① (21/10/12)

מבוא

יורם לוזון louzouny@math.biu.ac.il 03-5317165

ספר של NORMAN

תרגילים 20-25%.

אלגוריתם - דרך לפתור בעיה. מקבלים קלט, ע"י חישוב (שנתון ע"י האלגוריתם) נקבל פלט.

החישוב נעשה במעבד (שעון) ודורש שני משאבים:

1) זמן חישוב

2) זכרון (ב- RAM או ב- CASH של המעבד)

דוגמאות

* כידוע, מערכים דו מימדיים נשמרים בזכרון כמערך אחד ארוך. למשל המערכים של 3x1000 ושל 1000x3, מבחינת יעילות הגישה לזכרון (במעבד) יהיה שונה בגלל גודל הקפיצות.

* עבור מערך שרוצים למיין, 13,20,-1,9,5 אורך הקלט הוא $n=5$.

* בעיית הסוכן הנוסע (TSP) - סוכן רוצה לעבור בין ערים שונות ביום אחד ע"י המרחק הקצר ביותר. גודל הקלט הוא כמס' הערים.

מה העלות של חישוב

נגדיר במס' הפעולות (סדר גודל) (והזכרון).

* עלות של הבעיה היא $O(f(n))$ אם קיים n_0, c כך שלכל $n > n_0$ עלות הבעיה קטנה מ- $cf(n)$.

דוגמאות

* עבור TSP, מס' האפשרויות למעבר בין ערים הוא $n!$ ידוע $(\frac{n}{e})^n \cdot$ החסם $\approx n!$ כלומר $n! = O(n^n)$, ובקצב בין 50 ל-60 ערים אנו עוברים מ-10^{60} פעולות.

* עבור סדרת מס' נבצע מיון ע"י לקיחת הגדול (נמצא אותו ע"י מעבר על כולם) ונעבירו להתחלה.
כמות הפעולות שנוי היא

$$n+(n-1)+(n-2)+\dots+1=\frac{n(n+1)}{2}\approx\frac{1}{2}n^2=O(n^2)$$

* עבור 3 נוסחות שונות עבור מס' הפעולות כתלות לגודל הקלט

מס' פעולות \ גודל קלט	10	100	1000	10000
$10n$	100	1000	10000	100,000
$0.2n^2$	20	2000	200,000	20,000,000
$0.01\cdot 2^n$	10	10^{28}		

ואנו רואים שאפילו עבור קבוע קטן יש הבדל גדול מאוד.

הגדרות

* נגדיר $g(n)=O(f(n))$ אם החל מ-n מסוים קיים $c>0$ עבורו $g(n)\le c\cdot f(n)$
* נגדיר $g(n)=\Omega(f(n))$ אם החל מ-n מסוים קיים $c>0$ עבורו $g(n)\ge c f(n)$
* נגדיר $g(n)=\Theta(f(n))$ אם $g(n)=O(f(n))$ וגם $g(n)=\Omega(f(n))$

דוגמה

$$7n^3\le f(n)=7n^2+8n^3+1027+252n+10^9\sin(35e^n)\le 11n^3$$

ונקבל $n^3=\Theta(f(n))$

* כל חזקה יותר גבוהה מכל log (למשל $\log^{20}n < n^{0.1}$).
כל אקספוננט יותר גדול מכל חזקה (למשל $1.1^n > n^{27}$).

* את האלגוריתם אפשר לבצע בשתי צורות איטרטיבית ורקורסיבית.

דוגמה

מיון רשימה ע"י quick-sort: בוחרים איבר כלשהו, את הקטנים ממנו שמים משמאלו ואת הגדולים ממנו שמים מימינו.

הנוסחה הרקורסיבית תהיה $Q(n) = \underbrace{n}_{\text{עלות ההעברות}} + \underbrace{2Q(\frac{n}{2})}_{\text{בהנחה שחתכנו באמצע}}$

$$Q(n) = n + 2Q\left(\frac{n}{2}\right) = n + 2\left[\frac{n}{2} + 2Q\left(\frac{n}{4}\right)\right] = 2n + 4\left[\frac{n}{4} + Q\left(\frac{n}{8}\right)\right] = \ldots =$$

$$= kn + 2^k Q\left(\frac{n}{2^k}\right).$$

נפסיק כאשר $\frac{n}{2^k} = 1$ כלומר $k = \log_2 n$

$$\Rightarrow Q(n) = \log_2 n \cdot n + \underbrace{1 \cdot 2^{\log_2 n}}_{n} \approx n \log_2 n$$

ובעצם $Q(n) = \Theta(n \cdot \log_2 n)$

(22/10/12) ②

ספרי גורפל

תזכורת

$f(n)=O(g(n))$ – חסם עליון

$f(n)=\Omega(g(n))$ – חסם תחתון

$f(n)=\Theta(g(n))$ – גם חסם עליון ותחתון

כמו כן ראינו שהחל מ-n כלשהו:

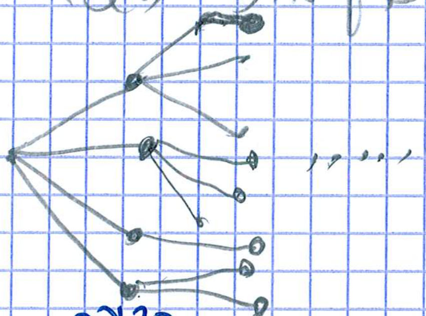
כל חזקה a^n גדולה מחזקה n^a

כל חזקה n^a גדולה מחזקה $\log^c n$

מכונה סדרתית – ניתן לבצע פעולה אחת בלבד בו זמנית.

מחשב מקבילי (לא דטרמיניסטי) – ניתן לבדוק את כל האופציות בזמן מקביל.

כך למשל ניתן לבצע את כל הבדיקות הבאות בזמן הנוסף במקביל ביעילות (הרבה):



בדיקה 1 בדיקה 2

בעיית P – אלגוריתם שניתן לפתור בזמן פולינומיאלי במכונה סדרתית.

בעיית NP – אלגוריתם שניתן לפתור בזמן פולינומיאלי במכונה מקבילית.

רדוקציה של בעיות

תהיינה שתי בעיות A ו-B כך ש-A הרבה יותר מורכבת, אבל שניתן לפתור אותה תוך שימוש ב-B.

למשל: A – מיון סדרת מספרים, B – מציאת מקסימום.

אזי, אם העלות של A היא פולינומיאלית כשמניחים שהעלות של B היא 1, נקבל שאם B פולינומיאלית גם A פולינומיאלית וניתן לקשר רדוקציה פולינומית/פולינומיאלית של A ו-B. כלומר (עלות הבעיה) $f(n)=O(g(n))\le\Theta(g(n))$.

ואם הבעיות של כל קריאה היו פולינומיאליות,
קיבלנו שיש הדימות ע"י מכפלת פולינומים שהם
פולינום, כלומר חסומה ע"י (פולינום)ס.

דוגמה

עבור בעיית הסוכן הנוסע נגדיר שתי בעיות:

1) מה המסלול הקצר ביותר (בעיית אופטימיזציה)
2) האם קיים מסלול שעלותו קטנה מ-L (בעיית הכרעה)

בעיה 1 מורכבת יותר מבעיה 2.

דוגמה

נניח שיש עצמים שונים הוגים לארוז, נניח שמכל עצם נקח כמות x_i, כך שנפח כל יחידה v_i ועלותה הכספית c_i.

נרצה $\sum_i x_i v_i \le V_{tot}$ כך שנקבל את מקס' הערך $\sum_i c_i x_i$.

נקח על בסיס זה שתי בעיות:

1) מהו הצירוף שנותן ערך מקס'? (בעיית אופטימיזציה)
2) האם קיים צירוף שנותן ערך הגבוה מ-L עבור נפח V_{total} נתון (בעיית הכרעה).

*** מעבר מפתרון בעיית הכרעה לפתרון בעיית אופטימיזציה**

נסתכל עבור בעיית הסוכן הנוסע:

א) נמצא את האורך המדויק של המסלול האופטימלי (ניתן לעשות זאת בזמן הגיוני ע"י שיטת החצייה והרחבה שלנו).

ב) עבור כל צלע בגרף נחליף את העלות בהמון. לאחר ההחלפה עבור צלע מסוים נבדוק שוב האם יש מסלול שאורכו כנ"ל: אם כן אז הצלע לא חייב להיות במסלול ואם לא אז הצלע חייב במסלול. עבור חלק זה, נקבל ([illegible] חלקים), ולכן באופן כללי נוכל לפתור הבעיה בזמן פולינומיאלי.

נוסחאות נסיגה

$$T(n) = b\,T\left(\frac{n}{a}\right) + f(n)$$

מס' הקריאות לרקורסיה

מס' החלקים שאליהם המערך מחולק

משפט מאסטר

עבור נוסחה $T(n) = b\,T\left(\frac{n}{a}\right) + f(n)$, יש שלוש אפשרויות:

1) $f(n) = \Omega(n^{\log_a b + \varepsilon})$ עבור $\varepsilon > 0$, אזי $T(n) = \Theta(f(n))$

2) $f(n) = O(n^{\log_a b - \varepsilon})$ עבור $\varepsilon > 0$, אזי $T(n) = \Theta(n^{\log_a b})$

3) $f(n) = \Theta(n^{\log_a b})$ אזי $T(n) = \Theta(n^{\log_a b} \cdot \log n)$

(3) (30/10/21)

קשיות

תזכורת

עבור אלגוריתם שמקבל קלט באורך n, ומוציא פלט באורך $f(n)$, נדון בשני סוגי קשיות: זכרון $(g_1(n))$ ועיבוד $(g_2(n))$. נעסוק בעיקר בקשיות עיבוד. מבחינת זמן - זה משתנה ממחשב למחשב ולכן לא נתייחס לזה כמעט.

נתייחס לקשיות בצורה $f(n)=O(g(n))$ שכן המקדמים יכולים להשתנות דרסטית מהפעלה להפעלה.

נאמר $f(n)=O(g(n))$ אם קיים $c>0$ כך שלכל n גדול: $f(n)\leq cg(n)$

$f(n)=\Omega(g(n))$ אם קיים $c>0$ כך שלכל n גדול: $f(n)\geq cg(n)$

$f(n)=\Theta(g(n))$ אם $f(n)=O(g(n))$ וגם $g(n)=O(f(n))$

P - אלגוריתמים שרצים בזמן פולינומיאלי במחשב דטרמיניסטי

NP - אלגוריתמים שרצים בזמן פולינומיאלי במחשב שאינו דטרמיניסטי.

ברור שמתקיים, אבל לא ידוע אם $P \subsetneq NP$.

רדוקציה פולינומיאלית - פותרים אלגוריתם A באמצעות בעיה B.
אם כאשר פותרים את B ב-$O(1)$ אז A נפתר בזמן פולינומיאלי, אזי ניתן לבצע רדוקציה פולינומיאלית

בעיית זיהוי - האם קיים פתרון?

בעיית אופטימיזציה - מהו הפתרון?

ובמקום לפתור בעיית אופטימיזציה ניתן לבצע רדוקציה פולינומיאלית. כלומר אם התנאי מתקיים, אז אם B נפתרת בזמן פולינומיאלי אז גם (A) לבעיית זיהוי.

דוגמה

(A) מה ההשקעה הכי חכמה בבורסה?

(B) האם קיימת השקעה שתניב רווח של מעל $x\%$?

בעיית רקורסיה $T(n)=aT(\frac{n}{b})+f(n)$ צריך לפתור ע"י אינדוקציה/הצבה/משפט מאסטר

דוגמה

$$T(n)=2T(\tfrac{n}{2})+3n$$

נשתמש באינדוקציה נוכיח $T(n)=O(n\log n)$, כלומר
שהחל מ-n_0 מסוים קיים c עבורו
$$T(n)\le cn\log_2 n$$

$n=2$: $T(2)=2T(1)+6$, ותמיד קיים c עבורו
$$2T(1)+6\le 2c$$

נניח נכונות עד n, ונוכיח עבור $2n$:
$$T(2n)=2T(n)+6n\le 2cn\log_2 n+6n=c[2n\log_2(2n)]$$
$$-2nc\log_2 2+6n=$$
$$=c[2n\log_2(2n)]+(3-c)n\underset{c\ge 3}{\le} 2cn\log_2(2n)$$

דוגמה

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

נוכיח $T(n)=O(n^2)$

$n=1$: $T(1)\le T(1)\cdot n^2=T(1)\cdot 1^2$ $\quad (c\ge T(1)\Leftarrow)$

נניח נכונות עד n, נוכיח עבור $n+1$:
$$T(n+1)=T(n)+n+1\le cn^2+(n+1)\underset{c\ge 1}{\le} cn^2+cn+c<cn^2+2cn+c=c(n+1)^2$$

דרך נוספת:
$$T(n)=T(n-1)+n=T(n-2)+n+(n-1)=\dots=$$
$$=T(1)+1+2+\dots+n=T(1)+\frac{n(n+1)}{2}\le T(1)+n^2$$
זה נותן אינטואיציה שזה $O(n^2)$, אבל זה לא מוכיח.

דוגמה

נותן אינטואיציה עבור
$$T(n)=2T(\tfrac{n}{2})+3n=$$
$$=2[2T(\tfrac{n}{4})+3\cdot\tfrac{n}{2}]+3n=2[2[2T(\tfrac{n}{8})+3\cdot\tfrac{n}{4}]+3\cdot\tfrac{n}{2}]+3n=$$
$$=2^kT(\tfrac{n}{2^k})+3kn\underset{n=2^k}{=}nT(1)+3n\log_2 n$$
נניח $n=2^k$ וסיום

נותן אינטואיציה עבור $T(n)=O(n\log_2 n)$

פואנה

$$T(n)=3T\left(\frac{n}{2}\right)+n^2=$$

$$=3\left[3T\left(\frac{n}{4}\right)+\frac{n^2}{4}\right]+n^2=3\left[3\left[3T\left(\frac{n}{8}\right)+\frac{n^2}{16}\right]+\frac{n^2}{4}\right]+n^2=$$

$$=\dots=3^{r}T\left(\frac{n}{2^{r}}\right)+\left[1+\frac{3}{4}+\frac{9}{16}+\dots\right]n^2=$$

$$n=2^{r}$$

$$=3^{\log_2 n}T(1)+\left[1+\frac{3}{4}+\frac{9}{16}+\dots\right]n^2=$$

$$3^{\log_2 n}=n^{\log_2 3}$$

$$=n^{\log_2 3}T(1)+cn^2$$

ומכאן נקבל אינטואיציה $T(n)=O(n^2)$. ממשפט מאסטר ניתן
להוכיח בקלות.

מבני נתונים

מחסנית

עובד על העקרון LIFO = Last In, First Out

אפשר לממש ע"י מערך בו נגיע למקום האחרון בלבד, או ע"י רשימה מקושרת בה נגיע למקום הראשון בלבד.

היתרון - כל הפעולות הן (המחסנית) $O(1)$

אתחול - נגדיר מערך A, נגדיל $n=0$. לא איכפת לנו שבמערך יש זבל בהתחלה, שכן אנו מתייחסים כאילו המחסנית ריקה ולכן לא יכולים לגשת.
ברשימה מקושרת נגדיר פוינטר ל-NULL.

PUSH - מכניסים איבר x כלשהו למחסנית.

במערך: אם n בתחום המערך, אז $A[n]=x$, $n++$.
אחרת, נודיע שהמחסנית מלאה.

ברשימה מקושרת: נקצה מצביע p_1, כאשר ראש המחסנית הנוכחי נקרא p.

```
p1.next=p;
p1.value=x;
p=p1;
```

y=POP(A) - הוצא איבר ראשון והכנס אל y.

במערך: אם $n>0$ אז $--n$ ואז $y=A[n]$
אם $n=0$ הודע שהמחסנית ריקה.

ברשימה מקושרת: אם p≠NULL:

```
y=p.value;
p_temp=p;
p=p->next;
delete p_temp;
```

אחרת המחסנית ריקה.

ISEMPTY - האם המחסנית ריקה?

EMPTY - במערך: $n=0$, ברשימה מקושרת: כל עוד המחסנית לא ריקה הוצא (פופ).

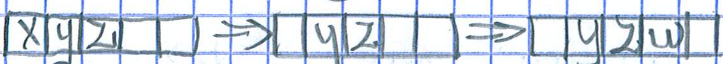
4 (4/11/12)

מבני נתונים

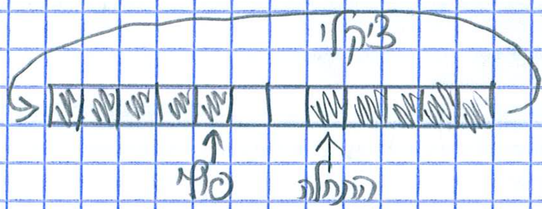
תור

עובד על עקרון FIFO - First In First Out

"מימוש תור במערך: מכניסים לסוף ומוציאים מההתחלה.
זה בעייתי שכן כל התור זז ימינה.



פתרון אחד הוא להגדיר את התור כמעגלי - מהסוף חוזרים להתחלה.



לשם כך נשמור משתנים start, end ו-l (אורך), ובהתחלה נגדיר את כולם כאפס. נניח גודל המערך כ-N.

"מימוש תור ברשימה מקושרת: מכניסים לסוף הרשימה ומוציאים מההתחלה.

נשמור מצביעים להתחלה ולסוף (ונאתחל את שניהם ל-NULL (אפשר גם שהסוף יצביע להתחלה ואז לא צריך לשמור (פוינטר) להתחלה).

ISEMPTY: במערך: התנאי L==0
ברשימה: התנאי start==NULL

ENQUE (הכנסה, יוסף לתור):

במערך: ראשית נבדוק אם L לא גדול או שווה מגודל המערך (N):

A[end] = ערך נתון; — true
end=(end+1)%N; // חישוב עבור מעגליות
l++;

שגיאה - false או שמגדילים המערך וממשיכים באותו אופן.

ברשימה מקושרת: מקצים זכרון עבור חוליה p, ואז:

```
p.data=x;
p.next=NULL;
```

אם start==NULL אז `start=p; end=p;`

אחרת
```
end.next=p;
end=end.next;
```

הוצאה מתור: (DEQUE)

מערך: בדיקה האם המערך ריק אם לא-
```
y=A[start];
start=(start+1)%n;
l--;
```

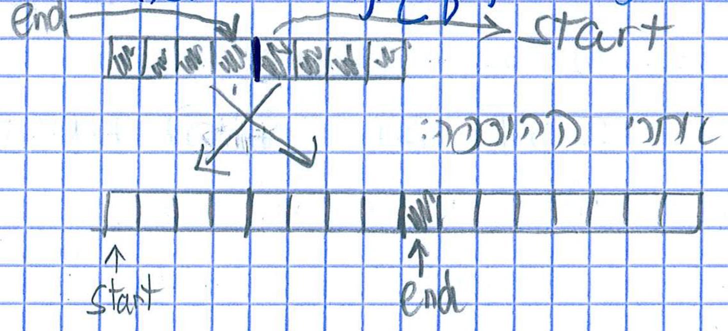
ברשימה: בדיקה האם start==NULL, אם לא-
```
y=start.value;
temp=start;
start=start.next;
free(temp);
```

ריקון התור:

מערך:
```
l=start=end=0;
```

ברשימה: להוציא את כל האיברים (אחד אחד)

מה קורה כשמכניסים למערך (והוא מלא):



תור קדימויות

נגמר תור רגיל, אך דברים עם קדימויות גבוהות יצאו לפני

מס' קדימויות אפשרי קטן: מחזיקים תור לכל קדימות

מס' קדימויות אפשרי גדול:

1) בהכנסה מכניסים לפי קדימות

במערך: $O(1)$ עבור מציאת מקום [ע"י שיטת החציה]

$O(n)$ עבור הכנסה [כי כל אלו שאחרי זה שהוכנס צריכים לזוז]

ברשימה: $O(n)$ עבור מציאת מקום [צריך לעבור על כולם]

$O(1)$ עבור הכנסה).

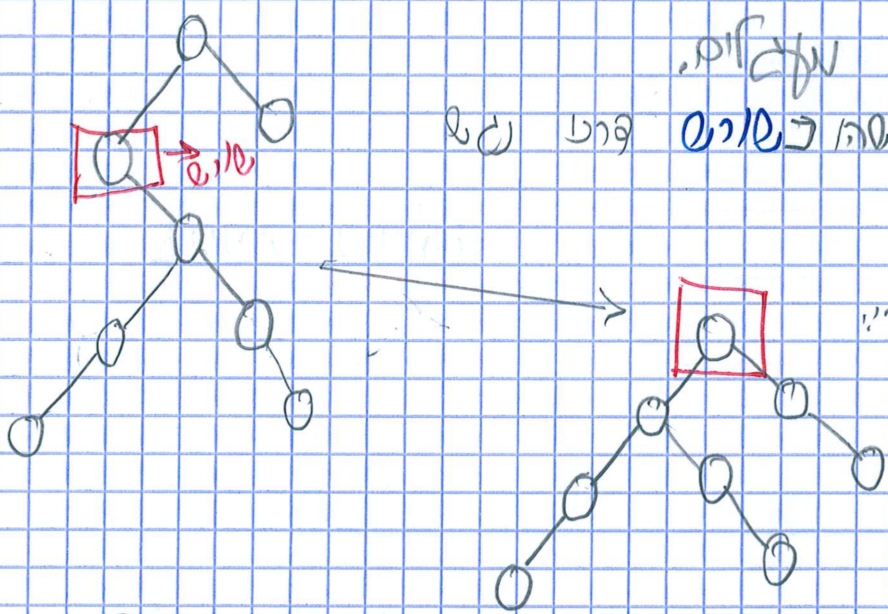
2) בהוצאה מוציאים לפי קדימות

3) ע"י ערימה (שבה גם הכנסה וגם הוצאה הם ב- $O(\log n)$).

עץ

גרף קשיר ללא מעגלים.

מתייחסים לקודקוד כלשהו כשורש פרט נגיש דרכו הגרף, למשל:



עץ עם שורש ייקרא עץ בינארי אם לכל קודקוד יש לכל היותר שני קודקודים היוצאים ממנו לרמה הנמוכה ממנו.

קודקוד שממנו יוצאים קודקודים ברמה נמוכה יותר ייקרא קודקוד פנימי.

עלה הוא קודקוד שאינו פנימי.

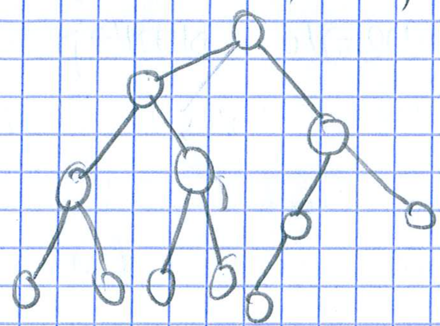
עץ מלא - אם לכל קודקוד פנימי יש שני קודקודים מתחתיו.

עומק של קודקוד - מס' הרמות בינו לבין השורש

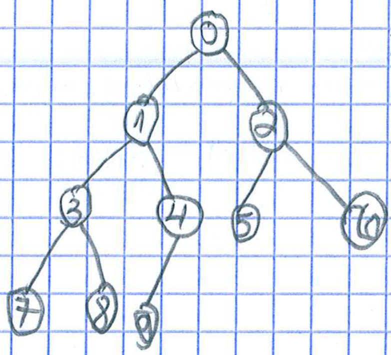
* עץ בינארי ייקרא שלם אם לכל הקודקודים הפנימיים (פרט לקודקודים בעומק הלפני האחרון) יש בדיוק שני קודקודים מתחתיהם.

בעומק הלפני אחרון נדרוש שכל הקודקודים
שבחלק השמאלי של העץ יהיו שני בנים, פרט
לאחרון מביניהם לו יתכן בן שמאלי בלבד.

שלם:



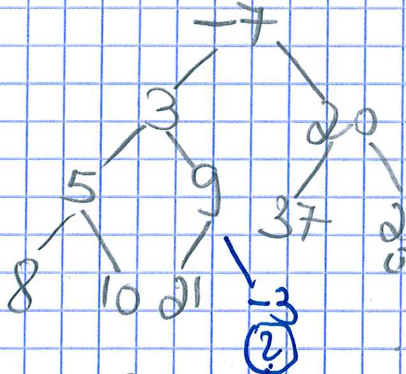
ייצוג עץ שלם במערך:



רמה:	1	2		3				4		
אינד:	0	1	2	3	4	5	6	7	8	9

ובמערך מקום האבא של קודקוד
מסוים יהיה $\left\lfloor \frac{x-1}{2} \right\rfloor$.
הבן השמאלי יתקבל ע"י $2x+1$,
והבן הימני יתקבל ע"י $2x+2$.

ערימת מינימום: מוגדרת כ-
1) עץ שלם
2) כל אבא קטן מילדיו



-7	3	20	5	9	37	23	8	10	21	-3

הכנסת -3:
1) נכניס למקום הראשון הפנוי (סוף הערימה)
2) אם -3 קטן מהאבא: נחליף ביניהם
נמשיך כך כל זמן הוא קטן מהאבא החדש שלו.
לבסוף נקבל

-7	-3	20	5	3	37	23	8	10	21	9

בסה"כ, ההכנסה של איבר x היא תמיד בתהליך זה ועושים
ה- $O(\log n)$ פעולות.

הוצאת שורש מערימה:

א. הוצאו שורש

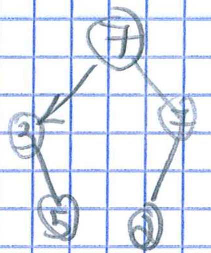
ב. הכנס את האיבר האחרון בערימה לשורש.

ג. כל עוד הנ"ל גדול מאחד מבניו, החלף עם הבן הקטן.

כך קיבלנו שהוצאה תעלה $O(\log n)$ לכל היותר.

עצי חיפוש על בינאריים כלשהו. ייצוגים:

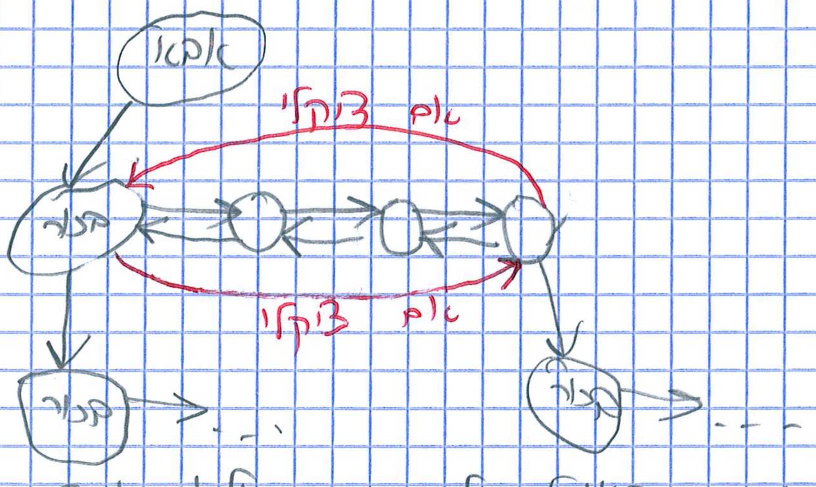
1) **במערך:** מניחים שלכל קודקוד יש שני בנים (ואם אחד מהם לא קיים מכניסים למערך INF-), למשל



7	3	7	-INF	5	2	-INF

2) **במצביעים לבנים:** קודקוד מכיל מצביעים לילדים שלו (למשל left ו- right). גם כאן אנו מוגבלים לכמות סופית של ילדים.

3) **במצביעים לבכור ולאחים:**



לומר כל קודקוד מחזיק פוינטר לבן בכור, לאח אחד (ואם צריך אז גם לאח הקטן ביותר). ניתן למימוש ע"י רשימה מקושרת של בנים.

(6/11/12) 5

מבני נתונים

בכל מבני הנתונים שהגדרנו בשיעורים הקודמים, חישבנו העלות תהליך גם על המקרה הממוצע וגם על מקרה מרבי.

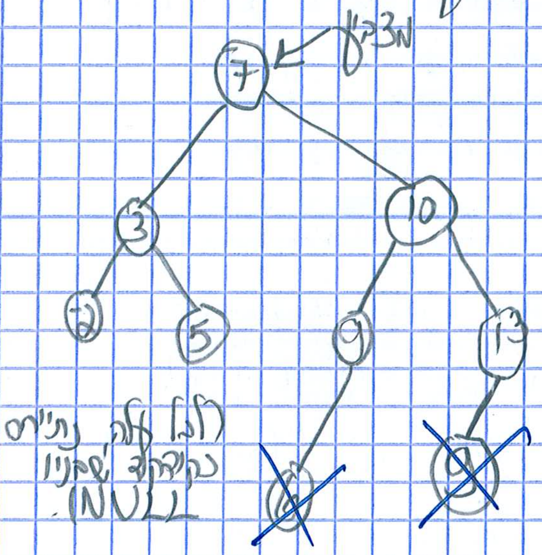
עצי חיפוש

גם כאן נרצה למצוא $f(n)=O(g(n))$, אבל כאן נצטרך למצוא הבחנה בין המקרה הטיפוסי למקרה הגרוע ביותר.

נדרוש:

1) ערך משמאל קטן מהשורש.
כך נרצה שגם כל האיברים מתחת העץ השמאלי יהיו קטנים מהשורש.

2) ערך מימין קטן מהשורש.
כך נרצה שגם כל האיברים מתחת העץ הימני יהיו קטנים מהשורש.



אלגוריתם חיפוש של x:

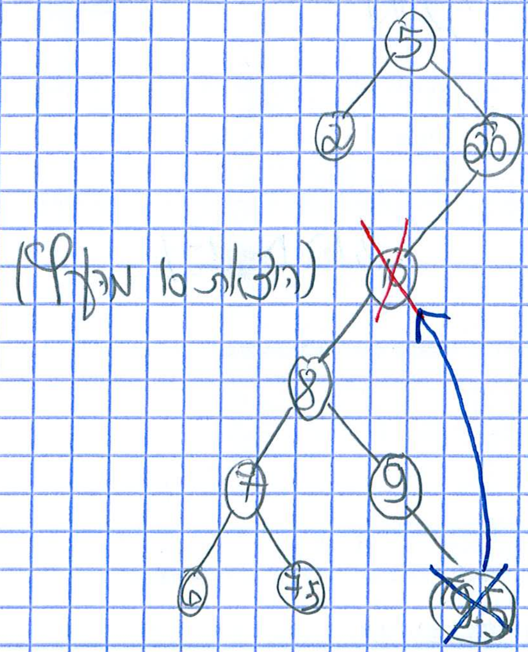
א) מצביע ל-NULL – x לא נמצא

ב) יש לנו כלשהו:
1) מצאנו את x
2) הערך גדול מ-x – חפש מתחת העץ השמאלי.
3) הערך קטן מ-x – חפש מתחת העץ הימני.

הכנסה של x לעץ:

א) חפש את x ושמור מצביע לערך שיהיה מעליו אם נכניס.
1) x קיים – לא עושים כלום.
2) x לא ק"ח ונגדל מהערך של האבא של המקום האחרון שחיפשנו אז תכניס את x כבן ימין של הפוינטר שמצאנו.
3) כנ"ל במקרה של קטן (הכנסה משמאל).
4) אם השורש ריק, הכנס אליו את x.

הוצאה של x מעץ:



אחפש את x ונשמור מצביע לאבא

אם x לא נמצא - סיימנו

אם x נמצא והוא עלה-

מחק את הזכרון של x ובאבא

מחק את המצביע אליו

אם ל-x בן יחיד-

1) שמור מצביע זמני ל-x.

2) חבר את האב של x לבן של x

(במקום x)

3) מחק את הזכרון של x.

אם ל-x שני בנים (ראה דוגמה)-

החלף את הערך של הבן הכי ימני בתת העץ שלו

או הכי שמאלי בתת העץ הימני, ומחק את הקודקוד

הזמני (לוקח הערך לפי המקרים הקודמים).

ריקון עץ (ברקורסיה): $O(n)$

מחק תת עץ שמאלי, תת ימני ואת הקודקוד עצמו.

דוגמה: מילוי עץ:

מקרה גרוע - הנתונים מסודרים ⇐ $O(n)$ למשל (1,2,3,4,5,6,7,8).

מקרה טיפוסי - נתונים מעורבבים ⇐ $O(\log n)$

עצים מאוזנים - נדרוש שגם במקרה הגרוע ביותר, עלות ההכנסה וההוצאה היא $O(\log n)$

הערה - ערימה אינה עץ מאוזן מאוד - רק מציירים אותה ככה.

עצי 2-3:

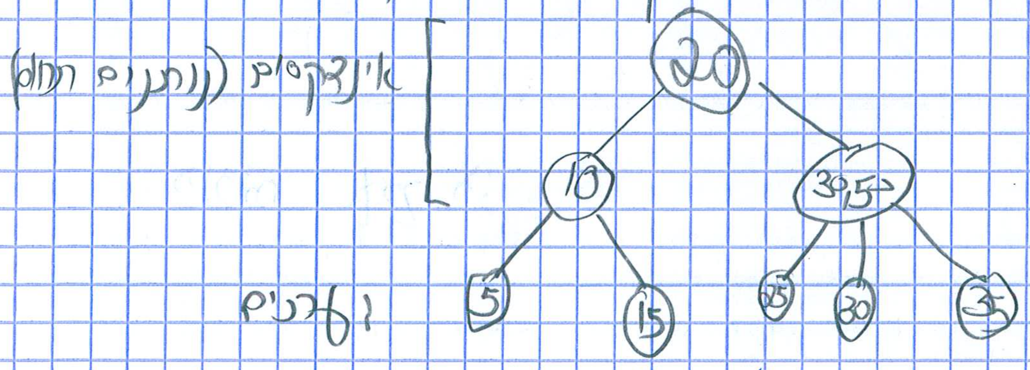
א) כל הערכים בעלים (אין לפנימיים value),

ב) כל העלים באותו עומק (מרחק מן השורש).

ג) לכל קודקוד פנימי יש 2 או 3 ילדים

3. בכל קודקוד פנימי יש אינדקסים שמפרידים בין הערכים בתת העץ הימני האמצעי, והשמאלי. לקודקוד עם שני ילדים יהיה אינדקס אחד, ולקודקוד עם שלושה ילדים יהיו שני אינדקסים.

ב. כאשר יש רק ערך אחד בעץ ייתכן שלשורש בן בודד.



חיפוש בעץ 2-3 (מחפשים את x):

א. אם השורש ריק - אין את x.

ב. אם הגעתי לקודקוד פנימי וx קטן מהאינדקס השמאלי - חפש בתת עץ שמאלי.

אם x גדול או שווה מהאינדקס השמאלי וקטן מהאינדקס הימני, חפש בתת עץ אמצעי.

אחרת, חפש בתת עץ שמאלי.

ג. אם הגעתי לעלה, וערכו x - מצאתי, אחרת x לא קיים.

הוספה של x לעץ 2-3:

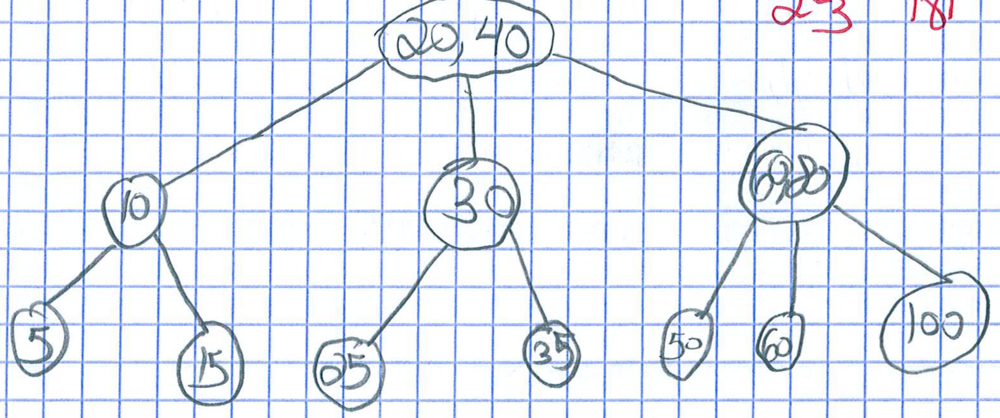
א. אם x קיים, סיימתי.

ב. אם x לא קיים, הוסף אותו לעלה שיוצא מהקודקוד הפנימי האחרון שאליו הגעתי, והוסף אותו בתור אינדקס, אלא אם כן הוא נכנס לערך השמאלי ביותר, ואז הכנס אותו לשם והכנס הערך הקודם הנוכחי בתור אינדקס.

ג. אם העץ ריק - x יהיה שורש העץ ואין שום אינדקס.

ד. במקרה שהגעתי לקודקוד פנימי עם שלושה אינדקסים, נפצל אותו ונעלה את האינדקס האמצעי לאבא. נחזור על התהליך עד שנגיע לקודקוד פנימי עם אינדקס בודד או שורש. אם שורש, נפצל אותו כך שהאינדקס האמצעי יהיה שורש חדש.

דוגמה לעץ 2-3

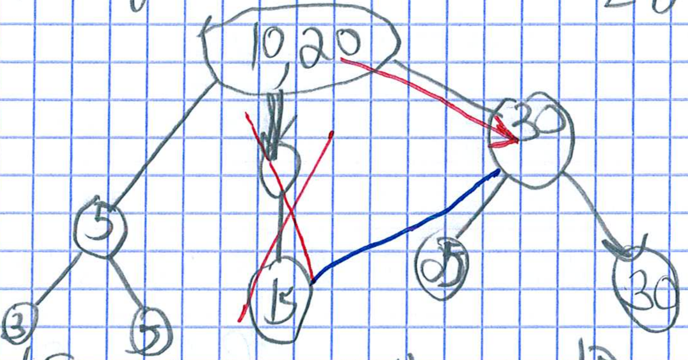


הורדת ערך x:

א. נחפש את x, נניח x לא קיים - סימני.

ב. אם x קיים - מחק את x ואת האינדקס לשמאלו באביו, אלא אם כן x הכי שמאלי ואז מחק את הערך לימינו

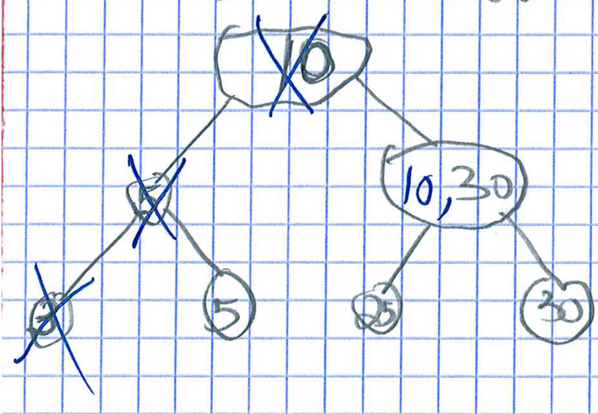
ג. אם בקודקוד הפנימי שמעל x נשאר אינדקס - סימני. אחרת חבר את אחיו של x לקודקוד שמימינו והוסף לקודקוד מימינו את הערך שהיה ברמה מעליהם, ביניהם.



אם אביו של x הוא הכי ימני חבר אותו לאחיו משמאלו.

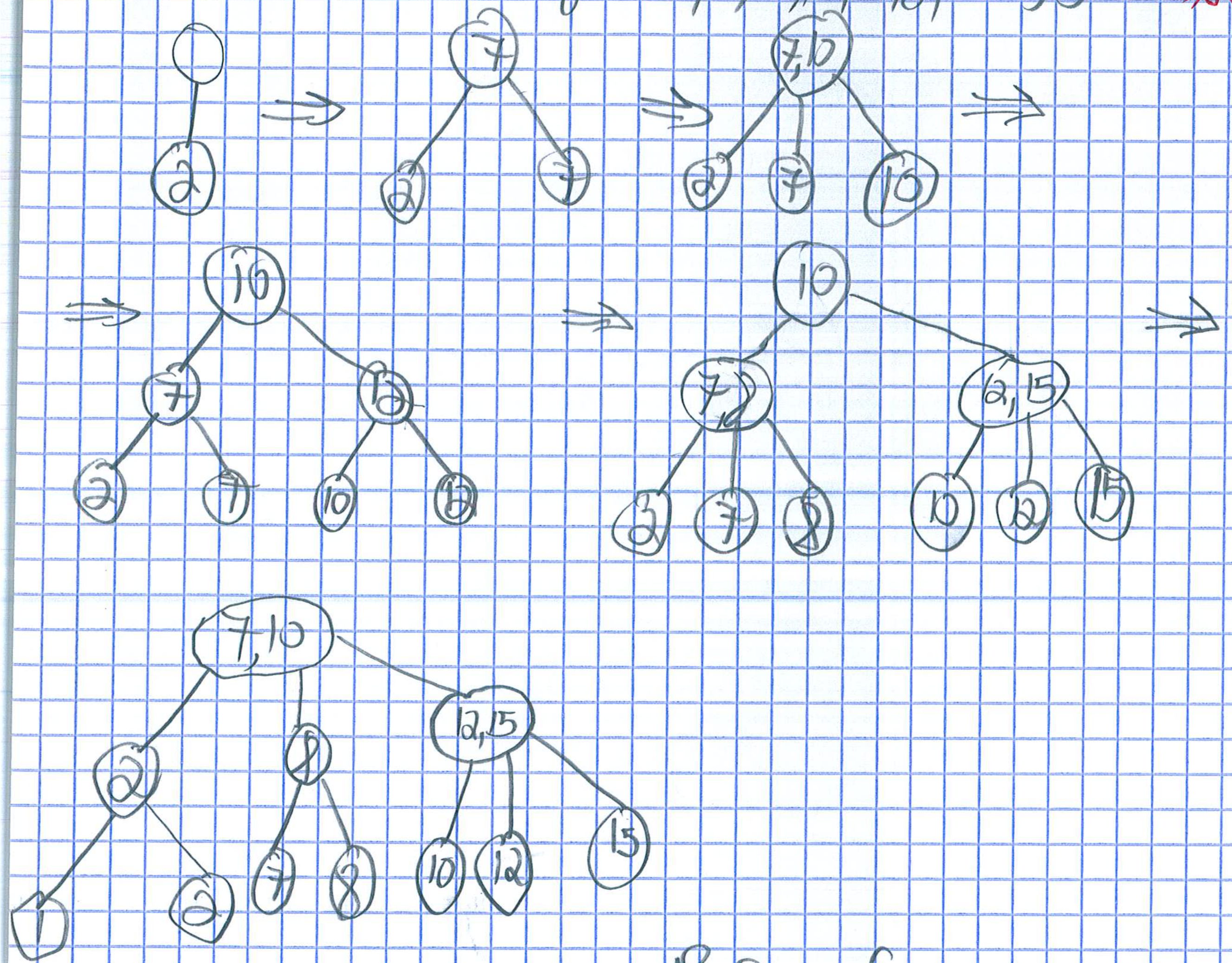
אם כתוצאה מן החיבור נוצר קודקוד פנימי עם 3 אינדקסים, פצל כנ"ל.

אם כתוצאה מן החיבור לקודקוד הפנימי שמעליהם לא נשאר אינדקס, חזור על התהליך

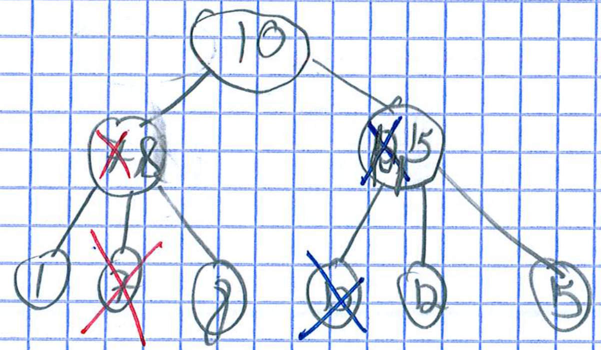


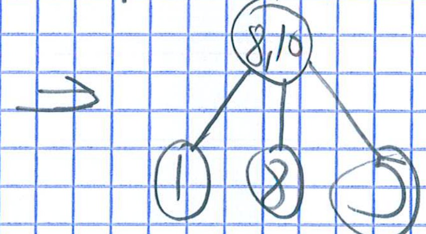
שעמה

נכניס 2,7,10,12,15,8,1 לעץ



כעת, נוציא מההתחלה לסוף





והשאר טריוויאלי.

⑥ (11/11/12)

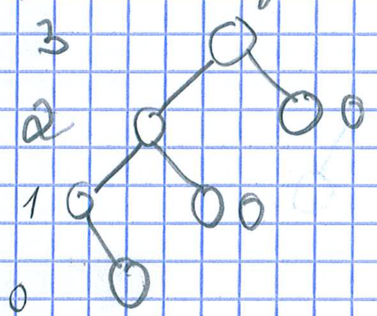
עצי חיפוש

בעצים לא מאוזנים, בממוצע הסיבוכיות הן $O(\log n)$, אבל אם הנתונים מסודרים אז $O(n)$, לכן יש גם צורך בעצים מאוזנים, למשל עצי 2-3 בהם הנתונים הם רק בעלים, כולם באותו עומק והסיבוכיות תהיה $O(\log n)$.

עצי AVL

לא מרשים חוסר איזון של יותר מאחד.

נגדיר **גובה** של קודקוד בתור עומק מירבי של קודקוד בתת העץ שמתחתיו יחסית אליו.

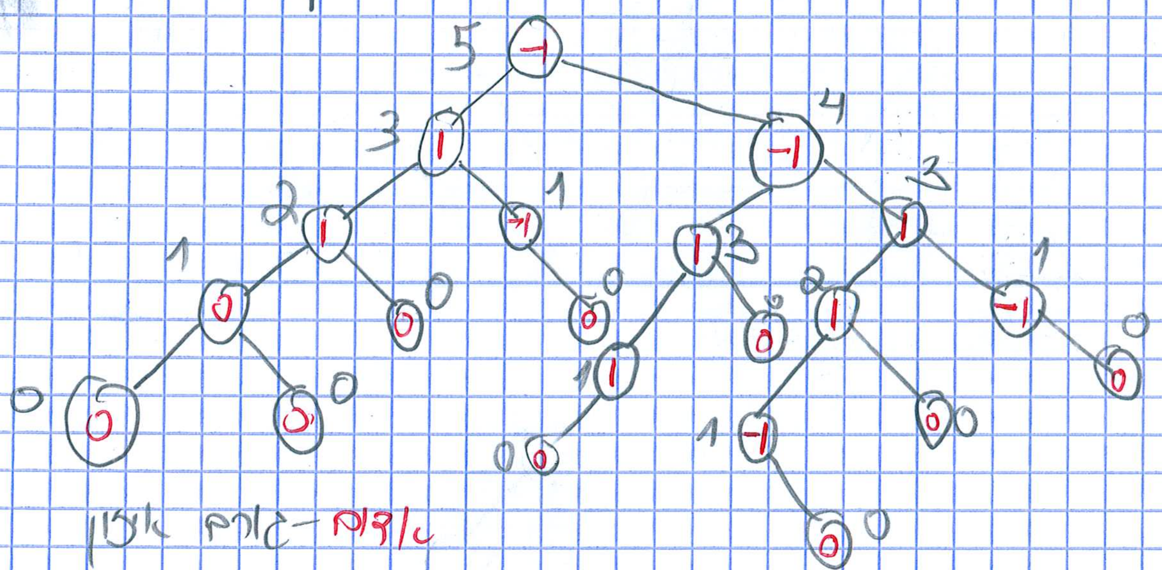


עץ בינ' עץ AVL הוא עץ בו כל קודקוד מקיים:

$$|h_L - h_R| \le 1$$

כאשר h_L הוא גובה תת העץ השמאלי ו-h_R גובה תת העץ הימני.

למשל:



אדום - גורם איזון

שחור - גובה

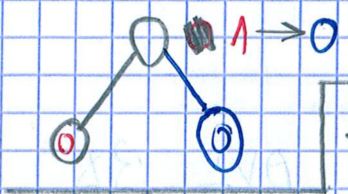
הערה

בהגדרה שלנו גם נדרוש שאם אין תת עץ ימני, תת העץ השמאלי הינו עלה.

⊕ בעץ הנ"ל כל קודקוד מכיל גורם איזון שהוא $h_L - h_R = I$ בנוסף לנתונים (מופיע באדום בדוגמה)

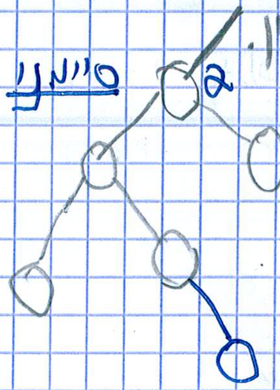
ההכנסה או ההוצאה, צריך לעדכן את גורם האיזון. המידע מההכנסה נתונים צריך לעדכן את המקום מעל ההכנסה לפי האיזון בו היא נעשתה (ימין: -1, שמאל: 1)

הכנסה (של x): נחלק למקרים:



1) אם איזנו קודקוד - סיימנו

2) אם הפכנו קודקוד מאוזן ל ±1:
צריך להמשיך לעדכן במעלה המסלול בתוספת +1 אם פנינו משמאל ו -1 אם פנינו מימין.



נסיים אם:

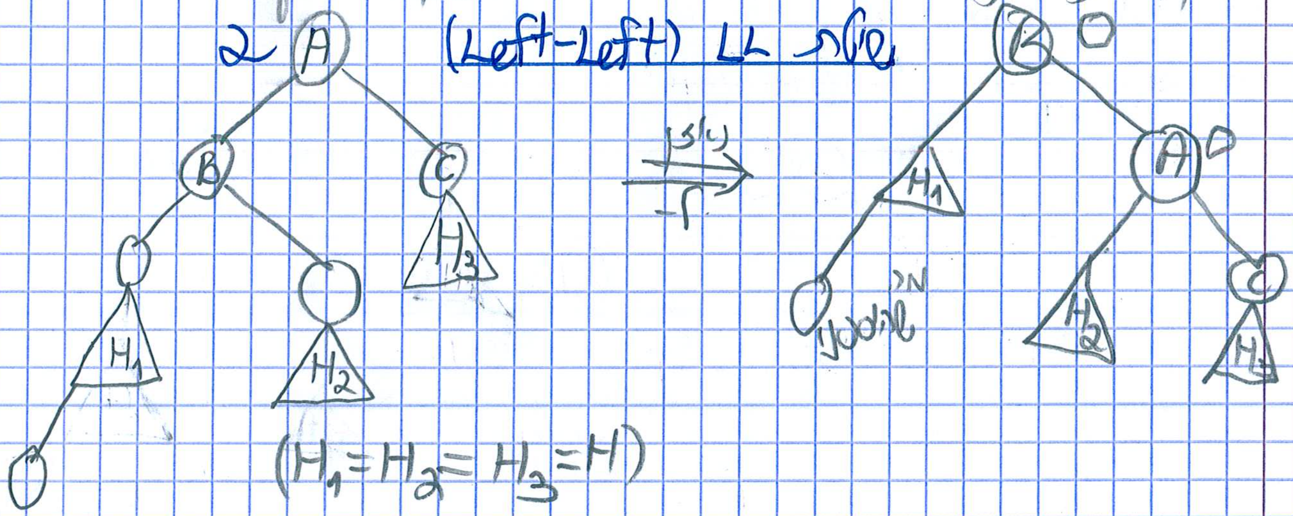
א. איזנו תת עץ.

ב. הגענו לשורש.

ג. חרגנו מחוקי AVL

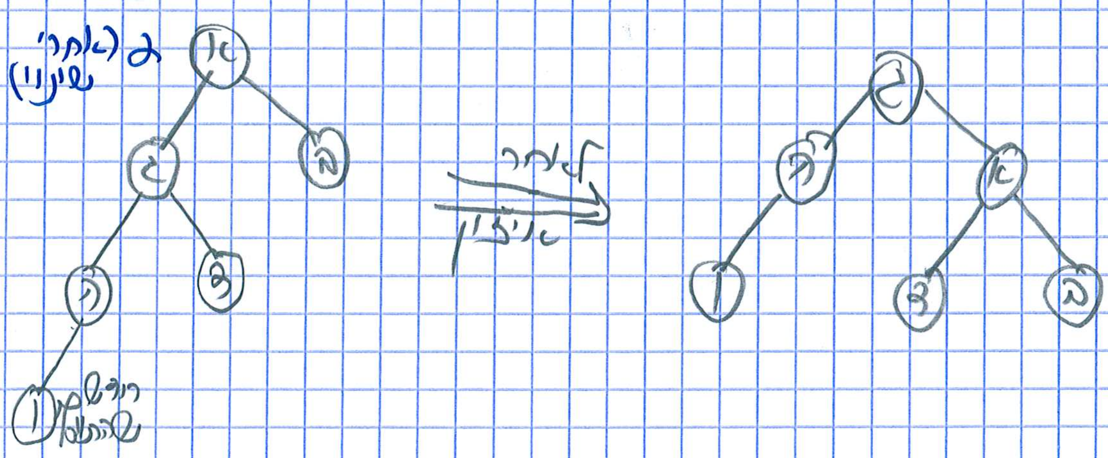
ומכאן, נתחיל בפעולות לאיזון העץ:

שיטת LL (Left-Left)



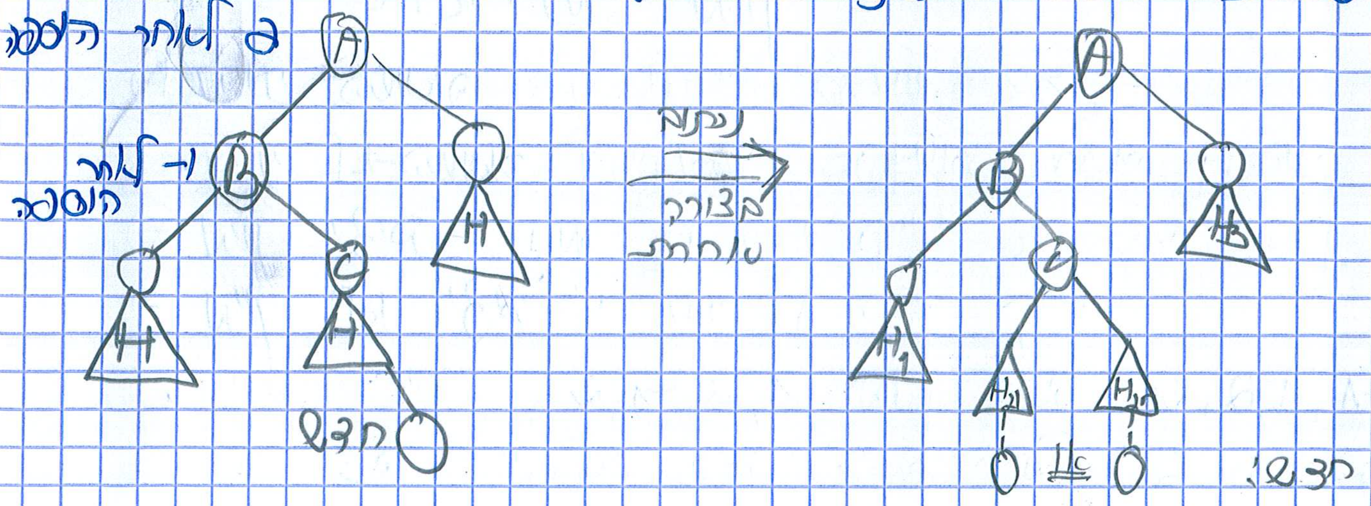
$(H_1=H_2=H_3=H)$

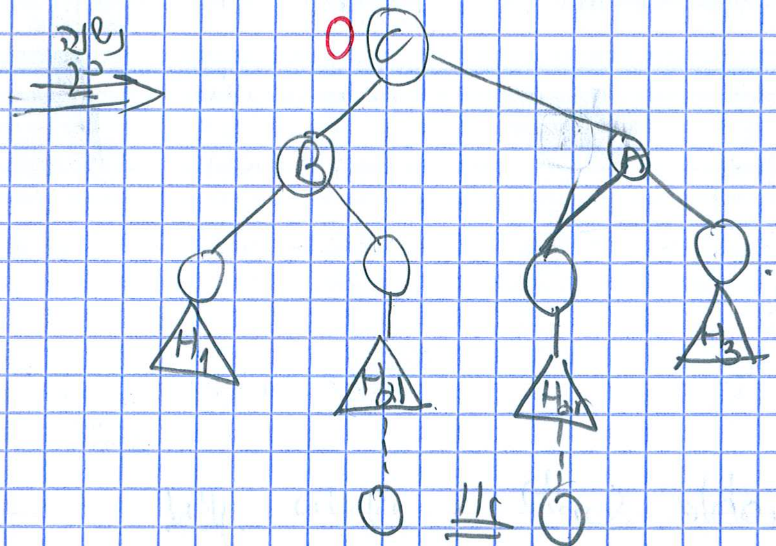
דוגמה



* באותו אופן עבור שיטת RR (Right-Right)

שיטת LR (Left Right)





הגובה של A ו B
(אורך מהם) יהיה ד,
ישנו השני 0, וכך יתאזן.

(א) הוצאה תיעשה באותו אופן.

אלגוריתמי מיון

* מיוני השוואה - משתמשים בפעולות $<, >, =$
מיוני לא-השוואה - משתמשים בפעולות נוספות על אלו.

* מיון יציב - שומר על סדר קיים אם כבר ממוין.
מיון לא יציב - לא שומר על סדר ק"מ

דוגמא: אם 2, 1-, 2, 1 ימוין לכדי 1-, 1, 2, 2
אז זהו מיון לא יציב (כי הסדר של הב' התהפך).

אז, למשל, המיון שמירת לפי שמות משפחה:

אבי כהן	⟸	רפאל ישראל
דוד כהן		דוד כהן
יעקב מנשה		אבי כהן
משה לוי		משה לוי
רפאל ישראל		יעקב מנשה

לא יציב

מיון בועות (Bubble Sort)

כל עוד המערך לא ממוין, עבור על כל האיברים.
אם $a[i]>a[i+1]$ החלף ביניהם.
עלות - $O(n^2)$, כי אם המערך מבולגן לגמרי, נצטרך לעבור עליו n פעמים ובכל פעם לבצע n החלפות.

* למשל, עבור המערך 20, 1-, 0, 7, 10, 15, 18, 19 המיון יעשה פעולות של $O(n)$.
אבל עבור המערך 3-, 0, 7, 10, 10, 15, 18, 19, 5- המיון יעשה פעולות של $O(n^2)$

מיון קוקטייל (Cocktail Sort)

כמו מיון בועות, אבל כל מעבר בכיוון אחר
פעם משמאל לימין ופעם מימין לשמאל.

מיון מהיר (Quick Sort)

א. אם גודל המערך הוא אחד, סיימנו.

ב. אחרת, בחר Pivot (ערך כלשהו במערך).

ג. סדר מחדש את המערך כך שכל הערכים הקטנים מה-Pivot יהיו משמאלו, והשאר מימינו.

ד. אם גודל תת המערך הימני גדול מאחד, הפעל Quick Sort.

אם נתבונן בנוסחת הנסיגה

$$T(n)=\frac{1}{n}[\underbrace{T(1)+T(n-1)}_{\text{ראשון Pivot}}+\underbrace{T(2)+T(n-2)}_{\text{שני Pivot}}+\dots]+n=\frac{2}{n}[\sum_{i=1}^{n}T(i)]+n$$

הסיבוכיות היא $O(n\ln n)$.

גם מהתבוננות על המקרה הפרטי $T(n)=2T(\frac{n}{2})+n$

ואז הסיבוכיות $O(n\log_2 n)$.

הערה: אם המערך ממוין, העלות תהיה $O(n^2)$.

Heap Sort $O(n\log n)$

א. הכנס את כל הנתונים לערימה

ב. הוצא אותם.

7 13/11/12

מיונים

למיון השוואה יש יכולת מוגבלת: נניח שהקלט הינו k פעולות,
ואז אפשר להבחין ב-2^k אפשרויות (גדול/קטן או לא) לכל היותר.
כעת, אם יש n איברים, יש להם $n!$ תמורות ומתוכן תמורה
אחת הינה הנכונה. לכן, במקרה הטוב ביותר:

$n! = 2^k$ (יש 2^k אופציות למיון ו-$n!$ אפשרויות למיון)

$$\Rightarrow k = \log_2(n!) = c \cdot \ln(n!)$$

ידוע (סטירלינג): $n! \approx \left(\frac{n}{e}\right)^n$, ולכן:

$$\ln(n!) = \ln\left(\left(\frac{n}{e}\right)^n\right) + \frac{1}{2}\ln(2\pi n) = n\ln n - n\ln e + O(\ln n) = \Theta(n\ln n)$$

מכאן לא יתכן מיון השוואתי שיעבוד בסדר גודל כשהו יותר מ-$\Theta(n\ln n)$

* Bubble Sort - לרוב עובד ב-$O(n^2)$ אבל אם ממוין כבר $O(n)$.
* Quick Sort - לרוב עובד ב-$O(n\log n)$ אבל אם ממוין אז $O(n^2)$.
* Heap Sort - הכנס לערימה, והוצא ממנה הכל ב-$O(n\log n)$.

עם זאת, כשמבני נתונים גדולים עדיף למיין לפי כתובות
(כשמבנים לא להעתיק מבנים גדולים) אבל זה לא אפשרי
במיון ערימה. מה שכן ממיינים לפי כתובת אפשר
למצוא פרמוטציה.

Merge Sort $O(n\log n)$

ממיינים כל חצי של המערך בנפרד, ואז ממזגים:

* אם ממזגים רשימות A,B: $k=0; i=0; j=0$

לולאה: אם $A[i] \le B[j]$ ו- $i<|A|$: $C[k]=A[i]; k++; i++$
אחרת אם $B[j] < A[i]$ ו- $j<|B|$: $C[k]=B[j]; j++; k++;$

זה עולה של $O(n)$.

כיון שאנחנו עושים $\log n$ פיצולים, בכל פיצול העלות הינה $O(n)$.
לכן העלות הכוללת הינה $O(n\log n)$.

דוגמה

$7 \mid -3 \mid 5 \mid 2 \mid 9 \mid -7 \mid -2 \mid 3$

$-3,7 \mid 2,5 \mid -7,9 \mid -2,3$

$-3,2,5,7 \mid -7,-2,3,9$

$-7,-3,-2,2,3,5,7,9$

Insertion Sort & Library Sort

מיון הכנסה:

עבור i שבין 1 עד לגודל המערך (כולל):
הכנס את האיבר שבאינדקס i למקומו במערך הממוין עד עתה.

למשל, עבור אותה דוגמה מקודם

$7 \Rightarrow -3,7 \Rightarrow -3,5,7 \Rightarrow -3,2,5,7,9 \Rightarrow -7,-3,2,5,7,9$

וכו'.

במקרה הגרוע ביותר, זה עלות של $O(n^2)$
אבל אפשר להשתמש במערך אחד (ע"י החלפות) ולכן אין עלות זכרון.

בנוסף, בהנחה שיש לי L איברים לא במקום, העלות היא $O(L \cdot n)$. ואז אם $L < \log n$, אז יש לנו אלגוריתם יותר טוב מ-$n\log n$. אבל אין זה קריטי, כי בפועל שהנחנו שיש רק L איברים לא במקום אנו מגבילים עצמנו מבחינת כמות פרמוטציות.

מיון ספריה: נניח שעבור i x איברים המערך יש לנו y מקומות פנויים (את זה אנחנו משיגים ע"י מעבר אחד על המערך). על החלק המלא הראשון, מבצעים מיון הכנסה כרגיל. על החלקים הבאים, נבצע מיון הכנסה על שניים - אם עוברים אחרי איבר בסופו, בחינת שאר האיברים קפיצה היא קלה יותר - כי יש מקומות פנויים שאין צורך להזיז.

Bucket Sort

נחלק לדליים (בקטים), ובכל אחד נעביר המיון שלנו לפי המיקום!
למשל, עבור המערך מהדוגמה הקודמת:

−7	−3, −2	5, 2, 3	7, 9
−10 … −6	−5 … −1	0 … 5	6 … 10

ונקבל רשימה חדשה −7, −3, −2, 5, 2, 3, 7, 9
ואז אפשר לבצע מיון הכנסה (במקרה כזה) יעבוד טוב
כי הוא קרוב למקום.

−7, −3, −2, 5
2, 5
3, 5, 7, 9

ואם נקח מס' רב של דליים אפשר להתקרב ל- לינארי.

Counting Sort

אם יש טווח חסום, סופרים את כמות האיברים בכל
ערך אפשרי וממיינים לפי זה, למשל:

1, 2, 3, 3, 1, 2, 1, 2, 1, 3, 2, 1, 1, 3

1 − 5
2 − 4
3 − 4

$\Longrightarrow$ 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3

L.S.D Radix Sort

נמיין ב- Bucket Sort את הספרה הבינארית האחרונה,
בחזרה על התהליך O עד לא הגענו לספרה הראשונה.

דוגמה

7, −3, 5, 2, 6, −7, −2, 3

$\Rightarrow$ 1111, 0101, 1101, 1010, 1110, 0001, 0110, 1011

(נקבע ש- 0 כ- 0000, 7 כ- 1111).

מתחילים למיין מהספרה הימנית, נקבל:

1010, 1110, 0110, 1111, 0101, 1101, 0001, 1011

נחזור, נמיין לפי הספרה השנייה, ונקבל:

0101, 1101, 0001, 1010, 1110, 0110, 1111, 1011

כעת, עבור הספרה השלישית נקבל

0001, 1010, 1011, 0101, 1101, 1110, 0110, 1111

עבור הספרה האחרונה, נקבל:

0001, 0101, 0110, 1010, 1011, 1101, 1110, 1111

(זהו עלות של $O(n)$ עד כדי הכפלה בכמות הביטים).
הוא גם יציב כי כל הפעולות הן ביטיות ויציבות.

M.S.D Radix Sort

נמיין לפי הספרה הראשונה, נחלק לקב', ונבצע בכל
אחת מהקב' (כאלו שמתחילים ב-0 וכאלו שמתחילים
ב-1) מבצעים שוב פעם. בניגוד לקודם,
זהו מיון רקורסיבי (תתכן שאלה במבחן).

דוגמה

1111	0101	0001		
0101	0001	0101	0101	
1101	0110	0110	0110	
1010	1111	1010	1010	1010
1110	1101	1011	1011	1011
0001	1010	1111	1101	
0110	1110	1101	1111	1110
1011	1011	1110	1110	1111

* נשים לב שהאלגוריתם ייתן לנו הסדר לקסיקוגרפי. אם
נרצה סדר מספרי, נצטרך להוסיף אפסים מובילים
כך שכל המס' יהיו עם אותה כמות ביטים.
לחלופין, גם L.S.D ייתן סדר מספרי, כאשר נתייחס
למצב בו אין ספרה כמצב שייכנס לפני 0:

11
100
101