מבני נתונים ומבוא לאלגוריתמים

נושא 8

עצי חיפוש בינאריים
Binary search trees

אמיר רובינשטיין

בתוכנית

פרק 12 בספר הלימוד

- נכיר את ה- ADT "מילון"
- נכיר את מבנה הנתונים "עץ חיפוש בינארי" למימוש מילון, ועוד כמה תכונות ואלגוריתמים שימושיים בעצים

מילון (dictionary)

<u>מילון</u> הוא ADT המוגדר ע"י פעולות הבאות:

- $\text{Insert}(S, x)$ – הכנסת האיבר אליו מצביע x למבנה S
- $\text{Delete}(S, x)$ – מחיקת האיבר אליו מצביע x מהמבנה S
- $\text{Search}(S, k)$ – חיפוש איבר שמפתחו k והחזרתו, אם קיים ב- S

ולפעמים גם פעולות חיפוש נוספות, כמו:

- $\text{Minimum}(S)$ – החזרת איבר בעל מפתח מינימלי
- $\text{Maximum}(S)$ – החזרת איבר בעל המפתח מקסימלי
- $\text{Successor}(S, x)$ – החזרת <u>עוקב</u> של x ב- S (איבר בעל מפתח מינימלי שגדול מזה של x)
- $\text{Predecessor}(S, x)$ – החזרת <u>קודם</u> של x ב- S (איבר בעל מפתח מקסימלי שקטן מזה של x)

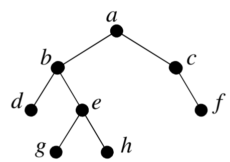
4 הפעולות האחרונות מניחות שיש סדר על מפתחות האיברים.

<u>למילון שימושים רבים ומגוונים (למשל?)</u>

מבני נתונים למילון

- במימוש של מילון בכל אחד ממבני הנתונים שלמדנו עד כה, לפחות אחת מהפעולות תתבצע בזמן ליניארי במספר האיברים.
- נראה כעת מימוש אפשרי למילון, באמצעות מבנה הנתונים <u>עץ חיפוש בינארי</u>. <u>בממוצע</u> כל הפעולות יבוצעו בסיבוכיות של $\Theta(\log n)$.
- בשיעור הבא נראה שיכלול של עצי חיפוש בינאריים - עצי AVL - שמבטיח סיבוכיות של $\Theta(\log n)$ גם במקרה הגרוע.

עץ ~~חיפוש~~ בינארי - תזכורת



<u>מונחים - תזכורת</u>

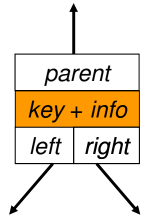
- צומת
- קשת

- שורש
- בן השמאלי, בן ימני, אב, אב-קדמון, צאצא
- תת-עץ שמאלי, תת-עץ ימני
- עלה, צומת פנימי

- עומק של צומת
- גובה של צומת, של עץ

- מימוש במערך, מימוש באמצעות רשומות עם מצביעים:

מבנה של רשומה (צומת):



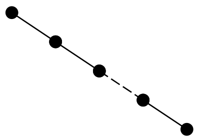
5

גובה של עץ בינארי

נסמן את מספר הצמתים בעץ בינארי ב- n, ואת גובהו ב- h.

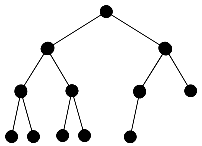
<u>מהו הגובה המינימלי ומהו הגובה המקסימלי של עץ בינארי, כתלות ב- n?</u>

מקסימלי: שרשרת או "זיג-זג"



$h = n\text{-}1 = \Theta(n)$

מינימלי: עץ (כמעט) שלם



$h = \lfloor \log n \rfloor = \Theta(\log n)$

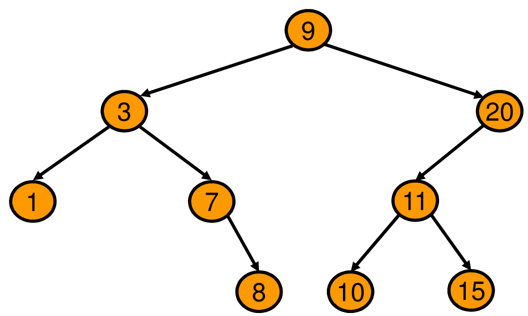
6

עץ חיפוש בינארי

<u>עץ חיפוש בינארי</u> (binary search tree) הוא עץ בינארי עם התכונה הבאה:

בהינתן צומת עם מפתח k: - צמתים בתת העץ הימני שלו בעלי מפתחות $\geq k$

- צמתים בתת העץ השמאלי שלו בעלי מפתחות $\leq k$.

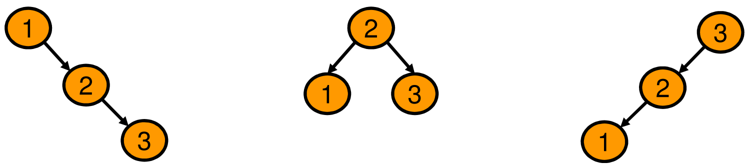


בהינתן קבוצת מפתחות, האם קיים עץ חיפוש בינארי <u>יחיד</u> המכיל מפתחות אלו?

7

עץ חיפוש בינארי

בהינתן קבוצת מפתחות, האם קיים עץ חיפוש בינארי <u>יחיד</u> המכיל מפתחות אלו?



עוד?

8

כמה עצים בינאריים שונים בעלי n צמתים קיימים?

נסמן את התשובה ב- b_n.

נוסחה רקורסיבית עבור b_n:

עבור $n = 0$: יש רק עץ אחד ללא צמתים – העץ הריק. לכן, $b_0 = 1$.

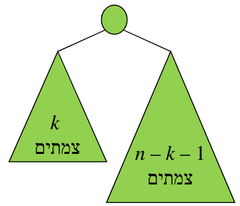
עבור $n \geq 1$: העץ מורכב מצומת אחד בשורש,

k צמתים בתת-עץ השמאלי $(k = 0, 1, ..., n-1)$,

ו- $n-k-1$ צמתים בתת-עץ הימני.

לכן:

$$b_n = \sum_{k=0}^{n-1} b_k \cdot b_{n-k-1}$$



הפיתרון לנוסחה (לא במסגרת הכלים שנלמדו בקורס זה):

$$b_n = \frac{1}{n+1}\binom{2n}{n} = \frac{4^n}{\sqrt{\pi} n^{3/2}}\left(1 + O(1/n)\right)$$

b_n נקראים **מספרי קטלן** (Catalan numbers).

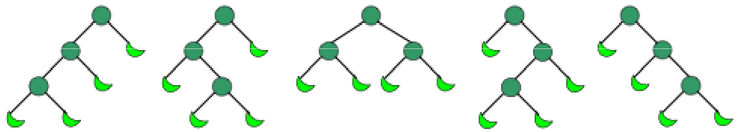
מספרי קטלן (Catalan numbers)

מספרי קטלן מייצגים גם את מספר הדרכים השונות לשים סוגריים על $n+1$ גורמים שונים.

לדוגמה, כאשר $n=3$ יש 5 דרכים שונות לשים סוגריים על 4 גורמים:

((ab)c)d   (a(bc))d   (ab)(cd)   a((bc)d)   a(b(cd))

ישנה פונקציה חח"ע ועל בין קבוצת הביטויים הנ"ל לבין קבוצת העצים הבינאריים בעלי n צמתים:



(תמונה מתוך הערך "מספר קטלן" בויקיפדיה)

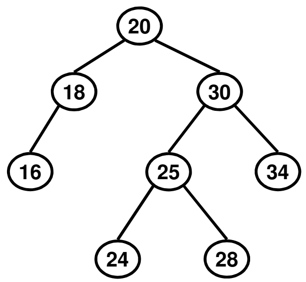
<u>חיפוש</u>

<u>כיצד נחפש מפתח בעץ חיפוש בינארי?</u>

```
Tree-Search(x, k)
1. if x = nil or k = key[x]
2.     return x
3. if k < key[x]
4.     return Tree-Search(left[x], k)
5.     else return Tree-Search(right[x], k)
```

בקריאה הראשונה x הוא שורש העץ.

מהו מסלול החיפוש של 26 בעץ הבא?



מה סיבוכיות הזמן והזיכרון הנוסף?

סיבוכיות זמן: $\Theta(h)$

סיבוכיות זיכרון נוסף: $\Theta(h)$

<u>חיפוש</u>

<u>ניתן גם לכתוב גרסה איטרטיבית:</u>

```
Tree-Search(x, k)
1. while x ≠ nil and k ≠ key[x]
2.     if k < key[x]
3.         x ← left[x]
4.     else x ← right[x]
5. return x
```

סיבוכיות זמן: $\Theta(h)$

סיבוכיות זיכרון נוסף: $\Theta(1)$

<u>שאלה</u>: מהי סיבוכיות הזמן במקרה הטוב?

שאילתות נוספות – מינימום ומקסימום

חיפוש משתייך לסוג של פעולות הנקראות שאילתות.

בפעולות אלו אין שינוי של מבנה הנתונים, אלא רק שליפת מידע ממנו.

שאילתות נוספות:

- החזרת איבר בעל מפתח מינימלי:

  (החזרת המקסימום - סימטרי)

```
Tree-Min(x)
1.  while  left[x] ≠ nil
2.        x ← left[x]
3.  return x
```

סיבוכיות זמן: $\Theta(h)$

ובמקרה הטוב?

שאילתות נוספות – עוקב וקודם

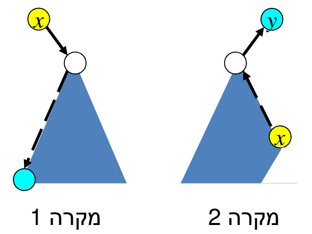
- החזרת האיבר העוקב לאיבר נתון x

מקרה 1: ל- x יש בן ימני

העוקב הוא המינימום של התת-עץ הימני של x

מקרה 2: ל- x אין בן ימני

העוקב (אם ישנו) הוא הצומת הראשון במעלה הדרך אל השורש שעולים אליו מבן שמאלי




```
Tree-Successor(x)
1.   if  right[x] ≠ nil                       } מקרה 1
2.         return Tree-Min(right[x])
3.   y ← parent[x]                            } מקרה 2
4.   while y ≠ nil  and  x =  right[y]
5.         x ← y
6.         y ← parent[y]
7.   return y
```

מימוש החזרת הקודם - סימטרי.

סיבוכיות זמן: $\Theta(h)$.

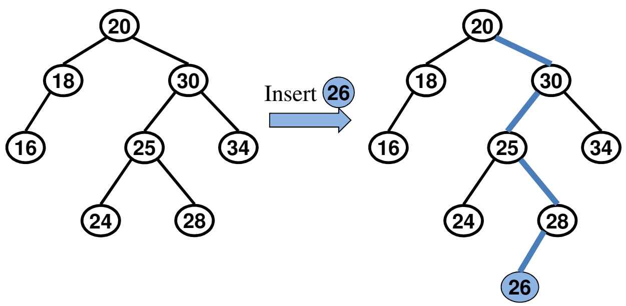
הכנסה

סוג אחר של פעולות הן פעולות דינמיות, המשנות את מבנה הנתונים (כמו הכנסה, מחיקה, שינוי).

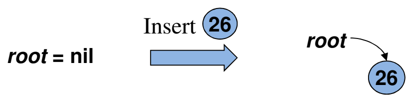
הכנסה

מחפשים את מקום ההכנסה המתאים (בדומה לחיפוש רגיל) ו"תולים" את האיבר החדש כעלה.

הדגמה



הכנסה לעץ ריק:



הכנסה

האלגוריתם:

רוצים להכניס צומת חדש z בעל מפתח $key[z]$ לעץ T.

מניחים ש- z מכיל שדות מצביעים לבן שמאלי וימני ולאב.

– שורות 1-7:

מחפשים אב y ש"מוכן לאמץ" את z

(כך שתישמר תכונת עץ החיפוש הבינארי)

– שורות 8-13:

מציבים את z כבן (שמאלי או ימני) של y

– שורות 9-10:

טיפול במקרה הקצה ש- z הוכנס לעץ ריק

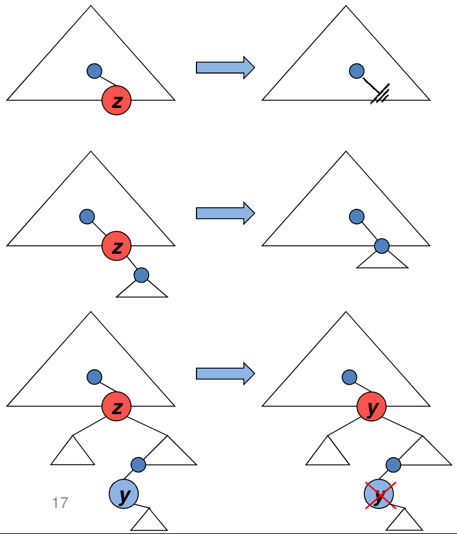
```
Tree-Insert(T, z)
1.  y ← nil
2.  x ← root[T]
3.  while  x ≠ nil
4.        y ← x
5.        if key[z] < key[x]
6.              x ← left[x]
7.        else x ← right[x]
8.  parent[z] ← y
9.  if y = nil
10.       root[T] ← z
11.       else if key[z] < key[y]
12.             left[y] ← z
13.       else right[y] ← z
```

סיבוכיות זמן: $\Theta(h)$.

מחיקה

רוצים למחוק מהעץ T צומת קיים, ונתון מצביע z לצומת זה.

- מאיפה יש לנו מצביע לצומת הזה?



מפרידים למקרים:

<u>מקרה 1</u>: z עלה

מציבים nil במצביע מאביו של z

<u>מקרה 2</u>: ל- z יש בן יחיד

נבצע "מעקף" מאביו של z לבנו

<u>מקרה 3</u>: ל- z יש שני בנים

יהי y העוקב של z.

נעתיק את תכולת y אל z,

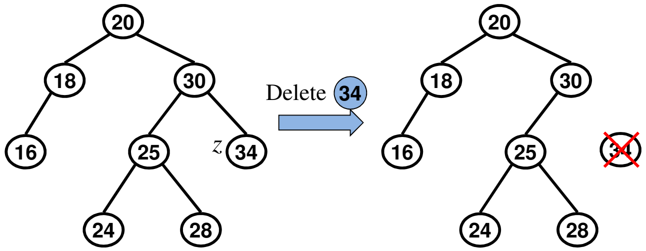
ונמשיך עם מחיקת y.*

* מחיקת y שייכת למקרים 1 או 2 (מדוע?)

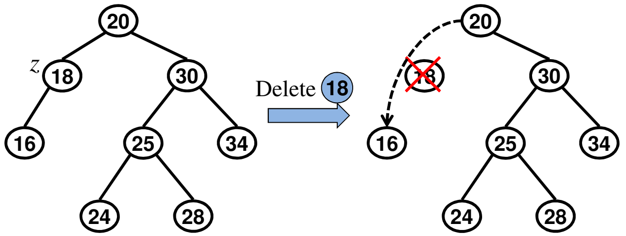
מחיקה

<u>הדגמה</u>

<u>מקרה 1</u>



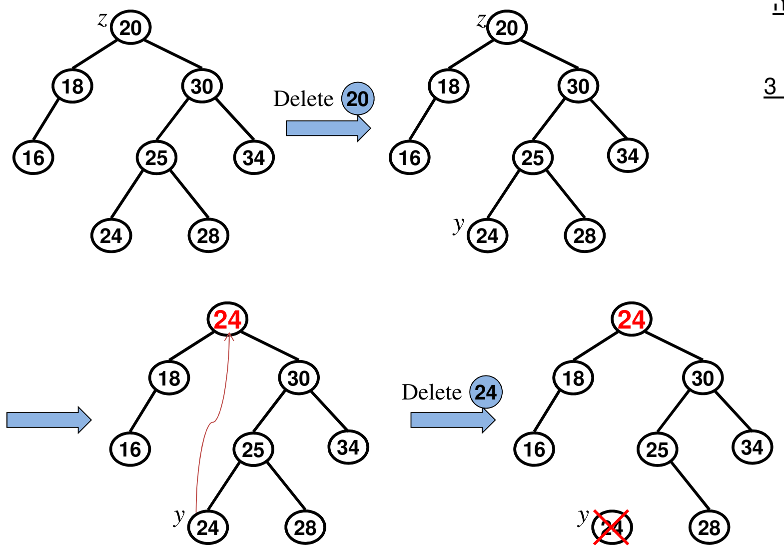
<u>מקרה 2</u>



מחיקה

<u>הדגמה</u>

<u>מקרה 3</u>



מחיקה

```
Tree-Delete(T, z)
1.  if left[z] = nil or right[z] = nil
2.    then y ← z
3.    else y ← Tree-Successor(z)
4.  if left[y] ≠ nil
5.    then x ← left[y]
6.    else x ← right[y]
7.  if x ≠ nil
8.    then p[x] ← p[y]
9.  if p[y] = nil
10.   then root[T] ← x
11.   else if y = left[p[y]]
12.          then left[p[y]] ← x
13.          else right[p[y]] ← x
14. if y ≠ z
15.   then key[z] ← key[y]
16.        ► copy y's satellite data...
17. return y ► for recycling
```

<u>האלגוריתם</u>:

בשורות 1-3 מציבים ב- y את הצומת שיוסר "פיזית" מהעץ:

- z עצמו, אם יש לו לכל היותר בן אחד
- העוקב של z, אם יש לו 2 בנים

סיבוכיות זמן: $\Theta(h)$.

סיורים בעצים

לעיתים אנו מעוניינים לעבור באופן שיטתי על כל צמתיו של עץ בינארי למטרה כלשהי.

פעולה כזאת נקראת סיור בעץ או סריקה של העץ.

נכיר 3 שיטות לסריקת עץ בינארי, הנבדלות זו מזו בסדר בו נסרקים הצמתים.

סיור תחילי (pre-order)

- מבקרים בשורש
- מסיירים רקורסיבית בתת העץ השמאלי
- מסיירים רקורסיבית בתת העץ הימני

```
Preorder-Tree-Walk(x)
if x ≠ nil
    Visit(x)
    Preorder-Tree-Walk(left[x])
    Preorder-Tree-Walk(right[x])
```

סיור תוכי (in-order)

הביקור בשורש נעשה בין הסיור בתת העץ השמאלי שלו לבין הסיור בתת העץ הימני שלו

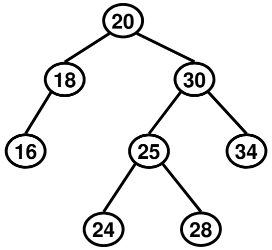
סיור סופי (post-order)

הביקור בשורש נעשה אחרי הסיורים בתתי העצים השמאלי והימני שלו

סיורים בעצים

תרגיל

רישמו את סדר הביקור בצמתים בכל אחד מהסיורים.



- סיור pre-order:
- סיור in-order:
- סיור post-order:

סיורים בעצים

טענה:

סיור in-order בעץ חיפוש בינארי, שבו Visit מדפיסה את מפתח הצומת, מניבה סדרה ממוינת (של מפתחות העץ).

הוכחה:

באינדוקציה על מספר הצמתים בעץ.

בסיס: עבור עץ עם צומת אחד הטענה בודאי נכונה.

צעד: עבור עץ עם יותר מצומת אחד, יודפסו קודם כל הצמתים בתת העץ השמאלי של השורש, אח"כ השורש ורק לאחר מכן כל הצמתים בתת העץ הימני שלו. לפי הנחת האינדוקציה, הסיור בכל אחד מתתי העצים מניב סדרה ממוינת, ולפי תכונת עץ החיפוש הסדרה כולה ממוינת.

סיורים בעצים

סיבוכיות זמן

טענה: כל אחד משלושת הסיורים הנ"ל רץ על עץ בעל n צמתים בזמן $\Theta(n)$.

אינטואיציה: עוברים דרך כל צומת 3 פעמים: בדרך לבנו השמאלי, בדרך לבנו הימני, ובדרך לאביו.

הוכחה שגויה: נוסחת הנסיגה המתאימה היא: $T(n) = 2T(n/2) + 1 = \Theta(n)$

מה הטעות?

הוכחה: נסמן את מספר הצמתים בתת-העץ השמאלי של השורש ב- k.

נניח שביקור בצומת דורש $d>0$ פעולות.

$T(0) = 1$

$T(n) = T(k) + T(n-k-1) + d$

נוכיח בשיטת ההצבה כי $T(n) = (d+1)n + 1$:

בסיס: $T(0) = (d+1)\cdot 0 + 1 = 1$

צעד: $T(n) = (d+1)k + 1 + (d+1)(n-k-1) + 1 + d = (d+1)n + 1$

סיורים בעצים – זיכרון נוסף

סיבוכיות זיכרון נוסף

שאלה: כמה זיכרון נוסף דורשים הסיורים במקרה הגרוע? מהו המקרה הגרוע?

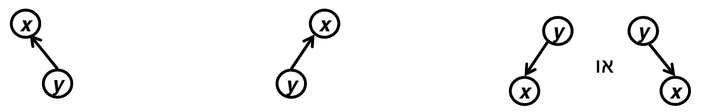
תשובה: עבור שרשרת (או "זיג-זג") עומק הרקורסיה הוא $\Theta(n)$.

שיפור: אפשר גם לממש את הסיורים עם $O(1)$ זיכרון נוסף (כמובן ללא רקורסיה).

הרעיון:

נשמור בכל רגע שני צמתים: הצומת בו אנו מבקרים - x, והצומת הקודם בו ביקרנו - y.

בצורה כזו אנו יודעים בכל רגע לאן עלינו להמשיך:



המשך הסיור: $x \leftarrow left[x]$, $y \leftarrow x$ | $x \leftarrow right[x]$, $y \leftarrow x$ | $x \leftarrow parent[x]$, $y \leftarrow x$

כמובן יש לטפל במקרי הקצה כנדרש (למשל כאשר x או y שווים Nil).

סיורים בעצים

תרגיל

א. רשימת המפתחות הבאה (משמאל לימין) התקבלה ע"י סריקה תחילית (pre-order) של עץ חיפוש בינארי:

20 10 5 2 8 30 27 40

שחזרו את העץ המקורי.

ב. האם ניתן לשחזר עץ חיפוש בינארי בהינתן סריקה post-order? in-order?

שימושים של סיורים – חישוב גובה

כיצד ניתן לחשב גובה של עץ בינארי?

(גובה של עץ ריק מוגדר להיות 1-)

```
height(x)
    if x = nil
        return -1
    else    return 1+ max{height(left[x]), height(right[x])}
```

סיבוכיות?

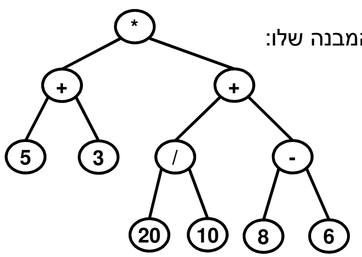
שימושים של סיורים - ביטויי postfix

כמובן, את כל הסיורים שראינו ניתן לבצע על עצים בינאריים שאינם עצי חיפוש.

דוגמה נחמדה ושימושית לכך היא חישוב ביטויים בכתיב postfix.

נסתכל על הביטוי הבא ועל העץ הבינארי שמייצג את המבנה שלו:

$(5+3)*((20/10)+(8-6))$



סיור inorder בעץ (כתיב infix):

5 + 3 * 20 / 10 + 8 - 6

הקריאה אינה יחידה – חייבים סוגריים

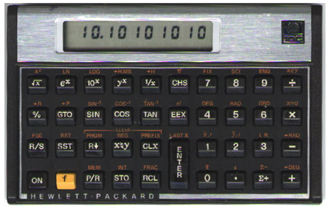
כתיב postfix: 5 3 + 20 10 / 8 6 - + *

קריאה יחידה – אין צורך בסוגריים

(כנ"ל בכתיב prefix)

ביטויי postfix נקראים **כתיב פולני הפוך**
ע"ש שם הלוגיקאי הפולני Łukasievicz
בביטוי postfix אין צורך בסוגריים: לכל ביטוי postfix יש לכל היותר פרוק יחיד!

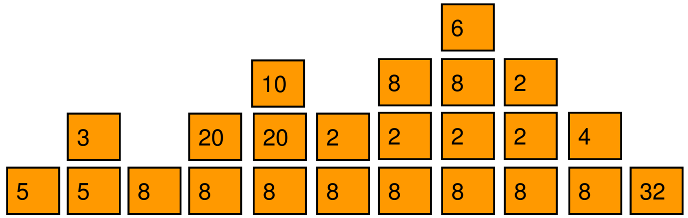
שימושים של סיורים - ביטויי postfix



HP-10c programmable scientific, 1980's. Cost: $80

האלגוריתם לחישוב ביטויי postfix

1. התחל עם מחסנית ריקה.
2. עבור על הביטוי משמאל לימין:
3. אם האיבר הבא הוא אופרנד- הכנס אותו למחסנית.
4. אם הוא פעולה – הפעל את הפעולה על שני האיברים שבראש המחסנית והכנס את התוצאה למחסנית.



postfix: 5 3 + 20 10 / 8 6 - + *

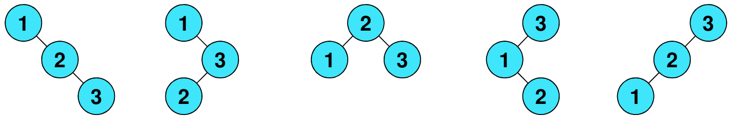
גובה ממוצע

ראינו קודם שהגובה המקסימלי של עץ בינארי הוא n-1, ואילו הגובה המינימלי הוא $\lfloor \log n \rfloor$.

שאלה: רוצים להכניס לעץ חיפוש בינארי את המפתחות 1,2,3.

- כמה סדרי הכנסה שונים אפשריים?
- אילו מהם יניבו עץ בגובה מקסימלי?
- אילו מהם יניבו עץ בגובה מינימלי?

<1,2,3> <1,3,2> <2,1,3> and <2,3,1> <3,1,2> <3,2,1>



גובה ממוצע הוא ממוצע הגבהים של העצים המתקבלים בסדרי ההכנסה השונים (בדוגמה: $1^2/_3$).

גובה ממוצע

משפט:

גובה ממוצע של עץ חיפוש בינארי בעל n צמתים הוא $\Theta(\log n)$, כאשר הממוצע הוא על פני כל סדרי ההכנסה השונים (הכנסות בלבד, ללא מחיקות או פעולות אחרות).

הוכחה בספר הלימוד, עמודים 223-225.

ניסוח אחר: כאשר סדר הכנסת האיברים הוא אקראי, תוחלת הגובה היא לוגריתמית במספר הצמתים.

שאלות חזרה

1. בקרו בקישור הבא, המציע סימולציה מצוינת של עצי חיפוש בינאריים (ועוד סוגים אחרים של עצי חיפוש):
http://people.ksp.sk/~kuko/bak/index.html
היעזרו בסימולציה כדי להבין את האלגוריתמים השונים על עץ חיפוש בינארי.

אתרי הדגמות נוספים, שעשויים לסייע:
א. http://nova.umuc.edu/~jarc/idsv/ (יש לבחור Search Trees בתפריט)
ב. http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html (יש לבטל את הסימון ליד AVL).

2. ציירו את כל העצים האפשריים המתקבלים ע"י הכנסת האיברים 1,2,3,4 לעץ חיפוש בינארי.
3. הציעו אלגוריתם רקורסיבי לחישוב מספר הצמתים בעץ בינארי. מהי סיבוכיות הזמן והמקום שלו?

תשובות לשאלות חזרה

3. האלגוריתם:

```
Count(x)
if x = NIL
    return 0
return 1 + Count(left[x]) + Count(right[x])
```

סיבוכיות זמן ליניארית במספר הצמתים.
סיבוכיות זיכרון ליניארית בגובה העץ.

תרגילים

תרגילים מומלצים מהספר

פרק 12

12.1-1
12.1-2
12.1-3
12.1-4
12.1-5
12.2-1
12.2-2
12.2-3
12.2-4
12.2-5
12.2-6
12.2-7
12.2-8*
12.2-9
12.3-1
12.3-2
12.3-3
12.3-4*
12.3-5
12.3-6

בעיה 12-1
בעיה 12-2 (שייך לנושא של מחרוזות)

תרגילים נוספים

1. נתון עץ בינארי בעל n צמתים, שבכל צומת שלו יש מספר חיובי.

הציעו אלגוריתם רקורסיבי לחישוב מסלול כבד ביותר מהשורש לעלה – מסלול מהשורש לעלה, שסכום ערכי הצמתים לאורכו הוא מקסימלי. הציעו גם דרך להדפיס את צמתי מסלול זה.

מה היה משתנה אילו המספרים היו לא בהכרח חיוביים?

2. סטודנט הפעיל סיורים *inorder* ו- *preorder* על עץ בינארי שהיה ברשותו. כעבור זמן מה, הוא גילה לתדהמתו שאיבד את העץ, אך יש עדיין בידו את תוצאות הסיורים.

עזרו לסטודנט לשחזר את העץ. מה סיבוכיות הפתרון במקרה הגרוע / הטוב ?

inorder : 2 6 4 7 1 3 8 5 9 10

preorder : 1 2 4 6 7 3 5 8 9 10

3. ניתן למיין קבוצה נתונה של n מספרים באופן הבא:

- תחילה בונים עץ חיפוש בינארי המכיל מספרים אלו, ע"י הכנסת האיברים בזה אחר זה לעץ.
- לאחר מכן מדפיסים את המספרים בסדר המתקבל ע"י סיור inorder של העץ.

מהו זמן הריצה במקרה הגרוע, במקרה הטוב, ובממוצע, של אלגוריתם מיון זה?

תרגילים נוספים

4. הוכיחו כי לא קיים אלגוריתם לבניית עץ חיפוש בינארי מרשימה נתונה של n איברים, שזמן ריצתו $o(n\log n)$.

5. ואריאציה של תרגיל 12.2-7 מספר הלימוד

ניתן לממש סריקה תוכית (in-order) של עץ חיפוש בינארי בעל n צמתים ע"י מציאת המינימום, ואח"כ מציאת העוקב של הצומת הנוכחי n-1 פעמים.

להלן חישוב חסם הדוק שגוי לזמן הריצה במקרה הגרוע:

- גובה העץ הוא $h=\Theta(n)$.
- כל אחת מפעולות Tree-Successor רצה בזמן $\Theta(h)$.
- לכן סה"כ זמן הריצה הוא $(n-1)\Theta(h) = \Theta(n^2)$.

הסבירו היכן הטעות ומדוע חישוב זה מתאים לחסם עליון בלבד (רמז: איפה צריך להחליף Θ ב- O?).

הסבירו מדוע למעשה זמן הריצה של אלגוריתם זה הוא $\Theta(n)$.

37

פתרון 1

```
Heavy-Weight(x)
    if x = nil
        return 0
    l ← Heavy-Weight(left[x])
    r ← Heavy-Weight(right[x])
    return key[x] + max{l, r}
```

תנו דוגמה שבה הפתרון אינו נכון, אם מותרים גם מספרים שליליים בצמתים.

מה צריך לשנות בפתרון במקרה כזה?

38

פתרון 4

נניח בשלילה שקיים כזה אלגוריתם.

אז ניתן למיין n איברים בזמן $o(n\log n)$ באופן הבא:

- נבנה את העץ בעזרת האלגוריתם הנ"ל, בזמן $o(n\log n)$
- נבצע סיור inorder בעץ להדפסת איבריו, בזמן $\Theta(n)$

סה"כ סיבוכיות הזמן היא $o(n\log n)$, וזה בסתירה לחסם התחתון לבעיית מיון ההשוואות שהוכחנו.

פתרון 5

אמנם, מציאת עוקב רצה במקרה הגרוע ב- $\Theta(h)$, אבל באופן הסריקה המוצג בשאלה רק חלק מהפעמים מתרחש המקרה הגרוע, ופעמים רבות מציאת העוקב תיקח זמן קצר יותר (תנו דוגמה למצב בו מציאת העוקב של צומת רצה בזמן קבוע). ניתן אם כן לתקן ולומר שכל אחת מפעולות העוקב רצה בזמן $O(h)$, ושהחסם העליון לזמן הריצה הכולל הוא $O(n^2)$.

חסם הדוק:

נשים לב שסדר המעבר על קשתות וצמתי העץ זהה לחלוטין לזה שבסריקה in-order.

כלומר עוברים בכל צומת ובכל קשת אותו מספר פעמים ובאותו סדר בדיוק.

ידוע שסריקה inorder בעץ רצה בזמן ליניארי במספר צמתיו.

39