLeft() == null) && (bt.getRight() == null);
}
```

בסישרפ:

```
public static bool IsLeaf(BinTreeNode<int> bt)
{
    return (bt.GetLeft() == null) && (bt.GetRight() == null);
}
```

ה – עץ חוליות בינרי – מבנה רקורסיבי

ההגדרה הבסיסית לעץ חוליות בינרי היא שהוא בעל חוליה אחת לפחות. פירוש הדבר הוא שלא קיים "עץ חוליות ריק", כלומר לא יכול להיות שהפניה לעץ חוליות בינרי תהיה שווה `null`. אי לכך היינו צריכים לכתוב פעולות רקורסיביות על עצים כך שתנאי העצירה יהיה חוליה בינרית יחידה:

למשל,

בג'אווה:

```
public int numOfNodes (BinTreeNode<Integer> tr)
{
    int leftCount = 0;
    int rightCount = 0;
    if (tr.getLeft() != null)
        leftCount = numOfNodes(tr.getLeft());
    if (tr.getRight() != null)
        rightCount = numOfNodes(tr.getRight());
    return (leftCount + rightCount + 1);
}
```

בסישרפ:

```
public int numOfNodes (BinTreeNode<int> tr)
{
    int leftCount = 0;
    int rightCount = 0;
    if (tr.GetLeft() != null)
        leftCount = numOfNodes(tr.GetLeft());
    if (tr.GetRight() != null)
        rightCount = numOfNodes(tr.GetRight());
    return (leftCount + rightCount + 1);
}
```

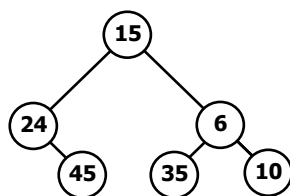
כיוון שצורת כתיבה זו היא מורכבת מאוד, בחרנו בדרך הכתיבה המוצגת בפרק. בכתיבה זו התנאי הראשון הנבדק הוא שוויון ההפניה לחוליה ל-`null`. דבר זה יכול לבלבל מעט, שכן הוא עלול להתפרש כאילו יכול להיות שהעץ שהועבר לפעולה אינו קיים (ערך הפרמטר הוא `null`). דבר זה גם עומד בניגוד להנחה הסמויה המלווה את היחידה, שפרמטר המועבר לפעולה אינו שווה `null`, אלא אם צוין אחרת.

יש להסביר לתלמידים שתנאי זה אינו יכול להתקיים בעת הזימון הראשון של הפעולה בדיוק מטעמים אלה: פרמטר שהועבר לפעולה אינו שווה `null`, והעץ בוודאי מכיל לפחות חוליה אחת. אך ניסוח זה של תנאי העצירה מתאים ל**סוף הרקורסיה** שם ודאי ניעצר במצב של הפניה לתת-עץ השווה ל-`null`.

תשובה לשאלת המחשבה המופיעה בסעיף זה:

? בצעו מעקב רקורסיבי על העץ שלפניכם. הפעילו עליו את הפעולות:

`numNodes(...)` / `PrintNodes(...)` ו-`printNodes(...)` / `PrintNodes(...)` וכתבו מה מתקבל.



תשובה:

ספירת הצמתים תחזיר את הערך 6.

הדפסת הערכים השמורים בעץ תחזיר את הסדרה: 10 35 6 45 24 15.

ו – מעברים על עץ בינרי

- מעברים על עץ בינרי. הבנה ראשונית של המעברים תיעשה בעזרת האיור. תבניות הסריקה ישמשו אותנו לפתרון בעיות רבות העוסקות בעצים, ראו מבחר ראשוני בסעיף ו בפרק. יש להבין את יעילות הסריקות. ניתן לקטוע את רצף הלימוד ולתרגל את הנלמד בעזרת התרגילים המתאימים המופיעים בסוף הפרק.
- בחישובי היעילות אנו מתייחסים לביקור בצומת, אך יש לשים לב ולציין לפני התלמידים שפרט למעבר בצומת בעת ביצוע הביקור בו, אנו חולפים בו עוד פעם או פעמיים בדרך להורה שלו או לילדיו.

תשובות לשאלות המחשבה המופיעות בסעיף זה:

? באילו מהדוגמאות שהבאנו לעיל יש חשיבות לסדר הביקור בצמתים, ובאילו אין?

תשובה:

כאשר נרצה לבדוק האם איבר מסוים נמצא בעץ, אין חשיבות לסדר הסריקה. כאשר נרצה להדפיס את עץ המשפחה לפי סדר הדורות, יש חשיבות לסדר הסריקה.

? עקבו אחרי סריקה בסדר סופי של העץ המופיע באיור לעיל, וכתבו את סדרת הערכים המתקבלת.

תשובה:

סדרת הערכים המתקבלת בסריקה בסדר סופי היא (משמאל לימין): E Z Y G D H A

? ממשו את הפעולות `postorderString(...)` / `PostorderString(...)` ו-`inorderString(...)` / `InorderString(...)`

תשובה:

בג'אוה:

```

public static String postorderString(BinTreeNode<String> bt)
{
    if (bt == null)
        return "";
    return postorderString(bt.getLeft()) +
        postorderString(bt.getRight())+ bt.getInfo() + " " +;
}

public static String inorderString(BinTreeNode<String> bt)
{
    if (bt == null)
        return "";
    return inorderString(bt.getLeft()) + bt.getInfo() + " "
        +inorderString(bt.getRight());
}

```

בס'שרפ:

```

public static string PostorderString(BinTreeNode<string> bt)
{
    if (bt == null)
        return "";
    return PostorderString(bt.GetLeft()) +
        PostorderString(bt.GetRight())+ bt.GetInfo() + " " +;
}

public static string InorderString(BinTreeNode<string> bt)
{
    if (bt == null)
        return "";
    return InorderString(bt.GetLeft()) + bt.GetInfo() + " "
        +InorderString(bt.GetRight());
}

```

? ממשו פעולה בשם levelOrderString המקבלת עץ חוליות בינרי של מחרוזות, ומחזירה מחרוזת

המתארת את תוכן העץ המסודר לפי רמות העץ.

כדי לכתוב את הקוד יהיה עליכם להגדיר תור של חוליות בינריות מטיפוס מחרוזת. נזכור כי גם החוליה הבינרית וגם התור הם מחלקות גנריות, ויש לקבוע טיפוס קונקרטי לכל אחת מהן. כלומר, את הטיפוס של החוליה הבינרית עלינו לכתוב עבור הטיפוס הקונקרטי מחרוזת. הטיפוס המתקבל הוא הטיפוס הקונקרטי לתור. הגדרת התור תיראה כך:

```
Queue<BinTreeNode<String>>
```

:תשובה

:בג'אוה

```
public static String levelOrderString(BinTreeNode<String> bt)
{
    String str = "";
    BinTreeNode<String> node;
    Queue<BinTreeNode<String>> q =
        new Queue<BinTreeNode<String>>();

    q.insert(bt);

    while (!q.isEmpty())
    {
        node = q.remove();
        str = str + node.toString() + ",";
        if (node.getLeft() != null)
            q.insert(node.getLeft());
        if (node.getRight() != null)
            q.insert(node.getRight());
    }

    return str;
}
```

:בסירפ

```
public static string LevelOrderString(BinTreeNode<string> bt)
{
    string str = "";
    BinTreeNode<string> node;
    Queue<BinTreeNode<string>> q =
        new Queue<BinTreeNode<string>>();

    q.Insert(bt);

    while (!q.IsEmpty())
    {
        node = q.Remove();
        str = str + node.ToString() + ",";
        if (node.GetLeft() != null)
            q.Insert(node.GetLeft());
        if (node.GetRight() != null)
            q.Insert(node.GetRight());
    }

    return str;
}
```

ז – שימוש בעץ חוליות בינרי – ייצוג ביטוי חשבוני

- ההתייחסות לעץ ביטוי תיעשה בעזרת הגדרה של פעולות שיקבלו עץ קונקרטי ויפעלו עליו. יש לשים לב שהביטוי נתון כמחרוזת, ויהיה צורך לבצע פירוק של המחרוזת כדי לבנות את העץ. ההנחות המתייחסות לביטוי מקילות את ביצוע המשימה. לגבי ייצוג העץ, שימו לב שעלינו

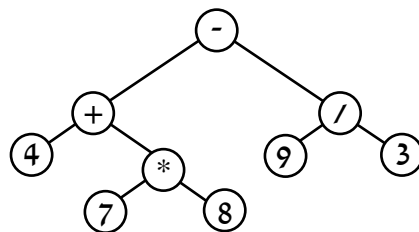
להשתמש באותו סוג חוליה הן לעלי העץ המייצגים מספרים, והן לצמתים הפנימיים המייצגים פעולות. גישה טבעית (אך לא יחידה) היא לייצג הן את הערכים והן את הפעולות כתווים (או כמחרוזות). לכן, לחישוב הביטוי, יש צורך ל"תרגם" את הערכים למספרים או לפעולות.

- זוהי דוגמה יישומית לשימוש במבנה הנתונים עץ בינרי. כך אכן פועלים קומפילרים בבואם לטפל בביטויים. במהלך השיעור יש להבין מדוע ההגדרה הפורמלית של ביטוי מרמזת על היכולת להציג אותו בעזרת עץ בינרי. יש להבין את הנושא ברמה האלגוריתמית הכללית. תרגיל 23 בסוף הפרק מתייחס לנושא זה ומתרגל אותו.

תשובות לשאלות המחשבה המופיעות בסעיף זה:

? ציירו את עץ הביטוי עבור: $((4 + (7 * 8)) - (9 / 3))$

תשובה:



? ממשו את האלגוריתם **חשב-ערך-ביטוי** כפעולה בשם `computeExpTree(...)`.

תשובה:

הקוד הבא נעזר בפעולה המוצאת האם עץ הוא עץ עלה.

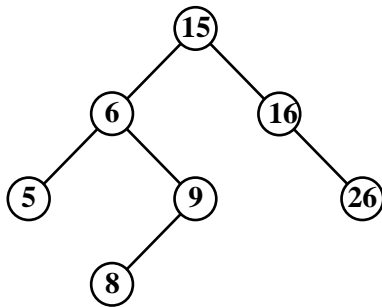
```
public static int computeExpTree (BinTreeNode<Character> tree)
{
    if (BinTreeUtils.isLeaf(tree))
    {
        return Integer.valueOf(String.valueOf(tree.getInfo()));
    }
    int leftVal = computeExpTree(tree.getLeft());
    int rightVal = computeExpTree(tree.getRight());
    char ch = tree.getInfo();
    if (ch == '+')
        return (leftVal + rightVal);
    if (ch == '-')
        return (leftVal - rightVal);
    if (ch == '*')
        return (leftVal * rightVal);
    if (ch == '/')
        return (leftVal / rightVal);
    return -1; //לא אמורים להיות כאן בקלט תקין
}
```

ח – עץ-חיפוש-בינרי

- עץ-חיפוש-בינרי, כפי שמוצג בפרק, הוא עץ בינרי המאורגן כך שבין איבריו מתקיים סדר מסוים. כפי שכבר אמרנו, איננו מגדירים מחלקה בשם זה, אך מחלקה עם פונקציונליות כזו המשתמשת בעץ-חיפוש-בינרי לייצוג נדונה בפרק הבא.

• ח.4. הוצאת ערך מעץ-חיפוש-בינרי

הפירוט הבא מיועד רק למורים ולתלמידים מתעניינים ובהחלט ניתן לדלג עליו: מכיוון שהאלגוריתם של פעולת ההוצאה אינו טריוויאלי ומימוש מורכב, נמנענו מלהתעמק בו בפרק. נסביר את מהלך פעולת ההוצאה מעץ-חיפוש-בינרי בעזרת האיור הזה:



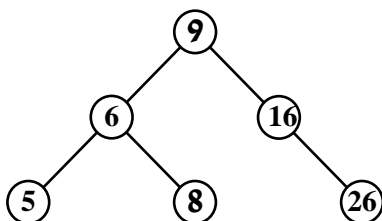
נניח כי אנו רוצים להוציא את הערך 8 מהעץ. ראשית נבצע חיפוש שימצא את הצומת שבו נמצא הערך. בדיקה מגלה כי זהו צומת עלה. אם כך ניתן למחקו מהעץ, על ידי החלפת ההפניה אליו מן ההורה (הצומת המכיל 9) ב-**null**.

נניח כי אנו רוצים להוציא את הערך 9. לאחר שהגענו לצומת המכיל אותו, אנו בודקים ומגלים כי יש לו רק תת-עץ שמאלי. כיוון שכך, נשנה את ההפניה בהורה של הצומת המכיל 9 (זהו הצומת המכיל 6), כך שתפנה אל תת-עץ זה (כלומר אל הצומת המכיל 8). כך הוצאנו את הערך 9 מהעץ, תוך שמירה מלאה על מבנהו ותכונותיו כעץ-חיפוש-בינרי. הוצאת הערך 16 תתבצע באופן דומה (תוך החלפת שמאל בימין).

נדון עכשיו במקרה הקשה ביותר: נניח כי אנו רוצים להוציא את הערך 15. לצומת המכיל אותו יש שני תת-עצים, ולפיכך הגישה הקודמת אינה ישימה.

פתרון אפשרי אחד הוא זה המתואר בקצרה בפרק, ונחזור עליו במפורט: נחליף את ההפניה בהורה לתת-עץ השמאלי (כמו במקרה הקודם). אחר כך נכניס את כל הערכים שבתת-עץ הימני לעץ, אחד-אחד. ברור שזה פתרון נכון, אך לא יעיל.

פתרון יעיל יותר למקרה זה: נחפש את הערך הימני ביותר בתת-העץ השמאלי של הצומת המועמד למחיקה. בעץ הנתון זהו הערך 9, הנמצא בצומת שאין לו תת-עץ ימני. נוציא את הצומת הזה מהעץ, כפי שתיארנו קודם. עכשיו, נחליף את ערך הצומת שברצוננו למחוק בערך שמצאנו, כלומר במקום ה-15 ייכתב 9. הגענו לעץ שהוא בדיוק מה שרצינו: עץ-חיפוש-בינרי, המכיל את כל הערכים שבעץ הקודם, פרט ל-15. אפשר כמובן, מטעמי סימטריה, להחליף בפתרון זה את "שמאלי" ב-"ימני":



ולבסוף, וריאציה על ההצעה האחרונה: הערך השמאלי ביותר בתת-העץ הימני של צומת נתון הוא זה העוקב לו בסריקה בסדר תוכי. אם כך, נבצע סריקה בסדר תוכי מהצומת שהערך בו מיועד להחלפה. כשנגיע לערך העוקב לזה שבצומת שבו התחלנו, נכניס אותו לאותו צומת, במקום הערך שברצוננו להוציא. עכשיו יש לפנינו אותה בעיה בצומת שאליו הגענו בסריקה: הוצאנו ממנו את הערך, ויש להכניס בו ערך אחר מהעץ. נפתור בעיה זו באותה גישה. כך קיבלנו אלגוריתם רקורסיבי (פחות יעיל מההצעה הקודמת).

• ח.5. יעילות הפעולות על עץ-חיפוש-בינרי

אנו מדברים על יעילות העץ בהנחה שהעץ קרוב לעץ מאוזן, ולא מחשבים את היעילות על פי המקרה הגרוע שהוא עץ לא מאוזן. כיוון שעד כה לימדנו את התלמידים לנתח יעילות על פי המקרה הגרוע, יש להסביר מדוע חרגנו ממנהגנו. אמנם איננו עוסקים בפועל באיזון עצים עקב הקושי של האלגוריתם, אך משום שבאופן עקרוני ניתן לאזן את העץ, אפשר לנתח את היעילות בהנחה שהעץ מאוזן (בחישובים האמיתיים כמובן שחייבים להוסיף את מחיר האיזון).

ניתן להסביר את יעילות הפעולות בצורה מדויקת פחות, אך אינטואיטיבית יותר:

- ברמה האחרונה של העץ יש 2^k צמתים ($k =$ מספר הרמות).
- בהנחה שהעץ מאוזן, יש בכל שאר הרמות $2^k - 1$ קודקודים, כלומר באחד פחות מאשר מספר הצמתים ברמה האחרונה (אפשר לתת לתלמידים לבדוק את העובדה הזו על ידי דוגמאות). לכן יתר הצמתים זניחים ומספיק להתמקד ב- 2^k .
- נסמן ב- n את מספר הצמתים הכללי:

$$2^k = n$$

$$\log_2 2^k = \log_2 n$$

$$k \log_2 2 = \log_2 n$$

$$k = \log_2 n$$

כלומר מספר הרמות נמצא ביחס לוגריתמי למספר הצמתים הכללי:

הוא $O(\log n)$. הבסיס של הלוגריתם אינו חשוב כיוון שהוא ניתן לשינוי על ידי ההכפלה בקבוע.

תשובות לשאלות המחשבה המופיעות בסעיף זה:

? כתבו גרסה איטרטיבית של הפעולה existsInBST(...).

תשובה:

בג'אווה:

```
public static boolean existsInBST(BinTreeNode<Integer> bst, int x)
{
    BinTreeNode<Integer> node = bst;

    while (node != null)
    {
        if (x == node.getInfo())
            return true;
        if (x < node.getInfo())
            node = node.getLeft();
        else
            node = node.getRight();
    }

    return false;
}
```

בסישרפ:

```
public static bool ExistsInBST(BinTreeNode<int> bst, int x)
{
    BinTreeNode<int> node = bst;

    while (node != null)
    {
        if (x == node.GetInfo())
            return true;
        if (x < node.GetInfo())
            node = node.GetLeft();
        else
            node = node.GetRight();
    }

    return false;
}
```

- ? א. נמקו את הטענה שהערך המינימלי בעץ-חיפוש-בינרי נמצא בצומת השמאלי ביותר.
- ב. האם הצומת השמאלי ביותר הוא תמיד עלה? אם לא, האם ייתכן שיש לו תת-עץ שמאלי? האם ייתכן שיש לו תת-עץ ימני? אם התשובות לאחת מהשאלות האלה או לכולן חיוביות, הוכיחו אותן בעזרת דוגמאות מתאימות.
- ג. הגדירו היכן נמצא הערך המקסימלי בעץ-חיפוש-בינרי.

תשובה:

- א. כיוון שבכל שלב בעת בניית עץ או הוספת ערך לעץ-חיפוש-בינרי, נעשית השוואה בין ערך הצומת לערך המוכנס לעץ, הערכים הגדולים מנווטים ימינה, והערכים הקטנים שמאלה. בסוף תהליך ההכנסה הערך הקטן ביותר יימצא במקום השמאלי ביותר בעץ.
- ב. הצומת השמאלי ביותר לא חייב להיות עלה. ייתכן שיש לו תת-עץ ימני. אך לא ייתכן שיש לו תת-עץ שמאלי, כיון שאז הוא אינו השמאלי ביותר.
- ג. בצומת הימני ביותר.

? ממשו את הפעולה:

```
public static void insertIntoBST(BinTreeNode<Integer> bst, int x)
    הפעולה מכניסה מספר שלם לעץ-חיפוש-בינרי המכיל מספרים.
תשובה (בעזרת פעולה רקורסיבית):
    בג'אוה:
```

```
public static void insertIntoBST(BinTreeNode<Integer> bst, int x)
{
    if(x < bst.getInfo())
    {
        if(bst.getLeft() == null)
            bst.setLeft(new BinTreeNode<Integer>(x));
        else
            insertIntoBST(bst.getLeft(), x);
    }
    else
    {
        if(bst.getRight() == null)
            bst.setRight(new BinTreeNode<Integer>(x));
        else
            insertIntoBST(bst.getRight(), x);
    }
}
```

בסישרפ:

```
public static void InsertIntoBST(BinTreeNode<int> bst, int x)
{
    if(x < bst.GetInfo())
    {
        if(bst.GetLeft() == null)
            bst.SetLeft(new BinTreeNode<int>(x));
        else
            InsertIntoBST(bst.GetLeft(), x);
    }
    else
    {
        if(bst.GetRight() == null)
            bst.SetRight(new BinTreeNode<int>(x));
        else
            InsertIntoBST(bst.GetRight(), x);
    }
}
```

}
}

? בדקו כי העצים שבאיורים אכן נוצרו על פי הסדרות המתאימות, על ידי הכנסת הערכים שבהן משמאל לימין. הציגו סדרה המכילה אותם ערכים, אך עץ-החיפוש-הבינרי הנבנה ממנה שונה מכל העצים שבאיור.

תשובה:

אחת הדוגמאות האפשריות היא הסדרה: 6, 5, 26, 15, 30, 9, 29, 8

? עץ א לעיל נוצר על פי סדרת ערכים הממוינת בסדר יורד. מה תהיה צורתו של העץ שייווצר על ידי אותם ערכים הממוינים בסדר עולה?

תשובה:

צורתו של העץ תהיה אלכסון ישר לכיוון השני (מהשורש יפנה האלכסון כלפי ימין).

? נסו להוכיח את השוויון באמצעות אינדוקציה או על ידי חישוב של סכום הנדסי.

תשובה:

הוכחה כסכום הנדסי:

יש לשים לב כי זו היא סדרה הנדסית שבה $k+1$ איברים, וכי $q=2$.

$$S_n = \frac{a_1(q^n - 1)}{q - 1}$$

$$S_n = \frac{2^0(2^{k+1} - 1)}{2 - 1}$$

לפי הנוסחה:

$$S_n = \frac{2^{k+1} - 1}{1}$$

$$S_n = 2^{k+1} - 1$$

הוכחה באינדוקציה:

נבדוק עבור $k=1$:

$$2^0 + 2^1 = 2^2 - 1$$

$$1 + 2 = 3$$

נניח את נכונות הטענה עבור $k=m$, כלומר:

$$2^0 + 2^1 + 2^2 + \dots + 2^m = 2^{m+1} - 1$$

נוכיח את נכונות הטענה עבור $k=m+1$:

נוכיח שמתקיים:

$$2^0 + 2^1 + 2^2 + \dots + 2^m + 2^{m+1} = 2^{m+1+1} - 1$$

לפי הנחת האינדוקציה:

$$\underbrace{2^0 + 2^1 + 2^2 + \dots + 2^m + 2^{m+1}}_{2^{m+1} - 1} = 2^{m+1+1} - 1$$

לכן מספיק להוכיח ש:

$$2^{m+1} - 1 + 2^{m+1} = 2^{m+2} - 1$$

$$2^{m+1} + 2^{m+1} = 2^{m+2}$$

$$2 \cdot 2^{m+1} = 2^{m+2}$$

$$2^{m+2} = 2^{m+2}$$

בדקנו את הטענה עבור $k=1$, הנחנו את ההנחה עבור $k=m$ והוכחנו בעזרת ההנחה שהטענה מתקיימת עבור $k=m+1$. לכן הטענה נכונה עבור כל $k > 1$.

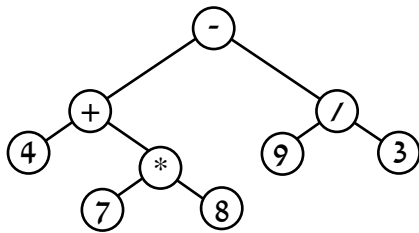
ניתן להעלות שתי שאלות מחשבה בנוסף לאלה המופיעות בפרק:

1. האם נכון שבכל עץ מלא מספר העלים גדול ב-1 ממספר כל שאר הצמתים בעץ? אם לא, הביאו דוגמה נגדית.

2. האם הטענה ההפוכה נכונה, כלומר: האם כל עץ שבו מספר העלים גדול ב-1 ממספר כל שאר הצמתים בעץ הוא עץ מלא? אם לא, הביאו דוגמה נגדית.

תשובות:

- כן, כיוון שבעץ מלא כל העלים מופיעים ברמה האחרונה, וראינו שיש 2^k צמתים ברמה האחרונה. בשאר הרמות יש $2^k - 1$ צמתים.
- לא. ניתן ליצור עץ שאינו מלא המקיים את הדרישה:



בעץ הנתון יש 5 עלים ו-4 צמתים נוספים. אבל הוא אינו עץ מלא.

? כתבו פעולה המקבלת עץ-חיפוש-בינרי ומדפיסה את איבריו בצורה ממוינת בסדר יורד.

תשובה:

בג'אוה:

```
public static void printBSTInDownOrderSort (BinTreeNode<Integer> bst)
{
    if (bst != null)
    {
        printBSTInDownOrderSort (bst.getRight ());
        System.out.print (bst.getInfo () + " ");
        printBSTInDownOrderSort (bst.getLeft ());
    }
}
```

בסישרפ:

```
public static void PrintBSTInDownOrderSort (BinTreeNode<int> bst)
{
    if (bst != null)
    {
        PrintBSTInDownOrderSort (bst.GetRight ());
        Console.Write (bst.GetInfo () + " ");
        PrintBSTInDownOrderSort (bst.GetLeft ());
    }
}
```

ט. מבני נתונים לעומת טיפוסי נתונים מופשטים

סעיף זה מסכם את כל מה שנלמד לאורך היחידה בנושאי טיפוסי אוספים, טיפוסי נתונים מופשטים ומבני נתונים. מי שעקבו אחר התפתחות הדברים עד כאן והבינו, כדאי שיסכמו את הנושא בפסקה זו המסכמת למעשה את היחידה כולה. בכיתות שבהן מתגלים קשיים שונים לא כדאי להתעכב על פסקת סיכום זו.

4. דגשים, קשיים והערות חשובות

- השרשרת והעץ הבינרי שניהם מבנים המורכבים מחוליות. הצבעה על הדמיון ביניהם תקל על התלמידים את הבנת מבנה העץ.
- לשרשרת הצגנו מחלקה עוטפת – הרשימה, אך לא עשינו כך עבור עץ בינרי. נמנענו מכך מכמה סיבות: מפאת חוסר זמן, כיוון שאין בזה רעיונות חדשים, וכיוון שעץ בינרי כללי אינו שימושי במיוחד. לעומת עץ בינרי כללי, עץ-חיפוש-בינרי הוא מבנה מעניין ושימושי, והעדפנו לעסוק בו. למבנה זה אנו מציגים מחלקה עוטפת בפרק הבא. כאשר מעבירים לפעולה עץ חוליות בינרי, טיפוס הפרמטר הוא BinTreeNode. כל צומת או שורש מקומיים (תת-עץ) הם מטיפוס זה של BinTreeNode. העץ שהגדרנו הוא למעשה אוסף של חוליות מטיפוס BinTreeNode, הקשורות בינן לבין עצמן (ללא שום מעטפת) במבנה של עץ חוליות בינרי. כאמור לעיל, הטיפוס הוא חוליה, ואין לו כשלעצמו שום משמעות של עץ.

- קיימים שני סוגים עיקריים של אלגוריתמים רקורסיביים על עצים, ולכל אחד מהם יעילות אופיינית:
 - הסוג האחד של אלגוריתמים רקורסיביים כולל סריקות שונות, ומתבצעת בו הפעלה רקורסיבית של האלגוריתם על כל אחד משני התת-עצים. אלגוריתמים מסוג זה עוברים פעם אחת על כל אחד מצומתי העץ, ולכן יעילותם היא מסדר גודל $O(n)$.
 - האלגוריתמים מהסוג האחר מופעלים רקורסיבית על אחד מהתת-עצים. דוגמאות לאלגוריתמים מסוג זה הם האלגוריתמים לחיפוש ולהכנסה של איבר לעץ-חיפוש-בינרי. אלגוריתמים אלה עוברים לאורכו של שביל מסוים בעץ. מספר הצמתים בשביל כזה הוא לכל היותר כגובה העץ, ולכן יעילותם של האלגוריתמים מסוג זה היא מסדר גודל של גובה העץ – (גובה העץ) O . גובה העץ עצמו נע כזכור בין $O(\log n)$ לבין $O(n)$, ובהתייחסות ליעילות האלגוריתמים יש להבדיל בין עצים מאוזנים לעצים שאינם כאלה.
- מבנה המחסנית מאפשר לדמות מהלך של רקורסיה. כדאי להדגים לתלמידים את הרעיון הזה בעזרת שאלות מהפרק (שאלה 9) ומתוך המאגר, במיוחד בפרק זה שבו השימוש ברקורסיות רב ביותר.

5. תרגילים

פתרונות לכל התרגילים מופיעים באתר המרכז להוראת המדעים, האוניברסיטה העברית בירושלים ונגישים רק לציבור המורים. בפרק זה מרוכזים כל הפתרונות בקובץ אחד.
http://sites.huji.ac.il/science/unit4_2007/

שאלה 1

תרגול המושגים הבסיסיים בנושא עצים. אפשר לתת כשיעורי בית.

שאלה 2

תרגול סריקות העומק ורוחב. אפשר לתת כשיעורי בית.

שאלה 3

בניית עץ על פי סריקות נתונות.
שאלה זו אינה פשוטה והיא מתרגלת הבנה מעמיקה של סריקות לעומק. מומלץ לתת רק בכיתות חזקות.
אפשר לתת קודם כשיעורי בית כדי לאפשר התמודדות לבד עם הניסיונות לבנות עצים, אך אחר כך לעבור לאט בכיתה.

שאלות 4–8

מעקב אחרי פעולות רקורסיביות.
מומלץ לעבור על חלק מהתרגילים בכיתה וחלק לתת כשיעורי בית.

שאלה 9

מעקב אחרי פעולה רקורסיבית: סריקה בסדר תוכי באמצעות מחסנית.
מומלץ לפתור בכיתה.
מומלץ להדגיש בפני תלמידים שהרקורסיה הוחלפה על ידי מחסנית.

שאלות 10–20 עוסקות כולן במימוש פעולות על עצים:

שאלה 10

כתיבת פעולה של סריקה לרוחב.
פעולה דומה מאוד מופיעה בפרק כשאלת מחשבה. אם פותרים את שאלת המחשבה יחד בכיתה ניתן לתת את השאלה הזו כשיעורי בית (יש כאן שינוי הטיפוס של ערכי העץ וערך ההחזרה בלבד).
אחרת מומלץ לפתור בכיתה.

שאלות 11–13

מימוש פעולות על עצים הקשורות באיתור ערך מסוים בעץ.
שלוש השאלות דומות מאוד, לכן מומלץ לפתור שאלה אחת בכיתה ולתת את שתי השאלות האחרות כשיעורי בית או כבוחן.

שאלה 14

מומלץ לפתור בכיתה.

שאלה 15

מומלץ לפתור בכיתה.
אפשר לתת רמז לתלמידים שבקריאות הרקורסיביות יפחיתו את הרמה ב-1.

שאלה 16

מומלץ לפתור בכיתה.

שאלה 17

כתיבת פעולה על עץ הבודקת האם העץ מלא.
קיימות שתי דרכים שונות לפתרון התרגיל:
דרך 1: באמצעות רקורסיה.

דרך 2 : באמצעות הבדיקה האם מספר הצמתים הכללי בעץ הוא $2^{h+1} - 1$ (h הוא גובה העץ). מומלץ להציג בכיתה את שתי הדרכים, לדון בהן ולתת לתלמידים לסיים את הפתרון בבית.

שאלה 18

איתור ההורה של צומת נתון.
 קיימות שתי דרכים שונות לפתרון התרגיל :
 דרך 1 : באמצעות רקורסיה.
 דרך 2 : באמצעות סריקה לפי רמות, כאשר בכל שלב בודקים האם הצומת הנוכחי אינו אב לצומת נתון.
 מומלץ להציג בכיתה את שתי הדרכים ולדון בהן, ולתת לתלמידים לסיים את הפתרון בבית.

שאלה 19

בניית עץ רנדומלי.
 ניתן לכוון את התלמידים לכך שאף המבנה של העץ נבחר על ידי מספר אקראי.
 קיימים ארבעה מצבים במבנה העץ : צומת עלה, צומת עם ילד שמאלי בלבד, צומת עם ילד ימני בלבד וצומת עם שני ילדים.
 בפתרון שאנו הצגנו, ההתפלגות של ארבעת המצבים אינה שווה (למשל הסיכוי שיצא 2 הוא 1 ל-11, ולעומת זאת הסיכוי שיצא לא 0, לא 1 ולא 2 הוא 6 ל-11, כלומר יהיו הרבה יותר צמתים עם שני ילדים מאשר צומתי עלים. ניתן לשאול את התלמידים האם הם יכולים לשפר את ההתפלגות. מומלץ לדון בכיתה בדרך הפתרון, תוך שהמורה מכוון את התלמידים במקרה הצורך. לאחר הדיון רצוי לתת לתלמידים לסיים את פתרון התרגיל בבית.

שאלה 20

בניית עץ על פי שתי סריקות.
 זוהי שאלה קשה. לפני שפותרים שאלה זו מומלץ לפתור את שאלה 3, כדי להבין באופן תיאורטי כיצד מבצעים את הפעולה.
 מומלץ לפתור בכיתה.

שאלה 21

הבנת ההגדרה של הפרש ילדים ומימוש פעולה על עץ.
 מומלץ לפתור בכיתה.

שאלה 22

הבנת ההגדרה של עצים דומים ומימוש הפעולה על עץ.
 מומלץ לפתור בכיתה.

שאלה 23

השאלה עוסקת בעץ ביטוי, ובכלל זה בבנייה של עץ ביטוי ממחרוזת נתונה. להזכירכם, הפעולות והמספרים מיוצגים בעץ כתווים. חישוב הביטוי על פי עץ נתון אינו נכלל בשאלה כיוון שהוא מופיע כשאלת מחשבה בסעיף ז. מומלץ לפתור בכיתה.

שאלה 24

השאלה עוסקת בעץ-חיפוש-בינרי. מומלץ לפתור בכיתה.

שאלה 25

השאלה עוסקת בעץ-חיפוש-בינרי. אפשר לתת כשיעורי בית, אך מומלץ מאוד לדון לפני כן בשאלה בכיתה. בסעיף א יש להדגיש שהצומת 6 הוא עלה.

שאלות 26–27

שאלות אלה חשובות מאוד, ועוסקות בשימוש בעץ בינרי לייצוג של מחלקה אחרת. מומלץ לפתור בכיתה.

יש לשים לב לאופן הכתיבה של הפעולות הרקורסיביות על העץ, שהוא תכונה של שתי המחלקות. הייצוג של המחלקה StudentList נראה כך :

```
public class StudentList
{
    BinTreeNode<Student> list;
}
```

כיוון שהעץ הוא תכונה, כל הפעולות הפנימיות אינן מקבלות אותו כפרמטר, אלא הן פועלות עליו כתכונה. הבעיה מתעוררת כאשר משתמשים בפעולות רקורסיביות. כל האלגוריתמים הרקורסיביים חייבים לקבל את העץ כפרמטר. למשל הכותרת של פעולת ההכנסה לעץ-חיפוש-בינרי היא :

```
public static void insertIntoBST(BinTreeNode<Integer> bst, int x)
```

כדי להתמודד עם הנקודה הזו יש להגדיר פעולות עזר רקורסיביות. נסביר ביתר פירוט: כדי לבצע את פעולות החיפוש וההכנסה נממש פעולות פנימיות פרטיות רקורסיביות שייקראו existHelp(...) / ExistHelp(...), insertHelp(...) / InsertHelp(...). פעולות אלה יקבלו כפרמטרים הן את ההפניה לעץ (תחילה בזימון הראשון: this.tree, ובהמשך יהיו אלה תת-עצים) והן את הערך, ויבצעו את המשימה הנדרשת, בהנחה שערך ההפניה אינו null ושהיא מייצגת עץ-חיפוש-בינרי. הבדיקה שהאוסף אינו ריק תתבצע על ידי הקוד של פעולות הממשק, שיזמנו את הפעולה הפרטית רק אם ההנחה מתקיימת. נראה לדוגמה את תבנית הקוד עבור הפעולה insert(...) / Insert(...):

```
public void insert(Integer x)
{
    if (tree == null)

        else insertHelp(this.tree, x);
}
```

פעולת ההוצאה:

עבור הפעולה remove(...) המוציאה ערך מהעץ, נסתפק ברעיון פשוט (שביצועו יקר יחסית): בהינתן ערך x, נחפש האם הוא קיים בעץ ש-tree מפנה אליו. אם העץ ריק, או שאינו ריק אך אינו מכיל את x, אין צורך לעשות דבר. אחרת, אלגוריתם החיפוש מגלה את הצומת המכיל אותו. ננתק את התת-עץ שצומת זה הוא שורשו, מן העץ, ואחר כך נכניס כל אחד מהערכים שבו (פרט ל-x שבשורש) בחזרה לעץ.