עץ בינרי

הערה חשובה:

מדריך למורה זה הוא טיוטת ביניים לשנה זו – ולכן הוא מתייחס לעצים של שלמים (כפי שהם מופיעים בפרק לתלמיד) ולא לעצים גנריים כפי שילמדו בשנים הבאות.

הצעה למהלך הוראה

1. **פתיחה וסעיף א**: הצגת מושג העץ, כמבטא מבנה היררכי מוכר מחיי היום יום. הכרת המושגים הראשונים. 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".
2. **סעיף ב**: התמקדות בעץ הבינרי. הכרת המושגים הרלוונטיים. הבהרת התחושה שמדובר במבנה רקורסיבי מעצם הגדרתו. הדגשת השימוש בהפניות כבסיס להגדרת המבנה (דומה לרשימה אך מורכב ממנה).
3. תרגיל 1 בסוף הפרק מתרגל את המושגים הראשונים שנלמדו.
4. **סעיף ב.1.:** הבחירה בייצוג. בעץ העדפנו לא להשתמש במחלקה עוטפת (כמו שעשינו ברשימה) כדי להתמקד באלגוריתמים. השימוש בעטיפה היה מקשה על ההצגה הרקורסיבית המתבקשת כאשר דנים בעץ בינרי. ניתן היה גם בפרק רשימה לבחור בייצוג בעזרת חוליות בלבד ולהרחיב את היישומים והשימושים הרקורסיביים ברשימה, אך אנו העדפנו לייצג כל אחד מהמבנים בדרך שונה ולמקד דיונים שונים במבנים. יש לשים לב ולהדגיש מידי פעם במהלך ההוראה ששורש העץ כולו כמו גם כל שורש מקומי, או צומת, או תת עץ הם כולם מסוג החוליה הבסיסית BinTree.
5. רעיון הגנריות. ניתן בהחלט לדבר על עץ בינרי גנרי כפי שעשינו במבנים ובטיפוסי הנתונים הקודמים שנלמדו ביחידה. כפי שציינו שם, לעיתים לא נוכל לבצע פעולות חיצוניות על מבנה גנרי מכיוון שהפעולות המתבצעות על הערכים השמורים במבנה, לא תמיד יהיו מוגדרות עבור כל ערך (או שהגדרתן תשתנה עבור ערכים מטיפוסים שונים). כדי להתמקד בעיקרי הנושאים החשובים לנו ביחידה זו, בחרנו לפשט את התהליך ולהאחיד את דרך כתיבת הפעולות. אנו נצמצם את העיסוק בפרק לעצים קונקרטיים שערך המידע השמור בצומתיהם הוא של מספרים שלמים בלבד. עם המעבר לעץ חיפוש בינרי נרחיב את העיסוק לעצים בהם פריט המידע הוא זוג של מפתח (מספר שלם) ומידע גנרי כלשהו המקושר אליו. (בגרסה הסופית של יחידת הלימוד יתייחס פרק עץ בינרי לעצים גנריים.)
6. העץ הריק. צורת הייצוג בה בחרנו אינה מאפשרת לדבר על "עץ ריק". העץ המינימלי מכיל צומת אחת לפחות. (אילו היינו משתמשים במעטפת, היינו יכולים לייצג עץ ריק, אך המחיר היה כפי שצויין לעיל, סיבוך של הפעולות הרקורסיביות.) אנו מדברים בהמשך על הפנייה מטיפוס עץ שערכה **null** (זה יהיה תנאי בדיקה ועצירה לפעולות שונות כאשר ילד ימני או

שמאלי יהיה שווה ל-**null**, כלומר לא נוכל להתקדם על פני העץ בכיוון זה). הפניה כזו אינה מייצגת "עץ ריק", המושג אינו קיים בייצוג בו בחרנו. בסעיף ז אנו עוסקים בפעולות חיצוניות. בשלב זה יכול בהחלט להיות שישלח לפעולה חיצונית המקבלת פרמטר מטיפוס עץ, פרמטר אקטואלי שערכו **null**. הטיפול באפשרות זו יהיה על ידי הכנסה לחוזה עם המשתמש (=תיעוד ממשק המחלקה) את ההנחה שלא ניתן לשלוח פרמטר שערכו **null** במקום בו הפרמטר מייצג עץ בינרי.

משמעויות שונות יש לכך שאיננו עוטפים את החוליות של העץ בשום מעטפת חיצונית, אלא מטפלים בחוליות והן אלו שאחראיות לקשרים בינן לבין עצמן. חלק מהדברים יכול להיות שיעלו בכיתות: הזהות בין שם המחלקה/הטיפוס לשם כל חוליה באוסף. הרעיון שאם הייתה מעטפת אז פעולות כמו getLeft, getRight היו צריכות להחזיר טיפוס אחר מטיפוס המחלקה (הפנייה לחוליה "פנימית" ממנה בנוי העץ, או דווקא הפנייה למחלקה עץ העוטפת), ועוד.

7. **סעיף ב.2.:** ממשק ו-UML. קודם להצגת הממשק של המחלקה BinTree יש להציג בשלבים ותוך דיון את הפעולות שאנו זקוקים להם בעבור מבנה נתונים זה: העמסת הפעולות הבונות, מדוע מסובך להגדיר הכנסה של איבר לעץ או מחיקת צומת בודד מהעץ. לסיום יש להציג את הממשק המלא והמוסכם. גם בפרק זה יש להקפיד על איורי UML המגדירים את העץ הבינרי, כקביעת דרך אחידה לדיון בעצים.
8. האם עץ בינרי הוא טיפוס נתונים מופשט? תליית עץ חדש על עץ קיים (בעזרת פעולות ה-set) עלולה לפגוע במבנה התקין של העץ הבינרי. 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 setLeft, setRight מהממשק, אך הנזק הנלווה היה חוסר יכולת לבנות עצים, כך שלא הייתה למעשה אפשרות להשתמש במחלקה. אנו מעדיפים לכלול את הפעולות האלו תוך הדגשת האחריות המוטלת על המשתמש בפעולות אלו, כלפי מבנהו התקין של העץ, ועם הסבר בהמשך הפרק, שעץ בינרי הוא מבנה נתונים. אגב, העץ יישאר מבנה נתונים שמבנהו חשוף, גם כאשר נגדיר עליו פעולות נוספות גלובליות שיטפלו בכולו ולא רק פעולות מקומיות כפי שמפרט הסעיף הבא.
9. 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 בעץ שלם. לכן גם סיבוכיות זמן הריצה של כל אחת מהפעולות היא *O(1)* מכיוון שהיא מתבצעת במקום עצמו (נושא זה מופיע בפרק בסעיף ד.1). (הדיון על השימוש במחלקה של חוליה בינרית לייצוג עץ, המופיע בסוף הפרק, רלבנטי כאן.)
10. תקינות העץ. העובדה שהממשק מכיל רק פעולות על צומת (חוליה בינרית) אחת, קשורה אף היא לעובדה שהעץ הוא מבנה נתונים בלבד. אין בממשק כלל פעולות שאחראיות לעץ כאוסף. בנית עץ וטיפול במבנהו הן לגמרי באחריות המשתמש במחלקה. למעשה, אנו מגדירים כאן מחלקה של חוליות, כאשר כל חוליה מכילה שדה נתונים ושני מצביעים לחוליות אחרות מאותו טיפוס (או ל-**null**). ניתן להשתמש בחוליות כאלו לבניית עצים בינריים, כפי שאנו

עושים בפרק זה, או לבניית רשימות משורשרות דו-כיווניות, 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 יכולת פתרון בעיות על עצים שכאלו.

11. הנחות. כאמור בפרקים הקודמים, ההנחות המוגדרות בתיעוד הממשק, הן החוזה המחייב בין מתכנת המחלקה והמשתמשים בה. חוזה זה מבטיח בנייה של עץ תקין ושימוש נכון בו. נכון שיש בידי המשתמש יכולת לפגוע במבנהו התקין של העץ אך ההנחות שלנו מהוות הסכמה לשימוש נכון ואמין בעץ.

12. **סעיף ג:** שימושים בפעולות הממשק. תוך הדגמה באיורים וציורים של ההתקדמות על פני העץ והפנייה לערכים שבצמתים. שאלות המעקב 6-8, שבסוף הפרק מאפשרות לעצור את שטף הלימוד ולתרגל את השימוש בעצים, בפעולות ובהתקדמות על פני עצים.

13. **סעיף ד:** מימוש העץ הבינרי. התלמידים צריכים להחזיק בידיהם את המחלקה BinTree לצורך מימושים ופתרונות מתקדמים בהמשך. קוד המחלקה אמנם מופיע בספר הלימוד, והמחלקה המקומפלת נמצאת ב-unit4.jar שבידי התלמידים אך כדאי וראוי שהתלמידים יממשו את המחלקה כולה בעצמם. כך יוכלו בהמשך להיכנס לקוד המחלקה ולשנותו על פי שאלות ותרגילים שיתעסקו בייצוג ובשיפורי יעילות של העץ הבינרי.

14. **סעיף ה**: פעולות נוספות על עץ בינרי. הפעולות יתחלקו לסוגים שונים: פעולות פנימיות פשוטות ופעולות פנימיות רקורסיביות, פעולות חיצוניות פשוטות ופעולות חיצוניות רקורסיביות. לא לכל פעולה נציג את גרסתה הפנימית הרקורסיבית כדי להקטין את הבלבול והתסבוכת.

15. **סעיף ו:** מעברים על עץ בינרי. הבנה ראשונית של המעברים תעשה לפי האיור, טרם הצגת הנוסחים הפורמליים. לאחר מכן יש להציג את התבניות לסריקות עומק של עצים בינריים ואת אופן הכתיבה האלגוריתמי בו אנו משתמשים לצורך תיאור סריקות אלו. תבניות אלו ישמשו אותנו לפתרון בעיות רבות העוסקות בעצים, ראו מבחר ראשוני בסעיף ז בפרק. יש להבין את סיבוכיות זמן הריצה של סריקות אלו. ניתן לקטוע את רצף הלימוד ולתרגל את הנלמד בעזרת התרגילים המופיעים בסוף הפרק (תרגיל 2, חלק מתרגילי המעקב, שחזור עצים על פי נתוני הסריקות ועוד). יש להדגיש שהמעברים אינם "יותר" פעולה פנימית מחיצונית וניתן להגדיר את הסריקות כך או כך באופן שווה לחלוטין, לכן גם צורפה הדגמה בפרק של סריקה בסדר תוכי כפעולה פנימית וחיצונית במקביל.

16. ביקור בשורש. בחישובי היעילות אנו מתייחסים לביקור בשורש אך יש לשים לב ולציין לפני התלמידים שפרט למעבר בשורש בעת ביצוע הביקור בו, אנו חולפים בו עוד פעם או פעמיים בדרך להורה שלו או לילדיו.

17. הכתיבה האלגוריתמית. באופן כללי נטינו שלא לייצר שפה אלגוריתמית חדשה, בעלת מטבעות לשון מחייבים, אלא להשתמש בשפה פשוטה ושוטפת לתיאור עיקרי התהליכים. במקרה של עצים אנו נדרשים להגדרה פורמלית מסוימת כדי להבהיר איך וכיצד מתרחשים התהליכים על עצים ובעיקר כיצד מתנהלות הסריקות והמעברים הרקורסיביים על עצים. לכן אנו משלבים הגדרת נוסח ברור ואחיד של הכתיבה האלגוריתמית, ומייצרים שתי מטבעות לשון אחידות וברורות:

i. "**הפעל ב-**" – כאשר זו פעולה פנימית המתבצעת דרך צומת מסוים ואנו מציינים מיהו הצומת.

ii. "**הפעל את**" כאשר מדובר בפעולה חיצונית המקבלת עץ בינרי כפרמטר, בתוך סוגריים.

18. **סעיף ז:** פעולות חיצוניות הסורקות עצים בינריים. יש להבהיר את רעיון החוזה בין המתכנת למשתמש לפיו בפעולות חיצוניות המקבלות פרמטר מטיפוס עץ, לא ישלח פרמטר אקטואלי שערכו **null**. יש להתעכב ולהזכיר את הנוסח האלגוריתמי לפעולות חיצוניות על עצים. כמובן שבגלל שאנו מתעסקים בעצים קונקרטיים בלבד (עצים ובהם ערכים שלמים), אין מניעה להתייחס לערכים השמורים בצמתי העצים (השוואה, חיבור וכד').
אחד התנאים המוזכרים בפרק בסעיף זה הוא הרעיון שניתן (ואפילו כדאי וחסכוני) להפסיק את הסריקה כאשר תנאי מסוים התקיים למשל מציאת נתון שאותו מחפשים. הקוד המובא בפרק מדגים עצירה שכזו, שימו לב לשורות המובלטות להלן. ברגע שהערך נמצא חוזרים במסלול הרקורסיה אחורנית עם הערך **true**:

```
/**      null אינו tree ערך הפרמטר :הנחה */
public static boolean existsIn (BinTree tree, int x)

{
    boolean resL = false;
    boolean resR = false;

    if (tree.getInfo() == x)
          return true;
    if (tree.getLeft() != null)
    {
         resL = existsIn (tree.getLeft(),x);
        if (resL)
             return true;
     }
     if (tree.getRight() != null)
        resR = existsIn (tree.getRight(),x);
     return (resR);
  }
```

מובן שיכול היה להופיע קוד קצת פחות יעיל שהיה ממשיך את הסריקה עד סופה ולא עוצר ברגע בו נמצא הערך הדרוש:

```
public static boolean existsIn (BinTree tree, int x)
```

```
{
    boolean resL = false;
    boolean resR = false;
     if (tree.getInfo() == x)
          return true;
     if (tree.getLeft() != null)
          resL = existsIn (tree.getLeft(),x);
     if (tree.getRight() != null)
         resR = existsIn (tree.getRight(),x);
     return (resL || resR);
  }
```

19. **סעיף ז2.**: שאלת המחשבה בסעיף זה היא למעשה שאלה מספר 2 המופיעה בשאלות שבסוף הפרק. ניתן לתת אותה בפועל כתרגול.

20. **סעיף ח:** סריקה לפי רמות. בפרק מופיע אלגוריתם ומימוש של הסריקה כפעולה פנימית הסורקת את העץ הנוכחי באופן איטרטיבי. שימו לב להבדל בין מבנה פעולה פנימית זו, למבנה הפעולות הפנימיות הרקורסיביות שהוצגו עד כה. בכל אלו, טיפול או ביקור בצומת נעשה במסגרת של הפעלה (רקורסיבית) של הפעולה בצומת עצמו. בסריקה לפי רמות הפעולה הנוכחית פועלת בשורש, והביקור בצמתים האחרים נעשה בשורש, ולא באותם צמתים. הסיבה להבדל היא הצורך בתור, שנוצר פעם אחת, וכל צומת עובר בו. בגלל המבנה הזה, קל לנסח פעולה חיצונית איטרטיבית, המקבלת עץ כפרמטר, המבצעת את הסריקה. לפניכם נוסח פעולה זו. כדאי לנסות ולהגיע לנוסח זה יחד עם התלמידים. יש לקיים דיון שיוביל להבנה שגם במקרה זה יעילות הריצה היא פונקציה ליניארית של מספר צמתי העץ. כדי להבין וכדי לממש סריקה זו יש להיזכר בטיפוס הנתונים תור ובממשק שלו.

סריקה לפי רמות כפעולה חיצונית איטרטיבית:

```
public static String levelOrderTraversal (BinTree tree)
        {
                BinTree t;
                String str = new String();
                Queue<BinTree> q = new Queue<BinTree>();

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

                return (str);
        }
```

אפשר לממש סריקה פנימית לפי רמות גם בסגנון רקורסיבי. לשם כך יש להגדיר פעולה פנימית רקורסיבית פרטית שמקבלת תור כפרמטר, והיא זו המבצעת את הסריקה, ופעולה פנימית פומבית שיוצרת תור, ואז קוראת לפעולה הפרטית, עם התור כפרמטר. פיתוח זוג פעולות אלו מתאים לכיתות מתקדמות:

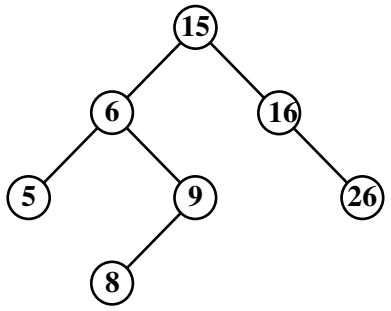
```
public  String levelOrderTraversal()
{
        String str = new String();
        Queue<BinTree> q = new Queue<BinTree>();
        return levelOrderTraversalHelp(q, str);
}

private  String  levelOrderTraversalHelp (Queue<BinTree> q, String str)
{
        str = str + this.getInfo() + " , ";
        if (this.getLeft() != null)
                q.insert (this.getLeft());
        if (this.getRight() != null)
                q.insert (this.getRight());
        if  (!q.isEmpty())
        {
                return q.remove().levelOrderTraversalHelp (q, str);
        }
        else
        {
                return str;
        }

}
```

21. **סעיף ט:** עץ ביטוי. זוהי דוגמה יישומית לשימוש במבנה הנתונים עץ בינרי. כך אכן פועלים קומפיילרים 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. התרגילים 16-18 בסוף הפרק מכילים התייחסות מעשית לנושא זה. בסעיף הפתרונות לתרגילים, מופיע אלגוריתם מפורט לבניית עץ ביטוי, אותו אתם יכולים לחלק לתלמידיכם לקראת פתרון שאלה 18.

<u>שימו לב</u>: מחלקת העץ השנה מוגדרת על מספרים שלמים ואם כך אינה יכולה לשמש אותנו לפתרון התרגיל. עץ ביטוי מוגדר כמכיל תווים ומספרים. הצמצום בנוסח השאלה לפיו כל מה שקיים בעץ הוא חד ספרתי ואינו מכיל "–", מאפשר לנו לחזור אל המחלקה ולשנות אותה לעץ בינרי של תווים, וכך לפתור את התרגילים.

22. **סעיף י:** לאחר כל הצגת העץ הבינרי, יש לדון ולהבין שהוא אינו מהווה טיפוס נתונים מופשט אך הוא מבנה נתונים שיכול לשמש אותנו לייצוג טיפוסי נתונים שונים כפי שהסברנו קודם.

23. **סעיף יא:** עץ חיפוש בינרי. הצגת ההגדרה של עץ חיפוש בינרי. למעשה יכול עץ חיפוש בינרי להכיל ערכים חוזרים. מקומם בעץ יקבע מימין לערך ממנו הם גדולים או שווים לו בדיוק. בפרק זה אנו מעוניינים להגיע לדמיון ולקשר שבין עץ חיפוש בינרי לטיפוס הנתונים המופשט מפה, ולכן אנו מצמצמים את הגדרת העץ החיפוש הבינרי לעץ שאינו מכיל חזרות. סיבה נוספת היא שאלגוריתמים על עץ חיפוש בינרי עם חזרות קצת יותר קשים למימוש. בשאלות המאגר תמצאו שאלה המתייחסת לייצוג עץ חיפוש בינרי כך שיכיל את מספר החזרות של מספר כערך הצמוד למפתח שהוא המספר עצמו.
24. **סעיף יא.1.:** כדאי לתרגל חיפוש על עץ נתון, למצבים של הצלחה וכישלון, כדי לראות את השביל לאורכו מתקדמים.
25. **סעיף יא.4.:** חישובים של יעילות. מדד עיקרי הקובע יעילות פעולות בעץ חיפוש בינרי הוא מספר הרמות בעץ מסוים. סעיף יא.3. עוסק בעצים מאוזנים ובלתי מאוזנים ומחשב את טווח הרמות האפשרי בעצים שונים. שימו לב שמספר הרמות בעץ הבסיסי ביותר הוא 0 שכן לעולם עץ קיים יש בו לפחות צומת אחד! תרגילים 12-11 המתרגלים פעולות פנימיות וחיצוניות נעזרים במידע שנלמד בסעיף זה ויכולים להוות נקודת עצירה בהוראת הנושא, לצורך תרגולו.
26. **סעיף יא.5.: פעולת ההוצאה**. מכיוון שהאלגוריתם של פעולת ההוצאה אינו טריויאלי ומימושו מורכב, נמנענו מלפרטו בפרק. נסביר את מהלך פעולת ההוצאה מעץ חיפוש בינרי בעזרת האיור הבא:



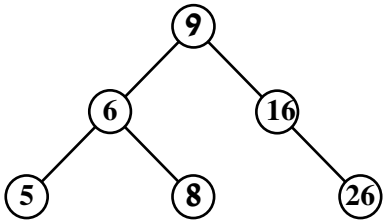
נניח כי אנו רוצים להוציא את הערך 8 מהעץ. ראשית נבצע חיפוש שימצא את הצומת בו נמצא הערך. בדיקה מגלה כי זה צומת עלה. אם כך ניתן למחקו מהעץ, על ידי החלפת ההפניה אליו מן ההורה (הצומת המכיל 9), ב-**null**.

נניח כי אנו רוצים להוציא את הערך 9. לאחר שהגענו לצומת המכיל אותו, אנו בודקים ומגלים כי יש לו רק תת עץ שמאלי. כיון 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 כלהלן: נחליף את ההפניה בהורה להפניה לתת עץ השמאלי (כמו במקרה הקודם). אחר כך נכניס את כל הערכים שבתת עץ הימני לעץ, אחד, אחד. ברור שזה פתרון נכון, אך די 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 תוכית. אם כך, נבצע סריקה תוכית מהצומת שהערך בו מיועד להחלפה. כשנגיע לערך העוקב לזה שבצומת בו 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).

שאלה 21 המבקשת מהתלמידים לכתוב את המחלקה BinSearchTree יכולה להיעשות ללא פעולת remove כדי להקל ולחסוך בזמן.

27. **סעיף יא.5. הפעולה getAll.** לצורך ביצוע פעולות שונות על הערכים השמורים בעץ החיפוש עלינו לדעת מיהם ערכים אלו. הפעולה toString המוגדרת על עץ חיפוש בינרי מחזירה את רשימת הערכים השמורים בעץ החיפוש, משורשרים למחרוזת. להבדיל ממפה, בה הפעולה getAll מחזירה צירופים של מפתחות עם ערכיהם, בעץ חיפוש בינרי מוחזרים רק ה"מפתחות", ולכן יכולנו להיעזר לכאורה בפעולה toString המוגדרת על כל עצם, כדי לקבל את רשימת הערכים השמורים בעץ. אך המחרוזת המוחזרת על ידי הפעולה אינה פשוטה לטיפול, ואינה מיועדת למטרה זו. לכן, כדאי להגדיר פעולה getAll המחזירה מערך או רשימה של הערכים.

28. ייצוג המחלקה. במחלקה עץ בינרי, מאפשרות הפעולות על העץ לקלקל את מבנהו ותכונותיו, למשל כאשר מחליפים את התת עץ השמאלי בעץ חדש, שהוא התת עץ הימני, ומקבלים מבנה עם תת עצים בעלי אותם צמתים (הפעולות מאפשרות למעשה למשתמש בהן לגעת בתכונות העץ). במחלקה המממשת עץ חיפוש בינרי לעומת זאת, התכונות הן פרטיות והפעולות הניתנות למשתמש אינן מאפשרות לפעול עליהן ישירות. כל פעולות הממשק שומרות על המבנה המדויק של עץ החיפוש הבינרי.

29. **סעיף יא.6.:** סעיף זה מתייחס לטיפוס הנתונים מפה. מורים שלא למדו את הנושא ולא תרגלו אותו ידלגו על קטעים מתוך סעיף זה וסעיף יא.7. בהתאמה, או ישלבו לימוד של הנושא Map.
30. **סעיף יא.7.**: אינו מתאים למי שלא לימדו את הסעיפים המקבילים בכל פרקי היחידה. מי שלא התייחסו למשמעותו של טיפוס נתונים מופשט מול מבנה נתונים יכולים לדלג על סעיף זה, אך למי שיצרו את ההבחנות, משמש סעיף זה סיכום נאה של כל היחידה.

הערות חשובות:

- בפרק מוצגים אלגוריתמים רקורסיביים רבים ומחושבת סיבוכיות זמן הריצה שלהם. למעשה, 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 $O(n)$.
  - 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 *(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.

תשובות לשאלות המחשבה בגוף הפרק (חלקי)

<u>סעיף יא.2.:</u>

הערך הקטן ביותר בעץ חיפוש בינרי ממוקם בצומת השמאלי ביותר בעץ. לפניכם אלגוריתם למציאת הערך המינימלי בעץ חיפוש בינרי:

מצא-מינימום-בעץ-חיפוש-בינרי()

אם לא קיים תת-עץ שמאלי לשורש, החזר את ערך השורש

אחרת,

*החזר את תוצאת **מצא-מינימום-בעץ-חיפוש-בינרי ()** המופעל בתת-עץ השמאלי*

הערה: שימו לב שנוסח האלגוריתם מצביע שהוא מגדיר פעולה פנימית של עץ חיפוש בינרי.

קשיים צפויים:

- הרשימה והעץ הבינרי שניהם ממומשים בעזרת חוליות. השוני בדרך הייצוג שלהם, הרשימה כמבנה המורכב ממיכל והחוליות שמורות בתוכו והעץ הבינרי כמורכב רק מהחוליות עצמן, יהוו ברבות מהכיתות מכשול שיש להתגבר עליו בעזרת איורים והסברים מקדימים.
- העץ כולו הוא מטיפוס BinTree, גם כל צומת או שורש מקומי (תת עץ) הוא מטיפוס זה של BinTree. העץ שהגדרנו הוא למעשה אוסף של חוליות מטיפוס BinTree, הקשורות בינן לבין עצמן (ללא שום מעטפת) במבנה עץ בינרי. כאמור לעיל, הטיפוס הוא חוליה ואין לו שום משמעות של עץ מבחינת מבנה, אלא כאשר אנחנו בוחרים באופן ארגון מסוים של החוליות הללו.
- קיימת בעייתיות בהגדרת פעולות רקורסיביות פנימיות. לצורך כך יש לעיתים להגדיר פעולת עזר פנימית, פרטית, רקורסיבית. ניתן להראות נושא זה בכיתות שמפגינות עניין וקלות תפיסה אך יש להבין שהנושא אינו פשוט.

תרגילים

התרגילים המצורפים לפרק אמורים לעזור בתרגול הנושאים הנלמדים בו אך לאו דווקא עם תום לימוד הפרק כולו. כדאי לשבור את רצף ההוראה ולתרגל כל נושא בעזרת התרגילים המתאימים. הנחיות ניתנו לאורך מהלך ההוראה המומלץ לעיל.

א. מושגים המגדירים עץ (1)

ב. סריקות לעומק (2-5)

תרגילים אלו בדומה לשאלות על פעולות חיצוניות, מתרגלים את ההבנה של הממשק והשימוש בו. שאלה 2 מופיעה כשאלת מחשבה בספר בסעיף 2.ז.

ג. תרגילי מעקב (6-8)

<u>שאלה 7</u> – התשובה: הקוד מחזיר את מספר העלים בעץ עליו הוא פועל.

ד. בניית עצים (9)

** במאגר השאלות מופיעות שאלות בנייה נוספות.

<u>שאלה 9</u> – למרות שבנוסח השאלה נאמר: "פעולה בונה", צריך לשים לב שפעולה בונה חיצונית אינה פעולה בונה רגילה. היא בונה באופן מעשי עצם חדש על פי הכללים המפורטים בשאלה, אך שמה אינו זהה לשמו של הטיפוס המוחזר על ידה.

שימו לב שהפתרון המצורף מתייחס לעץ גנרי ולא לעץ של שלמים בלבד.

הערה: ניתן היה לכתוב את הפעולה גם כפעולה פנימית למחלקה BinTree = פעולה **בונה מעתיקה**, בתנאי שהמחלקה BinTree מגדירה עץ של מספרים שלמים או כל טיפוס שניתן להעתיק את ערכיו בצורה עמוקה.

```
public static BinTree<Integer> buildIdentTree
                                        (BinTree<Integer> tree)
{
      BinTree<Integer> left = null;
      BinTree<Integer> right = null;

      if(tree.getLeft() != null)
            left = buildIdentTree(tree.getLeft());

      if(tree.getRight() != null)
            right = buildIdentTree(tree.getRight());

      return (new BinTree<Integer>(tree.getInfo(),left,
                                                  right));
}
```

ה. פעולות נוספות (13-10)

שאלה 11 – הפעולה יכולה להיכתב באופן איטרטיבי או רקורסיבי.

תזכורת שכדאי לוודא שזכורה לתלמידים: **גובה עץ** (tree height) הוא המרחק הגדול ביותר מהשורש לעלה כלשהו, כלומר זו הרמה הגבוהה ביותר בעץ.

שאלה 12 – כדי לפתור שאלה זו מספיק שהתלמיד יחשב את מספר הרמות בעץ, ובמקביל יחשב את מספר הצמתים. השוואה בין שני אלו תיתן את התשובה הנכונה, על פי החישובים המצוינים בסעיף יא.4. בפרק.

שאלה 13 – **מומלץ לבצע.** שימו לב שבהגדרת הפעולה יש כפילות ואפילו חזרה משולשת על המושג BinTree. ה**עץ** הנשלח לפעולה הוא מטיפוס BinTree. גם **הערך המוחזר** שהוא **הפנייה לשורש** בתוך העץ tree וכן ה**ילד** הנשלח כ**פרמטר** – שגם הוא הפנייה לשורש של תת עץ ב-tree – כולם מטיפוס BinTree. בלבול?! יש להבין שהגדרת העץ כאוסף של חוליות בלבד, אומרת למעשה שהתייחסות לעץ בכל צומת ושורש מקומי היא כהתייחסות לעץ שלם ולכן כל ההפניות הן מטיפוס BinTree.

כדי לחדד נקודה זו שהיא גם נקודת הבחנה חשובה בין מבנה הרשימה לבין מבנה העץ, מומלץ מאד לפתור שאלה זו עם התלמידים ולדון בטיפוסים השונים.

הערה: בשאלה זו ניתן להשתמש במחסנית, שיכולה לדמות תהליכים רקורסיביים בשל המבנה המיוחד שלה. שאלות נוספות המשתמשות במחסנית באופן דומה נמצאות במאגר.

ו. עצים על פי הגדרה (14-15)

שאלה 14 – סעיף א. שימו לב שהפעולה סכום צמתים מופיעה כקוד בפרק בסעיף ז.1. ניתן להפנות את התלמידים להיעזר בה.

ז. עץ ביטוי (16-18)

שאלה 18 – מתייחסת לבניית עץ ביטוי. אנו מצרפים את האלגוריתם המתאים וכן איור דרכו ניתן לעקוב אחר הבנייה של העץ מתוך הביטוי החשבוני. ניתן לחלק את האיור ו/או את האלגוריתם לתלמידים ככלי עזר לפתרון התרגיל:

האלגוריתם הרקורסיבי בנה-עץ-ביטוי מקבל כקלט ביטוי חשבוני כמחרוזת תוים (כלומר יש צורך להכניס את הביטוי תו אחרי תו מהקלט), ומחזיר את העץ המתאים. אם התו הנקרא הוא מספר, יִבָּנֶה עלה המכיל אותו כערך. אם התו הוא סוגר שמאלי, זוהי תחילתו של ביטוי חשבוני מהצורה (X op Y), ואז יבוצעו הפעולות הבאות:

- בניית עץ המתאים לביטוי החשבוני השמאלי X, באמצעות קריאה רקורסיבית לאלגוריתם בנה-עץ-ביטוי.
- קריאת התו op.
- בניית עץ המתאים לביטוי החשבוני הימני Y, באמצעות קריאה רקורסיבית לאלגוריתם בנה-עץ-ביטוי.
- בניית עץ ששורשו op והתת-עצים השמאלי והימני שלו הם בהתאמה העצים שנבנו על פי הביטויים החשבוניים X ו-Y.

משהסתיימה בניית העץ המתאים לביטוי שבתוך הסוגריים, התו הבא במחרוזת צריך להיות סוגר ימני.

לפניכם האלגוריתם לבניית עץ המייצג ביטוי חשבוני.

בנה-עץ-ביטוי

{האלגוריתם מחזיר עץ המתאים לביטוי חשבוני הנקרא מהקלט.}

{ch ו-op מאחסנים תווים הנקראים מהקלט. T1 ו-T2 הם עצים.}

*{**הנחה**: הביטוי שמתקבל כקלט הוא ביטוי חשבוני תקין. }*

קרא את הקלט תו אחר תו, לגבי כל תו בצע:

אם התו הוא ספרה, בנה עץ עלה T שהתו הוא ערכו

אחרת, אם התו הוא ')', בצע:

בנה-עץ-ביטוי ← *T1. { בניית עץ לתת-ביטוי השמאלי.}*

קרא תו op. { op צריך להיות אופרטור.}

בנה-עץ-ביטוי ← *T2. { בניית עץ לתת-ביטוי הימני.}*

בנה-עץ T עם הערך op, ילד שמאלי T1 וילד ימני T2

קרא תו *{ התו צריך להיות י.}*

החזר את T.

האיור הבא מציג את תהליך בנייתו של העץ הבינרי לביטוי: $\underbrace{((3-2)}_{X} * \underbrace{((4*1)+8))}_{Y}$

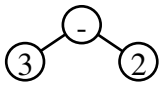
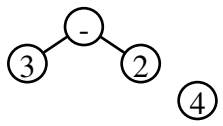
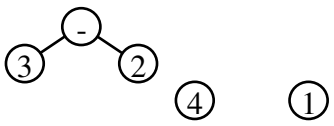
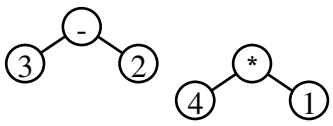
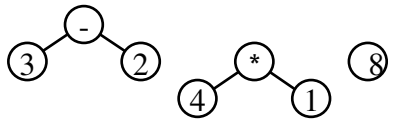
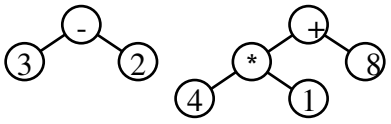
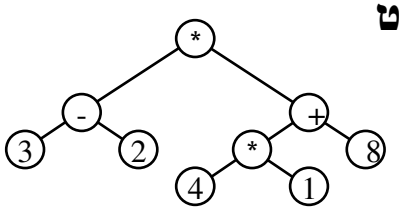
התו הראשון הנקרא הוא סוגר שמאלי, ולכן האלגוריתם פונה לבנייה רקורסיבית של עץ לתת-ביטוי השמאלי, X. בניית העץ המתאים מתבצעת בשלבים א-ג באיור:

בשלב א נקרא הסוגר השמאלי שב-X. קריאה רקורסיבית נוספת ל**בנה-עץ-ביטוי** יוצרת את הצומת 3.

בשלב ב מאוחסן התו 'י' ב-op, ו**בנה-עץ-ביטוי** מופעל על האופרנד 2.

בשלב ג נבנה עץ שבשורשו op. עם קריאת הסוגר הימני מסתיימת בניית העץ עבור הביטוי X.

לאחר בניית העץ המתאים ל-X מוכנס התו '*' ל-op, ומתחילה בניית עץ לתת-ביטוי הימני Y, בשלבים ד-ח באיור. בשלב ט מסתיימת פעולת האלגוריתם בבנייתו של עץ שבשורשו '*', והתת-עצים שלו הנם הביטויים המתאימים ל-X ול-Y.

א ((**3** - 2) * ((4 * 1) + 8))	**ב**  ((**3** - **2**) * ((4 * 1) + 8))	**ג**  ((**3 - 2**) * ((4 * 1) + 8))
ד  ((**3 - 2**) * ((**4** * 1) + 8))	**ה**  ((**3 - 2**) * ((**4 * 1**) + 8))	**ו**  ((**3 - 2**) * (**(4 * 1)** + 8))
ז  ((**3 - 2**) * (**(4 * 1)** + **8**))	**ח**  ((**3 - 2**) * (**(4 * 1) + 8**))	**ט**  **((3 - 2) * ((4 * 1) + 8)**

ח. עץ חיפוש בינרי (19-23)

שאלה 21 - יש להחליט על אופן הייצוג של עץ החיפוש. לנוחיותכם שני המימושים של המחלקה מצורפים באתר. שימו לב שפעולת המחיקה של איבר מעץ חיפוש אינה פשוטה. תיאור האלגוריתם מופיע לעיל בסעיף 26 של מהלך ההוראה. ניתן לוותר לתלמידים על מימוש פעולה זו.

ניתן לכוון את התלמידים לממש את העץ עם זוגות של מפתח-ערך, כלומר לממש את Map תוך שימוש בתכונה פנימית שהיא BinTree. הפעולות ימומשו בהתאמה.

שאלה 21 – מטרת השאלה לתרגל שימוש בעץ חיפוש בינרי. הכנסת הרשימה לעץ ושליפת הנתונים מהעץ שאינו מכיל חזרות תבצע את המשימה. דרך אחרת לכוון את התלמידים לשימוש בעץ היא הדרישה לממש את הפעולה בסיבוכיות שאינה עולה על *O(nlogn)*. אם לא ישתמשו בעץ חיפוש אלא ברשימה, סיבוכיות הפעולה תהיה גבוהה יותר. שימוש שכזה בעץ חיפוש נותן מוטיבציה לצורך בעץ שכזה. כמובן שניתן לבצע את המשימה גם בעזרת מפה.

שאלה 22 – לצורך המתואר בשאלה יש לאחסן בצמתי העץ זוגות של ערך ומספר מופעיו. זוהי למעשה מפה. אם עד כה לא מומש עץ חיפוש שה-info שלו הוא זוגות ולא רק שלמים, זו ההזדמנות לעשות זאת.

ט. שחזור עץ על פי סריקותיו (23-27)