אוניברסיטת חיפה
החוג למדעי המחשב

סיכום חומר הלימוד בקורס

מבני נתונים

מוגש מטעם נציג החוג באגודת הסטודנטים
עילאי הנדין

בכל שאלה, בעיה או הצעה ניתן ליצור קשר עם
עילאי בטלפון 054-4448698 או בדוא"ל
ilai@netvision.net.il

לחומרים נוספים גלשו לאתר הסיכומים:
cs.haifa.ac.il/students

נשמח לקבל סיכומים, מבחנים וכל חומר לימודי אחר לצורך הפצתו באתר

הבהרות והערות

קובץ סיכומים זה נכתב לפי חומר ההרצאות של פרופ' גדי לנדאו וחומר התרגולים של גב' כרמל קנט בסמסטר א' תשס"ד (לפי המתכונת החדשה של הקורס). למרות שהסיכומים נבדקו ונערכו, יתכנו טעויות, אי דיוקים ושינויים בחומר הלימוד בשנה הנוכחית. החוברת לא נכתבה ע"י סגל החוג ולכן אנו ממליצים להגיע לשיעורים ולא להסתמך עליה כחומר הבלעדי לבחינה. אנו מאחלים לכולכם שנת לימודים מוצלחת. נשמח לקבל הערות לדוא"ל ilai@netvision.net.il.

כתיבה ועריכה: עילאי הנדין.

מקרא צבעים:

A צומת שחור

B צומת אדום

X צומת שחור או אדום (לא רלוונטי)

סיבוכיות

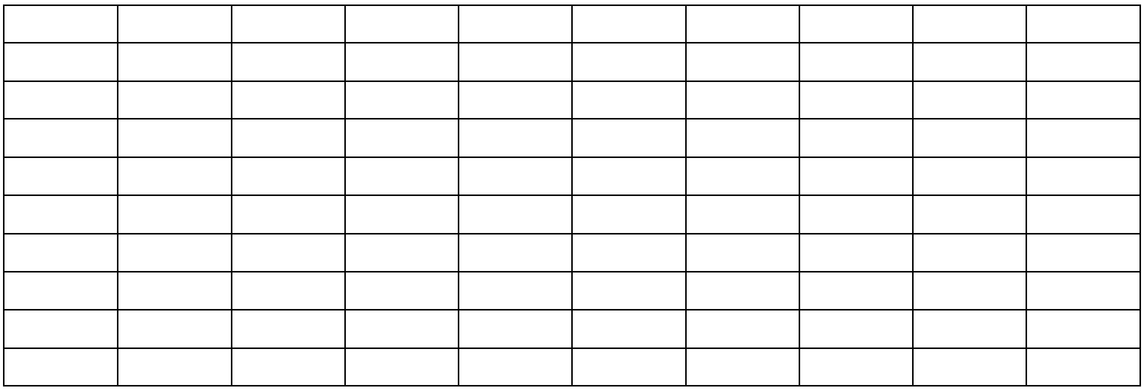
חסם תחתון – מספר הפעולות המינימלי לפתרון אותה בעיה.
נשאף להוריד את מספר הפעולות כדי להגיע לפתרון וגם למצוא את החסם התחתון העליון ביותר.

פתרון

חסם תחתון

אנו מניחים שאם בשאלה לא נאמר שמשהו מתקיים, אז אין להניח שהוא מתקיים.

לדוגמא: מציאת מקסימום במערך דו מימדי nXn



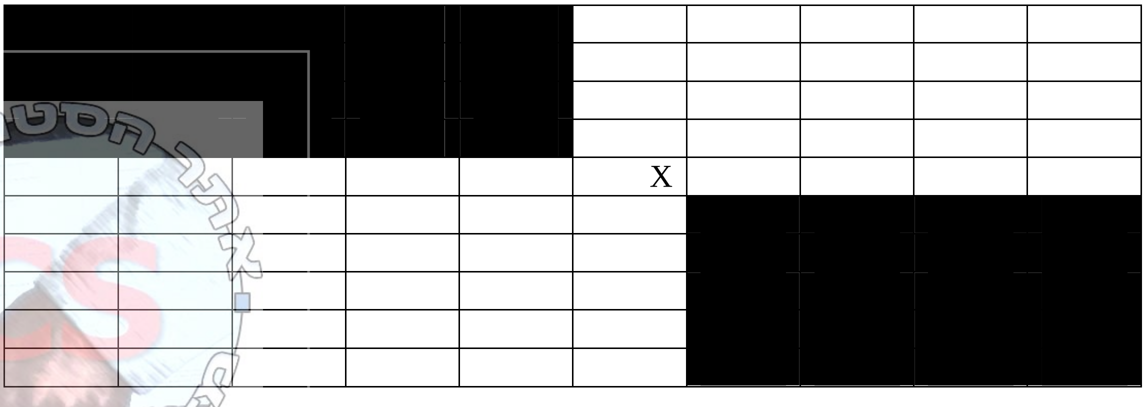
הסיבוכיות היא n^2

נוסיף הנחות:

האינדקס של המקסימום בשורה ה i = M(i)
$\forall i > 1 \quad M(i) \geq M(i-1)$ (האינדקס של המקסימום גדול בכל שורה מהשורה הקודמת)

פתרון:
להתחיל מהשורה האמצעית.

שורה $\frac{n}{2}$ עבודה: n



כל פעם חותכים את רבע הטבלה שנשארת

עבודה	שורה	צעד
n	$\frac{n}{2}$	1
n	$\frac{n}{4}, \frac{3n}{4}$	2
n	$\frac{n}{8}, \frac{3n}{8}, \frac{5n}{8}, \frac{7n}{8}$	3
n	כל האי זוגיים	y

מספר שורות	שורות	צעד
1	$\frac{n}{2}$	1
2	$\frac{n}{4}, \frac{3n}{4}$	2
4		3
8		4
2^{i-1}		i
$\frac{n}{2}$		y

$$2^{y-1} = \frac{n}{2}$$

$$2^{y} = n$$

$$y = \log n$$

ולכן מספר הפעולות הוא $n \log n$

<u>לקוח – בעיה</u>

אנחנו

א. הבנת הבעיה, הגדרה "מתמטית". לאחר מכן חזרה ללקוח ותיאום של הגדרת הבעיה איתו.
ב. פתרון כללי, מתמטי, תוכנה, חומרה.
ג. ישום.
ד. בדיקות, תיעוד.

עלות

- זמן פיתוח
- חומרה
- תחזוקה

יעילות

- זמן
- מקום

כשמבקשים פתרון לבעיה, מבקשים מאיתנו את הפתרון היעיל ביותר. אם יש שני פתרונות – אחד יעיל בזמן ואחד יעיל במקום, יש להציג את שניהם.

$T(n)$ - מספר הפעולות של אלגוריתם מסויים בהינתן הפרמטר n.

פעולה - חיבור, חיסור, כפל, חילוק, השוואה, פעולה לוגית כלשהי, כתיבה וכו'.

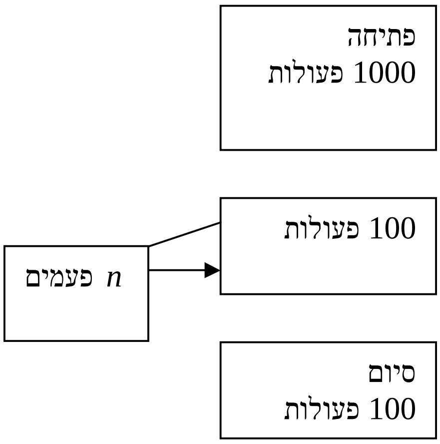
מילת מחשב – לא יותר מ $\log n$ ביטים
n - גודל הקלט

סדר גודל - אלגוריתם אחד יותר טוב מהשני רק אם הוא בסדר גודל שונה.

$O(n^2)$ - תיאור סיבוכיות הזמן במקרה הכי גרוע.
$O(g(n))$ - חסם עליון לזמן הריצה במקרה הגרוע ביותר.
$O(g(n))$ = { קבוצת פונקציות $f(n)$ כך שקיימים c, n_0 $0 \le f(n) \le c \cdot g(n)$ עבור כל $n_0 < n$ }
c הוא קבוע, לא קשור ל n.
n_0 הוא חסם תחתון – מספר הפעולות שאחריהן מתחילים לחשב.

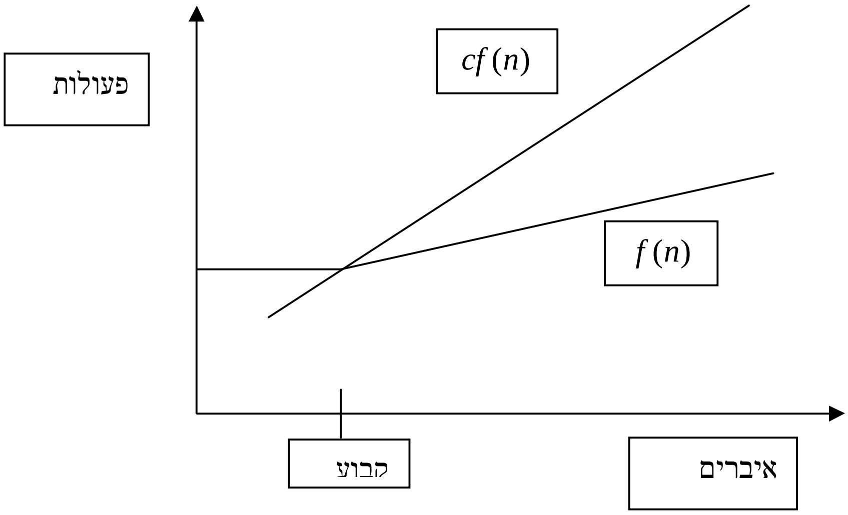
$T(n) = O(f(n))$ אם $T(n) \le cf(n)$ עבור כל $n > n_0$

דוגמא לתוכנית:



סדר גודל הזמן הוא n.

$f(n) = n$
$n_0 = 11$
$c = 200$
$T(11) = 2{,}200$
$T(100) = 11{,}100$



$T(n)=q(f(n))$ אם $c_2 f(n) \le T(n) \le c_1 f(n)$ עבור כל $n > n_0$

דוגמא נוספת:

i= 1 to n

j=1 to i

$$\sum_{i=1}^{n} ix = x\sum_{i=1}^{n} i = \frac{xn(n+1)}{2} = x(\frac{n^2}{2}+\frac{n}{2}) = O(n^2)$$

<u>מציאת מקסימום ב $O(1)$ (באמצעות מספר בלתי מוגבל של מחשבים)</u>

A מערך המספרים

		25				20			

B מערך עזר

	זמן	פעולות	מחשבים	
שלב א' – סופר את B	$O(1)$	n	n	
שלב ב' – דו קרב (השוואה בין כל הזוגות האפשריים).	$O(1)$	$O(n^2)$	$O(n^2)$	
שלב ג' - המנצח	$O(1)$	n	n	

בשלב הדו קרב ישנה השוואה בין כל הזוגות האפשריים וכל מספר שמפסיד מסומן כמפסיד. רק המקסימום לא יסומן כמפסיד.

נוסחאות שכדאי לזכור

$$\log n = \log_2 n$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_5 n = \frac{\log_2 n}{\log_2 5} = O(\log n)$$

טור חשבוני:

$$\sum_{k=1}^{n} k = 1 + 2 + ... + n = \frac{n(n+1)}{2} = O(n^2)$$

טור גיאומטרי:

$$1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{n} = \sum_{k=0}^{\log n} \left(\frac{1}{2}\right)^k \le 2 = O(1)$$

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... + 1 = O(n)$$

סדרי גודל

$$\log n < \sqrt{n} < n < n \cdot \log n < n^2 < n^3 < 2^n < n^n$$

$$n! = O(n^n)$$

נוסחאות נסיגה

באלגוריתם רקורסיבי נבנה נוסחאות נסיגה: $T(n) = a \cdot T(b) + c$

a - מספר תתי הבעיות

b - גודל כל תת בעיה

c - העבודה בכל קריאה רקורסיבית

1. השיטה האיטרטיבית:

$$T(1)=1$$

$$T(n)=2T\left(\frac{n}{2}\right)+n=2\cdot\underbrace{2T(\frac{n}{4})+\frac{n}{2}}_{T(\frac{n}{2})}+n=4T(\frac{n}{4})+2n=2^k\cdot T(\frac{n}{2^k})+k\cdot n$$

מפסיקים כאשר $\frac{n}{2^k}=1$

$$n=2^k\rightarrow k=\log n\rightarrow T(n)=2^{\log n}\cdot 1+\log_n\cdot n$$

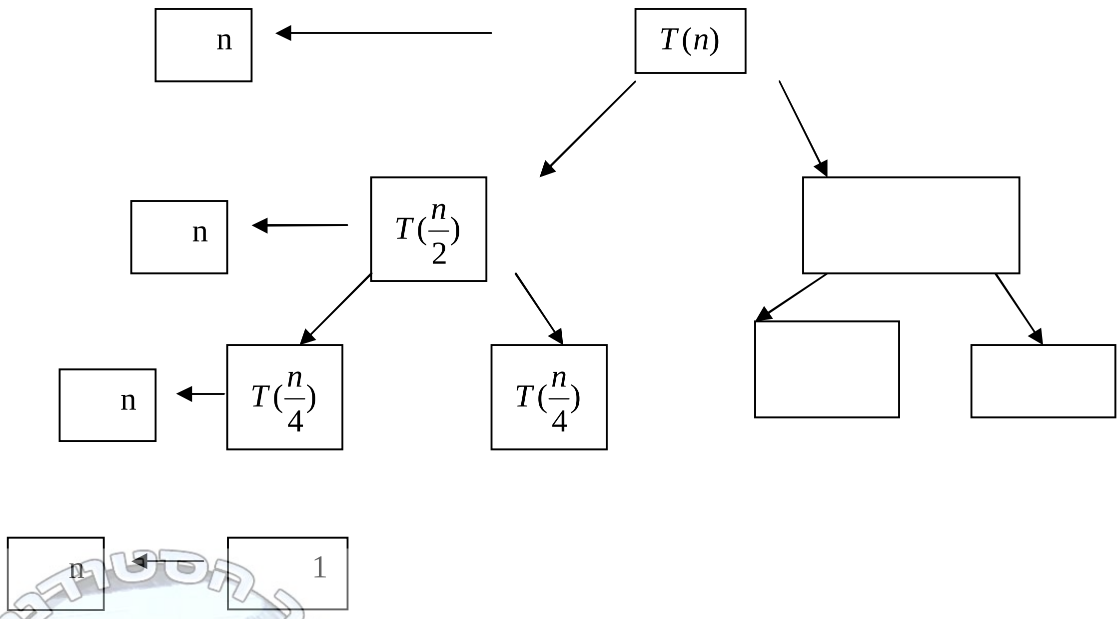
$*\ a^{\log_a b}=b$

$$T(n)=n+\log n\cdot n=O(n\cdot\log n)$$

```
for (k=n;k>1;k/=5)                    // log_5 n
      for (j=1;j<=logn;j++)         // logn
            for (m=1;m<=n/3;m++) {    // n/3=O(n)
                  a=b*b;                // O(1)
            }
```

$$T(n)=\log_5 n\cdot\log_2 n\cdot\frac{n}{3}=O(n\cdot\log^2 n)$$

2. עץ קריאות



$$T(1)=1$$

$$T(n)=2T(\frac{n}{2})+n$$

$$O(\log n\cdot n)$$

$$T(1)=1$$
$$T(n)=2T\left(\left\lfloor\sqrt{n}\right\rfloor\right)+\log n$$
$$m=\log n$$
$$T(2^m)=2T(2^{\frac{m}{2}})+m$$
$$s(m)=T(2^m)$$
$$s(\frac{m}{2})=T(2^{\frac{m}{2}})$$
$$\downarrow$$
$$s(m)=2s(\frac{m}{2})+m$$

(ואת נוסחת הנסיגה הזאת כבר ראינו)

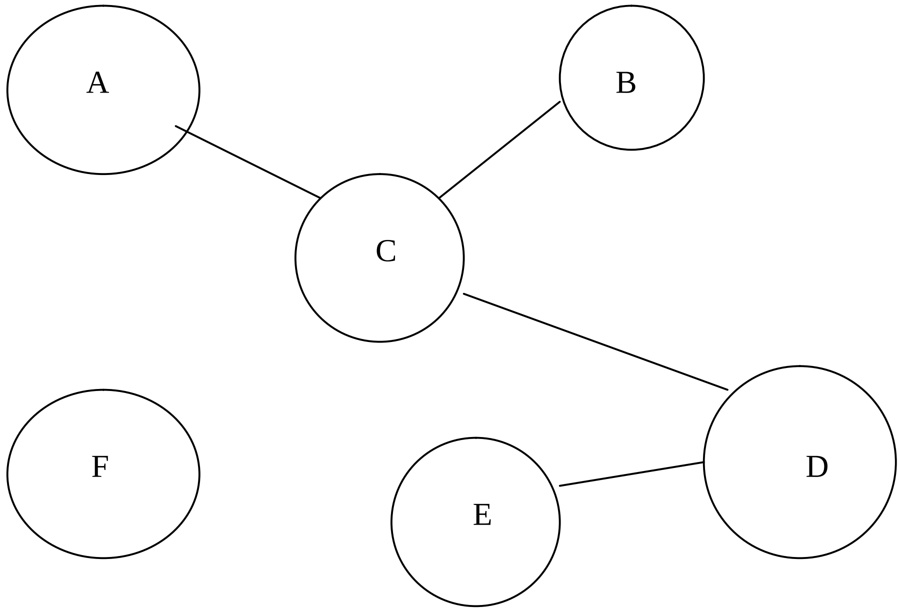
$$O(m\log m)$$
$$T(n)=T(2^m)=s(m)=O(m\log m)=O(\underbrace{\log n}_{\boxed{m}}\cdot\underbrace{\log(\log n)}_{\boxed{\log m}})$$

נתחו את הסיבוכיות של הקוד הבא:

```
for (k=2;k<=n;k*=2)            // = O(log n)
      for(i<=1;i<=n/k;i++)     // n/2 + n/4 + n/8 + ... + n/n = O(n)
            j--;
```

$$=O(n\log n)$$

גרף $G(V,E)$



תת גרף $G^{/}(V^{/},E^{/})$

$V^{/} \subset V$

$E^{/}$ - כל הקשתות שמחברות צמתים ב $V^{/}$

מסלול

$V_0, V_1, \ldots, V_k$

$V_n \rightarrow V_{n+1}$

מעגל

$V_0 = V_k$

גרף מלא – יש קשתות בין כל הצמתים

גרף דו צדדי – יש קשתות בין שני הצדדים, אך לא בינם לבין עצמם.

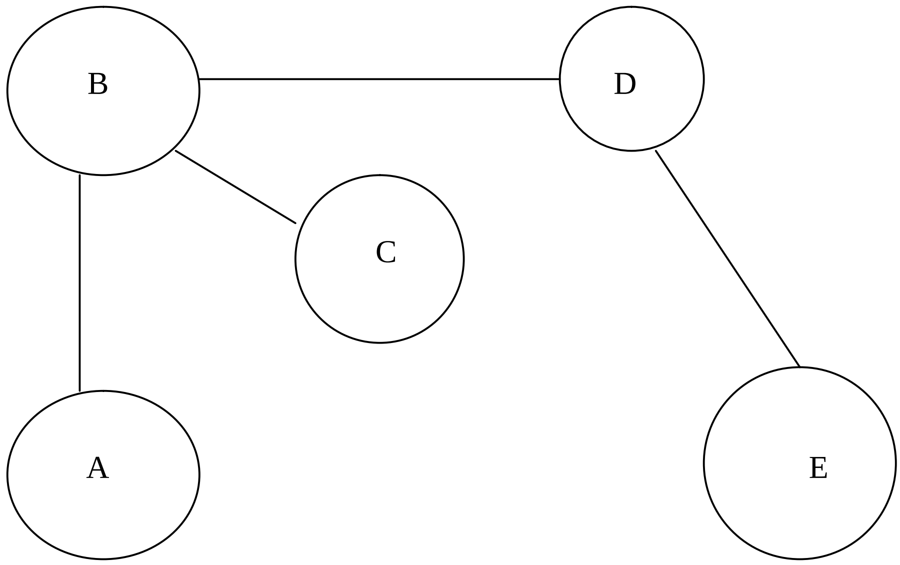
קורסים	חדרים
O	O
O	O
O	O
O	O
O	O

גרף קשיר – כל הצמתים מקושרים בצורה כלשהי זה לזה (ניתן להגיע מכל אחד לאחר במסלול כלשהו).

<u>עצים</u>

עץ חופשי

- גרף $G(V,E)$
- לא מכוון
- ללא מעגלים
- קשיר



$|V| = n$
(מספר הצמתים והקשתות)
$|E| = m$

בעץ חופשי $m = n-1$

הוכחה באינדוקציה:
$n = 1 \to m = 0$

נניח $n = k$
נוכיח $m = k+1$

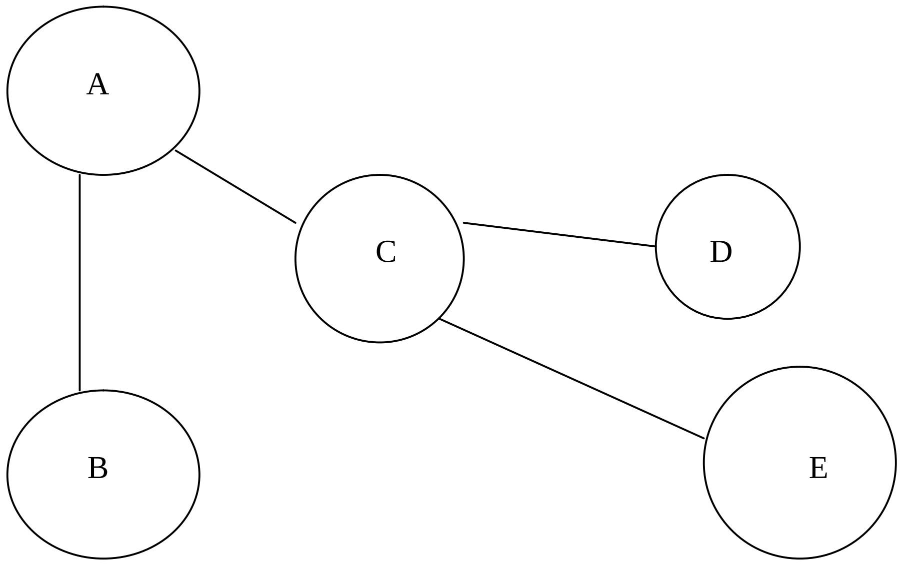
כל פעם ננתק צומת ואת הקשת שמובילה אליו, כלומר יהיו לנו צומת אחד פחות וקשת אחת פחות.
דרך אחרת היא להוכיח שאם התנאי לא מתקיים, אז הגרף הוא לא עץ.

עצים מושרשים

- שורש אחד
- צמתים פנימיים
- עלים

השורש הוא הצומת היחידי שאין לו הורה.
לצמתים פנימיים יש גם הורה וגם ילדים.
לעלים יש הורה, אך אין ילדים.

אחים – כל הצמתים שיש להם הורה.
תת עץ – צומת וכל הצאצאים שלו.



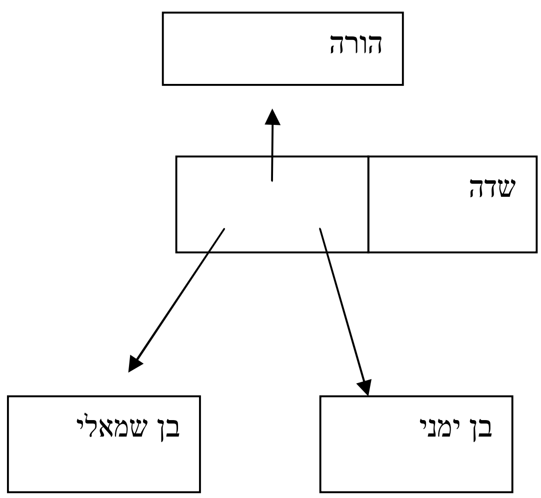
עץ מסודר – הילדים מסודרים על פי סדר מסויים (סוג מסויים תמיד משמאל וסוג מסויים תמיד מימין – גם אם יש רק ילד אחד).

עץ בינארי - אין לו יותר משני ילדים. אם לצומת יש ילד אחד בלבד, הוא יכול להיות או ימני או שמאלי.

עץ כללי – לכל צומת אין הגבלה על מספר הילדים. בעץ כללי הילדים יסודרו באופן רציף.

<u>יישום עץ בינארי</u>

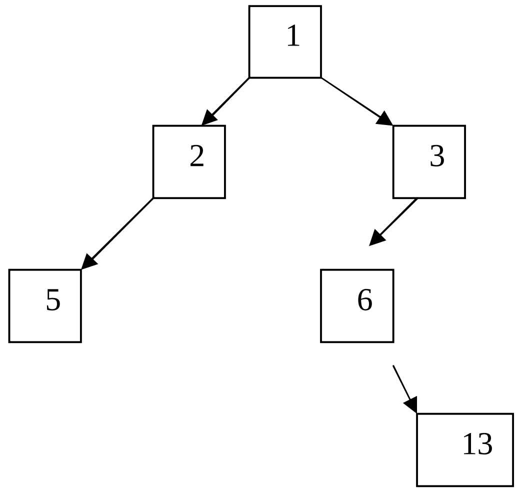
מבנה הצומת:



ייצוג עץ בינארי בשיטת המערך

כל תא בזיכרון יתפוס מקום אחד בלבד (חיסכון במקום)

ילדי הצומת i נמצאים ב $2i$ (בן שמאלי) וב $2i+1$ (בן ימני).



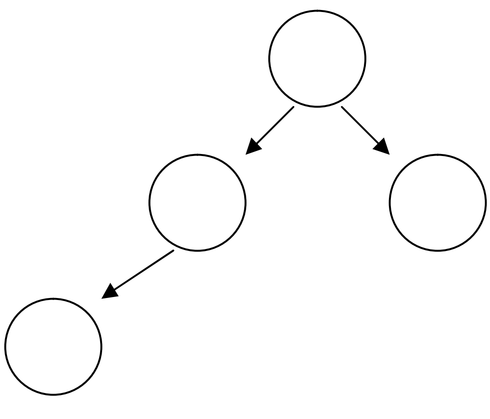
יתרון: חיסכון במקום (לא תמיד). חיסרון: המערך אינו אינסופי. לא תמיד הוא חוסך במקום (במקרה של צמתים רבים שלא קיימים).

גובה ורמה

לכל צומת ניתן לחשב את גובהו ורמתו.
גובה של צומת – אורך המסלול הארוך ביותר מצומת לעלה בתת העץ שלנו. גובה העץ הוא גובה השורש.
רמה (עומק) של צומת – אורך המסלול מהשורש לצומת.

עץ בינארי כללי - גובהו הוא $O(n)$.

עץ בינארי מלא (שלם) - כל השכבת מלאות, מלבד השכבה התחתונה והיא מלאה משמאל לימין.



מספר צמתים	גובה
1	0
2-3	1
4-7	2
$2^i - (2^{i+1} - 1)$	i
$2^h \le n < 2^{h+1}$	h

גובה עץ בינארי מלא:

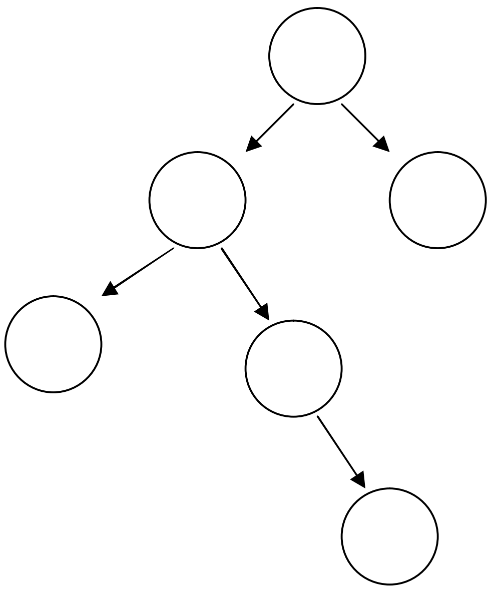
$$h \le \log n < h + 1$$

$$h = \lfloor \log n \rfloor$$

<u>עץ בינארי מאוזן</u>

$$h = O(\log n)$$

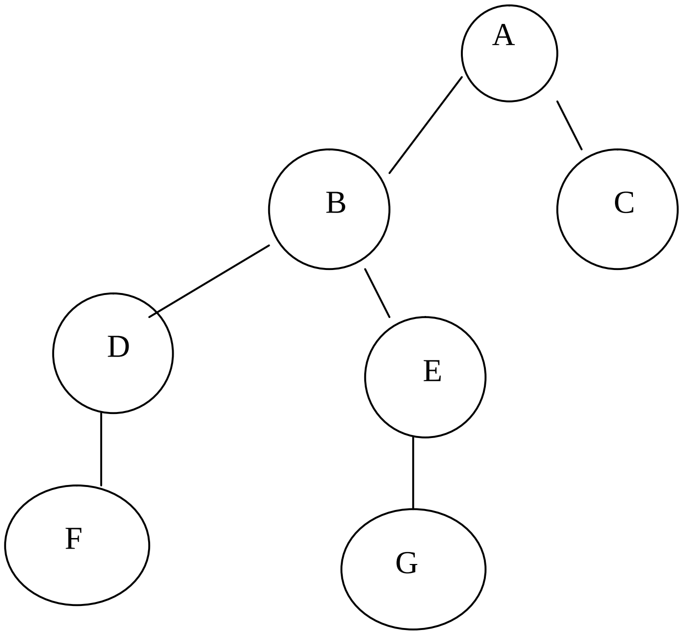
עץ מלא הוא תמיד עץ מאוזן. לעומת זאת לא כל עץ מאוזן הוא עץ מלא.
עץ מאוזן הוא עץ בינארי שגובהו $O(\log n)$. בעץ מאוזן יש פחות סדר מאשר בעץ מלא.



גובה	מס' צמתים עד לרמה זו
0	1
1	2-3
2	4-7
3	8-15
i	$2^i > 2^{i+1} - 1$

עץ בינארי מלא כדאי ליישם בעזרת מערך. אין לנו חורים באמצע. בעץ בינארי מאוזן יכולים להיות חורים באמצע.

מימוש עץ בינארי באמצעות מערך



עבור צומת i:

הבן השמאלי נמצא באינדקס $2i$.

הבן הימני נמצא באינדקס $2i+1$.

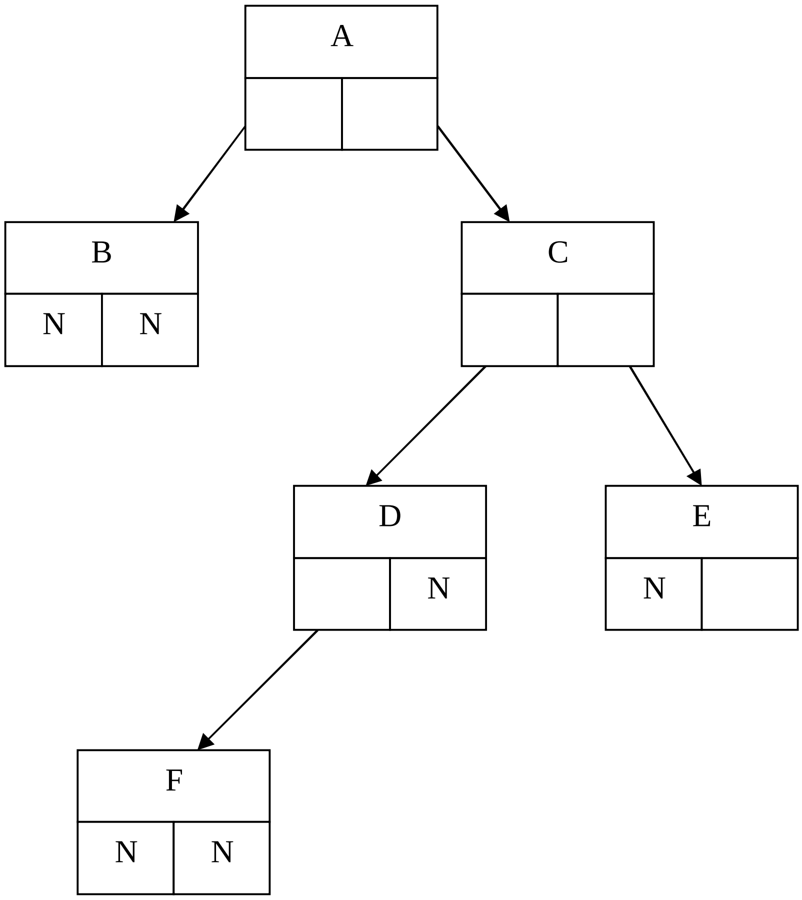
A	C	C	D	E	-	-	F	-	-	G	-
1	2	3	4	5	6	7	8	9	10	11	12

היתרון הוא שאפשר להגיע ב $O(1)$ לאבות קדמונים.

החיסרון הוא שעץ לא מאוזן מבזבז הרבה מקום.

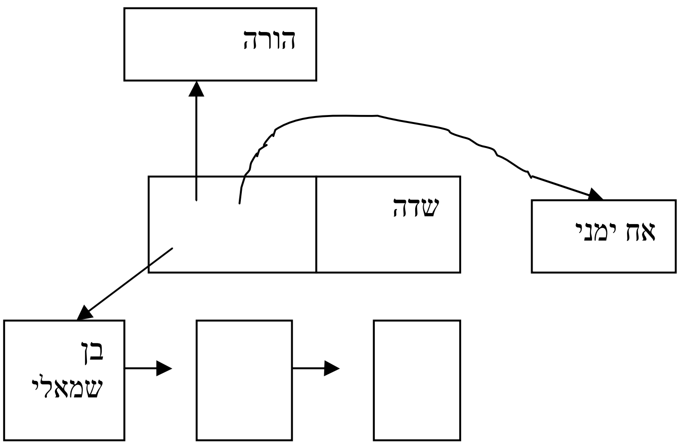
מימוש עץ בינארי באמצעות מצביעים

ערך	מצביע לבן השמאלי	מצביע לבן הימני

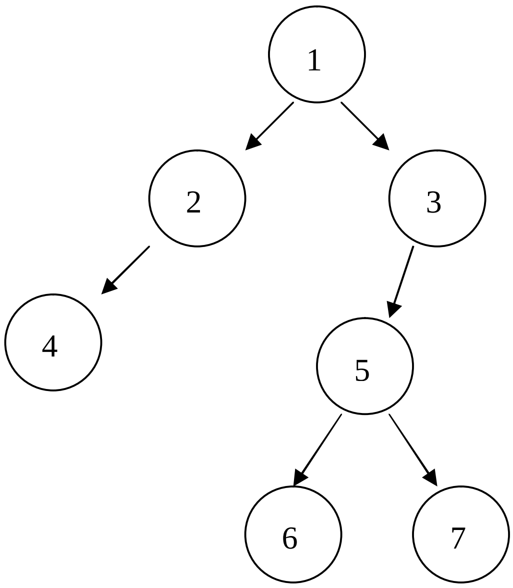


<u>מימוש עץ כללי באמצעות מצביעים</u>

לכל צומת יש שדה. השדה מכיל מצביע להורה, מצביע יחיד לבן השמאלי ומצביע לאח הימני.



<u>מעבר על צמתים בעץ</u>



<u>המטרה</u>: חשב לכל צומת את
א. גובה הצומת
ב. עומק הצומת

<u>שיטה א'</u>: נטפל בכל צומת בנפרד.
חישוב הרמה לכל הצמתים $O(n^2)$
חישוב הגובה ?
ניתן לראות שהחישוב הוא ארוך ומסובך.

<u>שיטה ב'</u>: טיפול בכולם בו זמנית.
הגובה של צומת הוא הגובה המקסימלי של ילדי הצומת (הגובה של הילד הכי גבוה) +1.

שיטת הטיפול: הליכה לצומת השמאלי ביותר שאפשר. כשנעצרים הולכים ימינה.
סדר המעבר (על פי השרטוט): 1,2,4,2,1,3,5,6,5,7,5,3,1.
זמן הפעולה: $O(n)$
סדר הצמתים שמחשבים את עומקם: 1,2,4,3,5,6,7.
סדר הצמתים שמחשבים את גובהם: 4,2,6,7,5,3,1.
חישוב הרמה של כל הצמתים: $O(n)$.
חישוב הגובה של כל הצמתים: $O(n)$.

Pre Order –חישוב הצומת בפעם הראשונה שמבקרים בה.
Post Order– חישוב הצומת בפעם האחרונה שמבקרים בה.

<u>ביקור בעץ בינארי</u>:
כניסה
ביקור ב Pre
left <-
ביקור ב In Order
right <-
ביקור ב Post

סדר הביקור ב In Order: 4,2,1,6,5,7,3.

<u>ביקור בעץ כללי</u>:
ביקור ב Pre
הפעל את כל הילדים <-
ביקור ב Post

הלולאה בכל צומת בטיול בעץ כללי:
- Pre
- סע לילד הראשון
- סע לילד השני
- ...
- סע לילד האחרון
- Post

הנסיעה בעץ כללי היא פעמיים ממספר הקשתות. אם יש לנו n צמתים, אז הנסיעה היא בסדר גודל של $O(n)$.

נבקר בכל ילד פעם אחת – כל ילד מסומן ב C_i.

$$\sum_{n=1}^{n} C_i = n-1$$

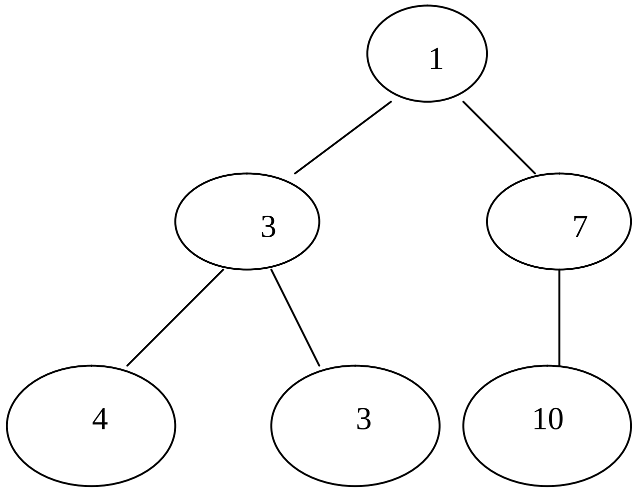
<u>ערימה</u>

n רשומות - לכל רשומה מפתח.

מתחילים ממבנה נתונים ריק
לבנות מבנה נתונים
לתכנן אלגוריתמים

עלינו לענות בצורה יעילה על השאלות והפעולות הבאות:
א. מצא את הרשומה החביבה ביותר.
ב. הוצא את הרשומה החביבה ביותר ממבנה הנתונים שבנית.
ג. הוסף רשומה למבנה הנתונים שבנית.
ד. בנה את מבנה הנתונים ל n רשומות.

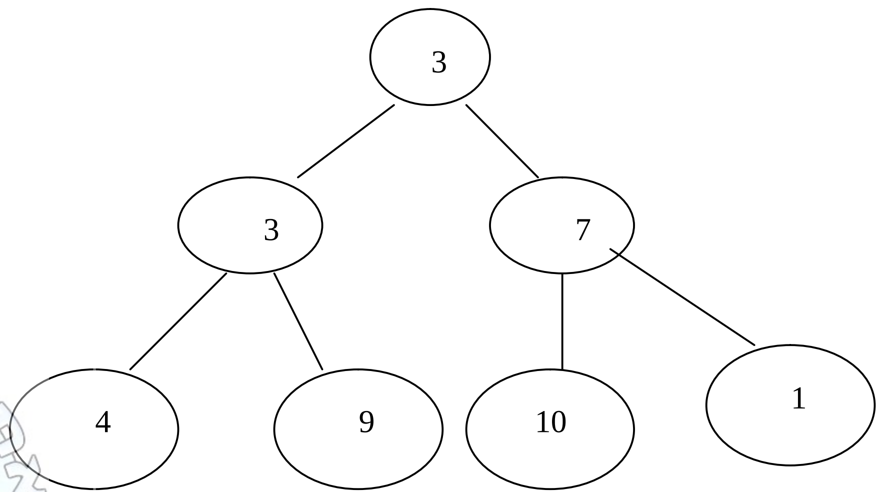
<u>מבנה ערימה בינארית</u>
א. עץ בינארי שלם מאוחסן במערך.
ב. כל רשומה בצומת V "חביבה" יותר (או שווה) מכל הרשימות בתת העץ שלה.



<u>פעולה בסיסית על צומת ושני ילדיו:</u>
1. השווה בין הילדים.
2. השווה בין ההורה והמנצח מצעד 1.
3. אם הילד ניצח בצעד 2, החלף בין ההורה והילד.

נוציא את 1 ונבנה את העץ מחדש.
נכניס במקומו את העלה האחרון (10).
בכל שלב יש לנו מפתח אחד שאינו במקום. תת העץ משמאלו שומר על תכונות הערימה. תת העץ מימינו גם שומר על תכונות הערימה. כך גם תת העץ של ההורים שמעל למפתח שאינו במקום.
כל פעם מחליפים את החביב ביותר עם החביב ביותר משני בניו. כעת נעשה את אותו הדבר בערימה בדרגה הנמוכה יותר. כך נמשיך עד שהאיבר שאנו מחליפים אותו יהיה חביב יותר משני בניו.
זמן הפעולה: $O(\log n)$.

הוספת איבר למבנה הנתונים:

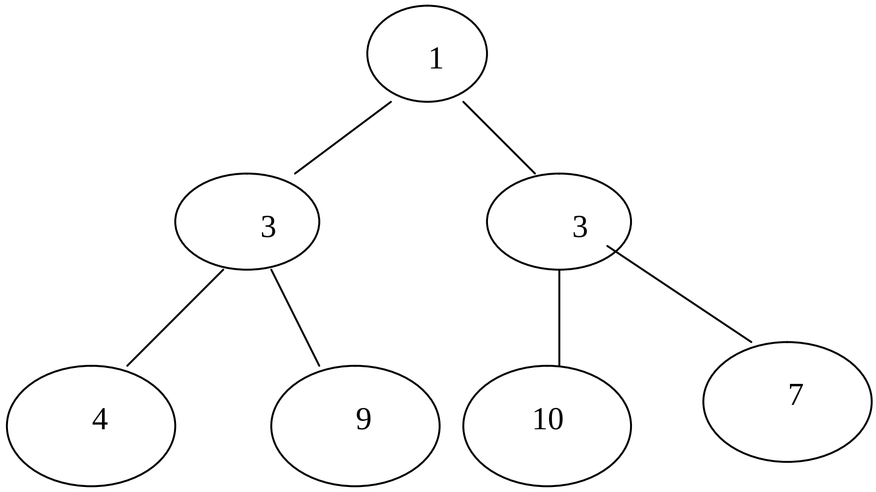


הוספנו איבר (1).

המערך:

3	3	7	4	9	10	1			

לאחר ההחלפות:

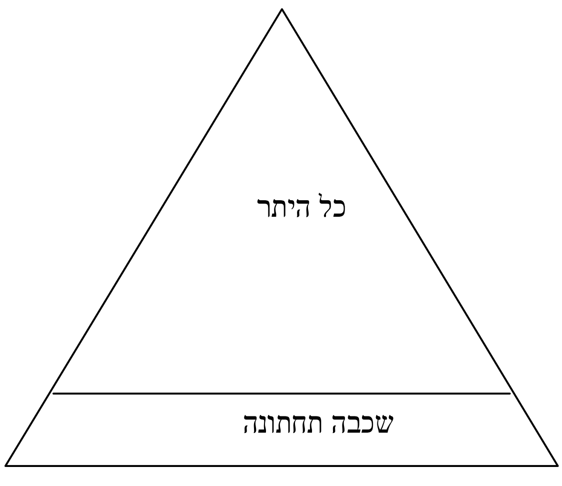


נשתמש באותה פעולה בסיסית שהגדרנו, על מנת לסדר את הערימה.
הזמן: $O(\log n)$

<u>בניית ערימה</u>

$$n = 2^1 - 1$$
$$i = \log(n+1)$$

אברי השכבה התחתונה של הערימה: $2^{i-1}...2^i - 1$
כל היתר: $1...2^{i-1} - 1$



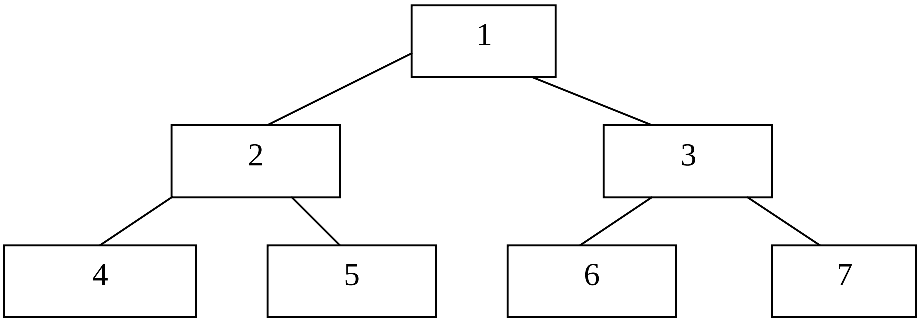
$$\frac{n+1}{2} \cdot \log(n+1) = O(n \log n)$$

גם אם יסדרו לנו את כל השכבות העליונות וישאר לנו לסדר רק את השכבה התחתונה, עדיין הסיבוכיות תהיה $n \log n$.

<u>שיטת סידור הערימה</u>

נתחיל מהעלים. כל אחד מהעלים הוא ערימה מסודרת בפני עצמה. משם נעלה כל שלב למעלה. סיבוכיות הבניה: $O(n)$.

בסיום הטיפול ברמה i כל תתי העצים ששורשיהם ברמה i הם ערימות חוקיות.



בכל פעם נבצע פעולה בסיסית כגובה תת העץ.

חישוב סיבוכיות האלגוריתם:

$\frac{n}{2}$ עלים

$\frac{n}{4}$ שכבה מעליהם

מספר הפעולות: $\frac{n}{2}\cdot 0+\frac{n}{4}\cdot 1+\frac{n}{8}\cdot 2+...+1\cdot\log n$

$$\sum_{i=0}^{\log n}\frac{n}{2^{i+1}}\cdot i=n\sum_{i=0}^{\log n}\frac{i}{2^{i+1}}\leq n\sum_{i=0}^{\infty}\frac{i}{2^{i+1}}=O(n)$$

ככל שהשורה יותר קרובה לשורש, אנו נבצע עליה את הפעולה הבסיסית במקרה הגרוע יותר פעמים. על שורת העלים אנו לא מבצעים פעולות. על שורת ההורים שלהם אנו מבצעים פעולה אחת. על השורה מעליהם אנו מבצעים במקרה הגרוע שתי פעולות (פעולה בסיסית עליהם ועל הבנים שלהם ופעולה בסיסית נוספת במקרה שצריכים להוריד אותם עוד שורה למטה). כך על כל שורה נוספת נבצע במקרה הגרוע עוד פעולה ככל שנתקרב לשורש.

הפעולה הבסיסית - Heapify:

- מקבלת אינדקס i
- תכונת הערימה נשמרת מעליו
- תכונת הערימה נשמרת בשני תתי העצים שמתחתיו

```
if (heap[2i]>heap[2i+1])
      swap(i,2i)
else
      swap(i,2i+1)
```

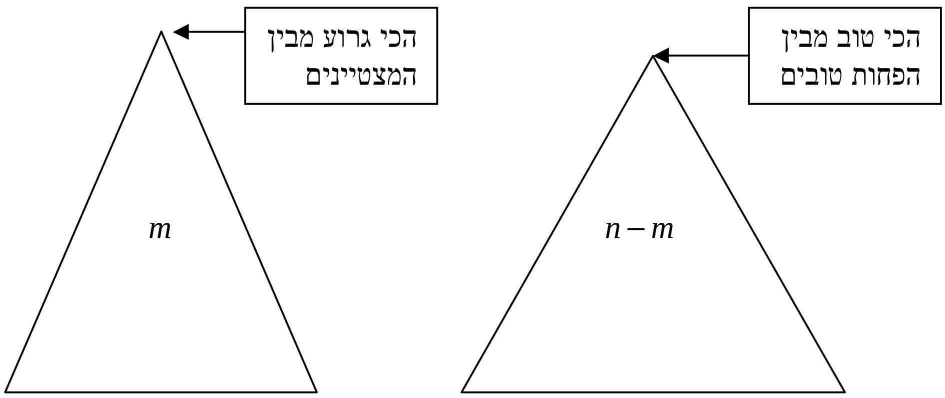
התיקון

- החלפת האיבר במקום i עם הבן הגדול יותר.

<u>שאלה</u>

א. בקורס יש n סטודנטים. לכל סטודנט ציון (מס' שלם 0-100). המרצה מעוניין למצוא את m המצטיינים. יש לאפשר פעולה של הוספת סטודנט ועדכון ציון של סטודנט.

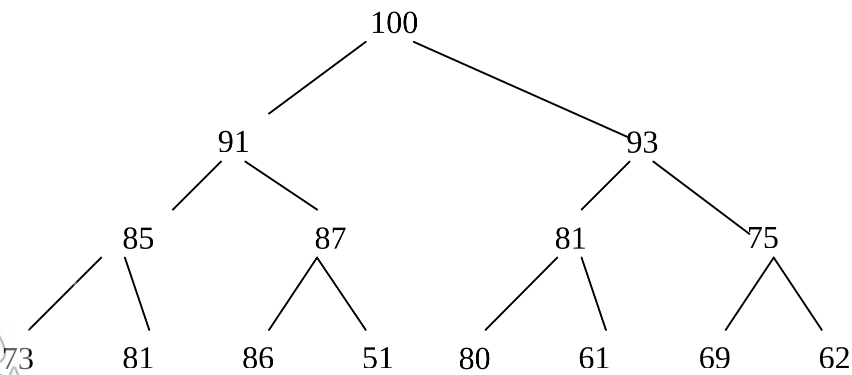
פתרון:
- בניית ערימת מקסימום בגודל n. סיבוכיות הבניה: $O(n)$.
- m הוצאות מערימת המקסימום לערימת המינימום.
  - o הוצאה אחת: $O(\log n)$
  - o m הוצאות: $O(m\log n)$
  - o m הכנסות: $O(m\log m)$



בניית מבנה הנתונים: $O(n+m\log n)$
הוספת סטודנט: $O(\log n)$
עדכון ציון של סטודנט: $O(\log n)$

ב. המרצה רוצה להדפיס את m הציונים המצטיינים.

פתרון:
- בניית ערימת מקסימום בגודל n.



נבנה ערימת מצטיינים בגודל m.
באתחול היא תכיל רק את שורש הערימה הכללית.

בלולאה מ 1 עד m:

- הוצאת מקסימום מערימת המצטיינים $O(\log m)$
- הוספת 2 הבנים של המקסימום מהערימה הכללית לערימת המצטיינים $O(\log m)$
- סה"כ: $O(n + m\log m)$

בכל שלב נוציא את השורש מערימת המצטיינים ונוסיף לערימה מצביעים לשני הבנים שלו. כך נעשה m פעמים.

<u>סיבוכיות הפעולות בערימה בינארית</u>

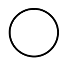
1. בניית ערימה $O(n)$
2. הוספת איבר $O(\log n)$
3. בדיקת מהו המקסימום $O(1)$
4. הוצאת המקסימום $O(\log n)$

<u>עצים בינומיים</u>

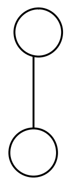
צומת אחד - B_0

איחוד של 2 עצי B_{n-1} - B_i

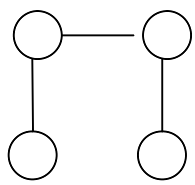
B_0 :



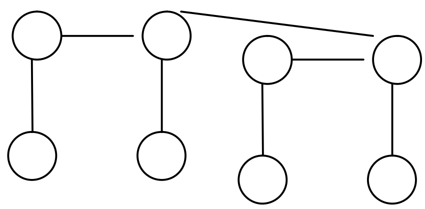
B_1 :



B_2 :



B_3 :



תכונות

B_k - מספר הצמתים 2^k. גובה i.

הוכחה באינדוקציה:

בסיס B_0 - 1

נניח B_i - מספר הצמתים 2^i

נוכיח B_{i+1} - מספר הצמתים 2^{i+1}

בסיס B_0 - 0

נניח B_i - גובהו i

נוכיח B_{i+1} - גובהו $i+1$

B_k - דרגת השורש היא k והיא הגדולה בעץ.

הוכחה:

בסיס B_0 - 0

נניח B_i - i ילדים לשורש

נוכיח B_{i+1} - בעל מספר הילדים הגדול ביותר

סיכום תכונותיו של עץ בינומי B_k

1. מכיל 2^k צמתים
2. גובהו k
3. ישנם בדיוק $\binom{k}{i}$ צמתים בעומק i.
4. דרגת השורש היא k - הדרגה הכי גבוהה בעץ.
5. הבנים של השורש הם עצים בינומים.
6. אם מספר הצמתים בעץ B_k הוא n, כאשר $n = 2^k$, גובהו הוא $\log n$.

יער בינומי

- אוסף של עצים בינומים
- אין 2 עצים בגודל שווה

נדע לזהות יער בינומי לפי הייצוג הבינארי שלו

29=11101

יהיה לנו עץ אחד של B_0, עץ אחד של B_2, עץ של B_3 ועץ של B_4.

ערימה בינומית

- n איברים ביער בינומי.
- השורשים מקושרים ברשימה מקושרת.
- כל עץ בינומי מקיים את תכונת ה"חביב".
- בערימה בינומית עם n צמתים יש $O(\log n)$ עצים, משום ש n הוא סכום של חזקות של 2.

פעולות על ערימה בינומית

א. מצא חביב - $O(\log n)$

ב. הוצא חביב - $O(\log n)$

ג. הוסף איבר - $O(\log n)$

ד. בנה מ n איברים $O(n)$

<u>איחוד 2 ערימות בינומיות</u>

ערימה 1: $29 = B_0 \;\; B_2 \;\; B_3 \;\; B_4$

ערימה 2: $15 = B_0 \;\; B_1 \;\; B_2 \;\; B_3$

$$B_0 + B_0 = B_1$$

$$\begin{array}{cccccc}
 & B_1 & B_2 & B_3 & B_4 & B_5 \\
B_0 & & B_2 & B_3 & B_4 & \\
B_0 & B_1 & B_2 & B_3 & & \\
\hline
 & & B_2 & B_3 & & B_5
\end{array}$$

איחוד 2 ערימות בנות m ו n איברים: $O(\log(n+m))$

<u>בניית ערימה בינומית</u>

פעולות	מספר האיבר
1	1
2	2
1	3
3	4
1	5

מספר הפעולות הוא כמספר העצים השונים בערימה.
אפשר לראות שכל פעם שמספר האיברים הוא זוגי, מספר הפעולות בהכנסת איבר נוסף יהיה 1.

$$\frac{n}{2}\cdot 1+\frac{n}{4}\cdot 2+\frac{n}{8}\cdot 3+\ldots = O(n)$$

יתרונות הערימה הבינארית על הבינומית: הוצאת האיבר המקסימלי לוקחת פחות זמן. ערימה בינארית נמצאת במערך ולכן תופסת פחות מקום.

<u>ההבדלים בסיבוכיות בין שני סוגי הערימות</u>

מציאת מקסימום:
ערימה בינארית: $O(1)$
ערימה בינומית: $O(\log n)$

איחוד ערימות:
ערימה בינארית: $O(n)$
ערימה בינומית: $O(\log n)$

<u>מיון</u>

קלט: n רשומות. לכל רשומה מפתח $a_1,...,a_n$

פלט: הרשומות מסודרות $a_{i_1},a_{i_2},...,a_{i_n}$ כך ש $a_{i_j} \le a_{i_{(j+1)}}$ עבור כל $1 \le i_j \le n$.

<u>מאפייני המיון</u>

1. <u>מיון במקום</u> – המיון לא דורש יותר מ $O(1)$ מקום בנוסף לקלט.
2. <u>מיון השוואה</u> – מיקומו של כל איבר נקבע ע"י השוואת איברים.
   <u>משפט</u>: מיון המבוסס על השוואות בלבד של n מפתחות יקח $O(n\log n)$ במקרה הגרוע והממוצע.
3. <u>מיון יציב</u> – אם יש איברים זהים על פי סדר מסויים, הם יהיו על פי הסדר גם לאחר המיון.
   לדוגמא: הקלט $3^a 1 3^b 0 5 3^c$ יהיה $0 1 3^a 3^b 3^c 5$. זהו מיון אשר לא משנה סדר יחסי של איברים בעלי מפתחות זהים.

<u>סוגי מיונים (לפי סיבוכיות)</u>

$O(n^2)$: Max, הכנסה, מהיר (במקרה הגרוע).

$O(n\log n)$: ערימה, מיזוג, מהיר (במקרה הטוב).

$O(n)$: בסיס, מניה, דלי.

<u>Max Sort (במקום, השוואה, ניתן למימוש כיציב)</u>

בכל איטרציה מקטינים את המערך שעליו מחפשים ב 1.

$$n+(n-1)+(n-2)+...+1=O(n^2)$$

5,8,3,1,7

נסרוק את המערך, נמצא מקסימום ונחליף בינו ובין המקום הראשון.

8,5,3,1,7

8,7,3,1,5

8,7,5,1,3

8,7,5,3,1

$$\sum_{i=1}^{n}(n-i+1)=O(n^2)$$

<u>מיון בועות (במקום, השוואה, יציב)</u>

```
swapped=true;
for(i=(n-1);(i>0) && swapped;i--) {
      swapped-false;
      for(j=0;j<1;j++) {
            if(A[j]>A[j+1]) {
                  swapped=true;
                  swap(A[j],A[j+1]);
            }
      }
}
```

האלגוריתם בודק אם השמאלי גדול מהימני. אם לא, הוא מחליף אותם.

הזמן במקרה הגרוע הוא עדיין $O(n^2)$, אך ישנה אפשרות שהוא יעצר לפני כן.

מיון הוספה

5,8,3,1,7

בכל שלב התחלת המערך ממויינת וכל פעם מגדילים את החלק הממויין ב 1.

```
for(j=2;j<n;j++) {
      key=A[j];
      i=j-1;
      while((i>0) && (A[i]>key)) {
            A[i+1]=A[i];
            i=i-1;
      }
      A[i+1]=key;
}
```

השלב ה i:

מיון ערימה (במקום, השוואה, לא יציב)

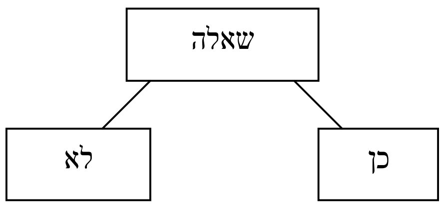
שלב א': בנה ערימה (ערימה בינארית המאוחסנת במערך). $O(n)$.

שלב ב': $n-1$ צעדים. בכל צעד הוצא את החביב והכנס לסוף המערך. $O(n\log n)$.

אזור ממוין	ערימה

ניתן לבצע זאת באותה סיבוכיות גם בערימה בינומית.

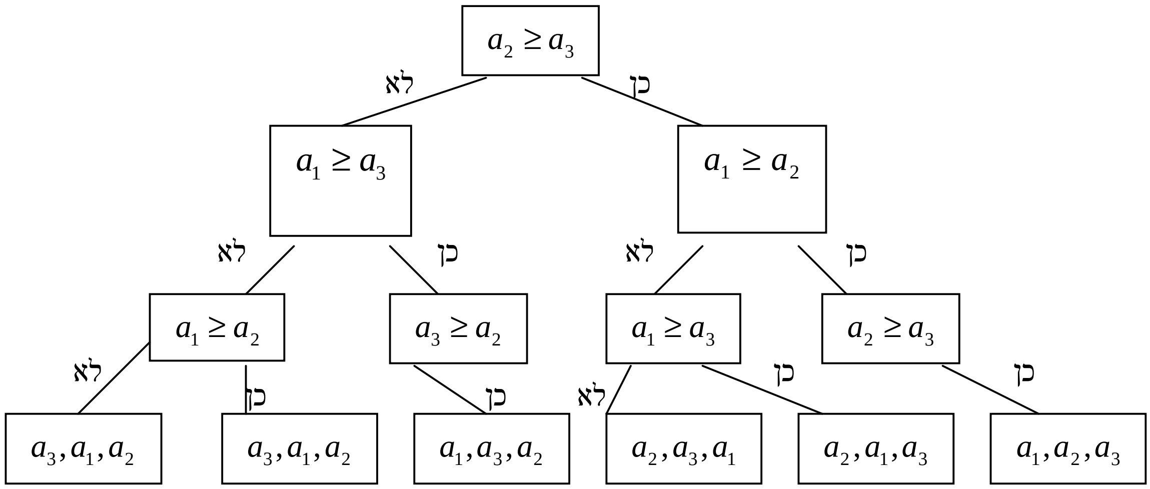
עץ החלטה



עץ המתאר Bubble Sort:

a_1, a_2, a_3

7,5,9



$$\sum_{i=1}^{n-1}(n-i)$$

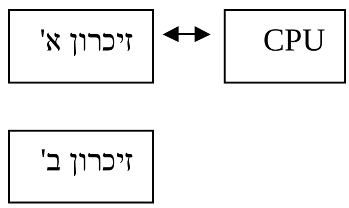
$$\sum_{i=1}^{3-1}(3-i)=(3-1)+(3-2)=3$$

במיון לכל יחס שונה בין איברים (פרמוטציה) יש מסלול שונה בעץ.
למשל: 7,5,9 ו 17,15,19 הם בעלי אותו מסלול. ל 5,7,9 יש מסלול שונה.
כל פרמוטציה מסתיימת בעלה.

א. לכל מיון עץ החלטה (בינארי).
ב. המסלול הארוך בעץ מבטא את סיבוכיות הזמן של האלגוריתם.
ג. מספר העלים בעץ יהיה לפחות (בדרך כלל שווה) מספר הפרמוטציות שהוא $n!$.

גובהו של עץ החלטה של אלגוריתם למיון הוא לפחות $\log_2(n!) \approx n\log n$.

<u>גורמים המשפיעים על מהירות המחשב:</u> CPU, תקשורת פנימית ותקשורת חיצונית.



אם יש אלגוריתם שאינו זקוק לזיכרון ב', הוא עדיף על אלגוריתם שזקוק לזיכרון ב', כמעט בכל מצב.
אנו לא מקבלים את האינפורמציה ביחידות, אלא בבלוקים.

מיון ערימה הוא מיון שלא מטפל טוב באיברים בזיכרון (קופץ מבלוק לבלוק במערך).
גודל בלוק: B.

ב Max Sort נצטרך להחליף בלוקים $\frac{B}{n^2}$ פעמים.

נראה דוגמא של עבודה על טייפ במיון מיזוג:

2 קבצי קלט: I_1, I_2.

2 קבצי פלט: O_1, O_2.

סיבוב 1:

קלט: גודל האזור הממויין: 1 (האיבר הראשון הוא ממויין).

פלט: אזורים ממויינים בגודל 2.

סיבוב 2:

קלט: גודל האזור הממויין: 2.

פלט: אזורים ממויינים בגודל 4.

סיבוב אחרון:

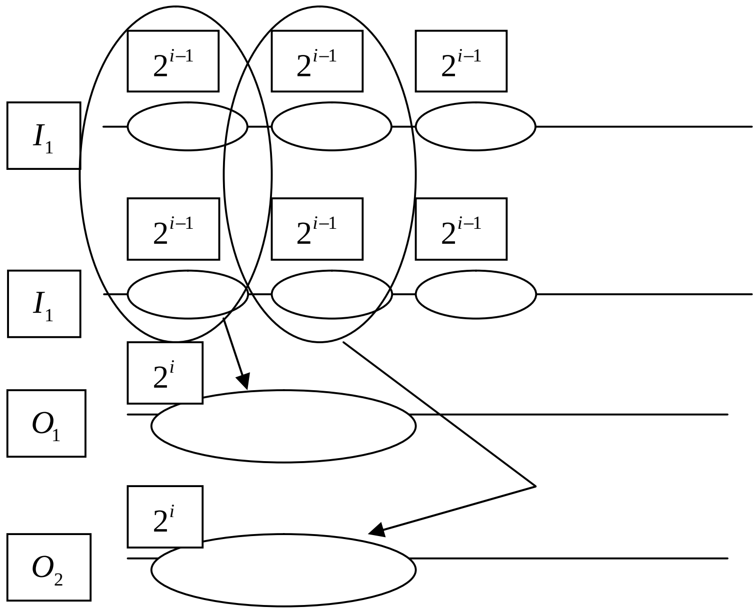
קלט: גודל האזור הממויין: $\frac{n}{2}$.

פלט: אזורים ממויינים בגודל n.

סיבוב i:

קלט: גודל האזור הממויין: 2^{n-1}.

פלט: אזורים ממויינים בגודל 2^i.



סה"כ: $O(n\log n)$.

מספר ההעברות של בלוקים: $\frac{n\log n}{B}$

ניתן לחסוך העברות ע"י מיון פנימי של כל בלוק באיזה מיון שנרצה ולאחר מכן מיזוג הבלוקים.

Quick Sort (במקום, השוואות, לא יציב)

```
QuickSort(A,p,r)
{
      if (p<r)
            then q=partition(A,p,r);
      QuickSort(A,p,r);
      QuickSort(A,q+1,r);
}
```

```
Partition(A,p,r)
{
      x<-A[p]
      i<-p-1
      j<-r+1
      while TRUE
            do repeat j<-j-1
                  until A[j]<=x
                  repeat i<-i+1
                  until A[i]>=x
            if i<j
                  then exchange A[i]<->A[j]
            else return j
}
```

בכל שלב ממיינים $A(i...j)$.

בוחרים Pivot מתוך $A(i...j)$.

למשל נבחר $pivot = A[i]$.

נסדר את $A(i...j)$ כך ש:

בתאים $A(i...y-1)$ כל האיברים הקטנים מה Pivot.

בתאים $A(y...j)$ כל האיברים הגדולים ושונים מה Pivot.

דוגמא:

12,3,14,7,4,11,8,9,13

i j

$x = 12$

12,3,14,7,4,11,8,9,13

i j

החלפה:

9,3,14,7,4,11,8,12,13

i j

9,3,14,7,4,11,8,12,13

i j

החלפה:

9,3,8,7,4,11,14,12,13

i j

9,3,8,7,4,11,14,12,13

j i

זוהי נקודת החלוקה.

עכשיו $x=9$

מערך 1: 9,3,8,7,4,11

אחרי הסידור: 4,3,8,7,9,11

מערך 1 החדש: 4,3,8,7

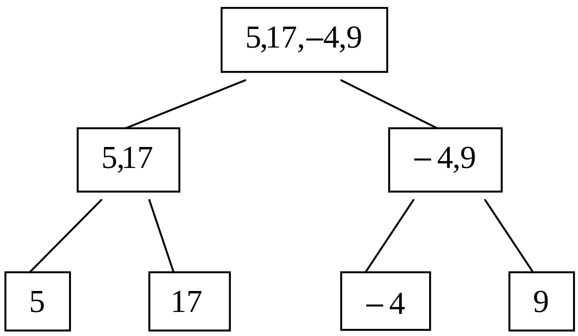
מערך 2 החדש: 9,11

במקרה הממוצע: סיבוכיות $O(n\log n)$.

במקרה הגרוע: סיבוכיות $O(n^2)$.

<u>מיון מיזוג (אפשר לממש במקום, השוואה, יציבות תלויה במימוש)</u>

```
mergesort(array,from,to)
{
      if(from<to) { // יש לפחות 2 איברים
            mid=(from+to)/2;  // חציון
            mergesort(array,from,mid);
            mergesort(array,mid+1,to);
            merge(array,from,mid,to); // מיזוג
      }
}
```



סיבוכיות: $O(n\log n)$.

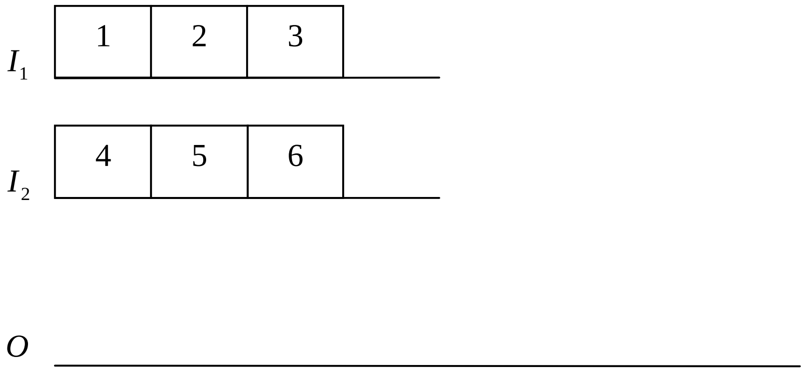
חיסכון בהעברות מיותרות לזיכרון:

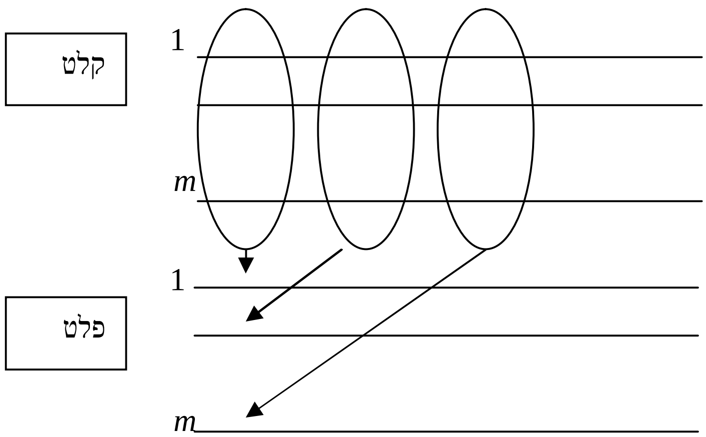
שלב א': לבנות קבוצות בגודל 2^k.

שלב ב': מיון מיזוג בצעדי $k...\log n$

בצורה זו אנו חוסכים $k-1$ העברות לזיכרון:

$$2^0 \to 2^1$$
$$2^1 \to 2^2$$
$$\vdots$$
$$2^{k-1} \to 2^k$$



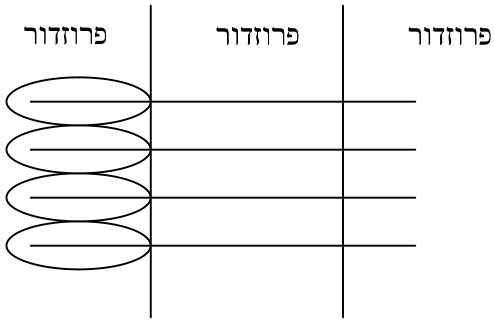


יש לנו m קבצי קלט ו m קבצי פלט. בשלב הראשון יש לנו קבוצות בגודל m. בשלב האחרון יש לנו $m^x = n$. כלומר $x = \log_n m = \dfrac{\log_2 n}{\log_2 m}$.

$$1 \to m$$
$$m \to m^2$$
$$m^2 \to m^3$$
$$m^i \to m^{i+1}$$

ניקח לדוגמא $m = 4$

$$\begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{array}$$



16

נבנה מכל סדרה של בלוקים ערימה. כל חלק מתוך m הבלוקים יהיה איבר בערימה. המפתח יהיה האיבר הראשון שלו. בכל הוצאה מהערימה נמחק את האיבר הראשון מכל בלוק ואז נסדר את הערימה מחדש על פי המפתח החדש.

א. בניית ערימה $O(m)$

ב. הוצאת איבר $O(\log m)$

m - מספר הקבצים.

n - מספר האיברים.

x - מספר האיברים בקבוצה ממויינת.

מספר שלבים $\frac{\log n}{\log m}$

מספר פרוזדורים $\frac{n}{m^{i+1}}$

עלות כל פרוזדור $m^{i+1} \log m$

עלות שלב $O(n \log m)$

עלות כוללת $O(n \log n)$

כאן חסכנו העברות בזיכרון. שילמנו בעלות כל שלב, אך מספר ההעברות בזיכרון ירד. כל איבר נוסע כאן בזיכרון $\frac{\log n}{\log m}$ פעמים. m מוגבל בגודל הזיכרון. אנו רוצים שכל הערימה תהיה בזיכרון. לכן m לא יכול להיות יותר גדול מגודל הזיכרון.

מיונים לינארים

מיון מניה (לא במקום, אין השוואות, יציב)

מיון על פי מספר סוגים קבוע מראש - k. עבור כל איבר מאותו סוג נקדם את המונה בתא במערך המיועד לו. לאחר מכן עוברים שוב על המערך המקורי ושמים בו את האיברים לפי הסדר של המקומות שהם אמורים להיות בהם.
סיבוכיות: $O(n+k)$.

m - מספר האיברים.
D - תחום המפתחות.

שלב א': נקה מערך. $O(D)$

D			…	2	1

שלב ב': עדכן מונה של כל איבר לפי הנתונים. $O(n)$
שלב ג': פלט. $O(D)$

אם האיברים הם לא רק המפתחות, אלא יש מאחוריהם גם שדה, נשים בתאים מצביעים לשדות.

D			…	2	1

פלט: $O(m+D)$.

דוגמא אחרת למיון מניה:

1. לכל i מ 1 עד k $C[i] \leftarrow 0$. $O(k)$
לכל i מ 1 עד n $C[A[i]]++$. $O(n)$

A:

8	7	6	5	4	3	2	1
2	6	5	2	1	2	1	6

C:

6	5	4	3	2	1
2	1	0	0	3	2

2. לכל i מ 2 עד k $C[i] \leftarrow C[i]+C[i-1]$

C:

6	5	4	3	2	1
8	6	5	5	5	2

3. לכל i החל מ n עד ל 1 $B[C[A[i]]] \leftarrow A[i]$, $C[A[i]]--$ $O(n)$

6 במקום השמיני:

C:

1	2	3	4	5	6
2	5	5	5	6	7

B:

1	2	3	4	5	6	7	8
							6

1 במקום השני:

C

1	2	3	4	5	6
1	5	5	5	6	7

B:

1	2	3	4	5	6	7	8
	1						6

וכך הלאה

אם K הוא קבוע ניתן להפוך את האלגוריתם למיון במקום (אבל אז לא תהיה יציבות).

<u>מיון דלי</u>

כאן אנחנו חייבים שתהיה התפלגות אחידה של מספר הציונים. נקצה מערך בו יש תאים רבים יותר לציונים באמצע ותאים מעטים יותר לציונים בקצוות.

<u>מיון בסיס Radix Sort</u>

נתונים n מספרים בעלי d ספרות כל אחד המוצגים בבסיס b, אזי ניתן למיינם בזמן $O(d\cdot(n+b))$.

כאשר b,d קבועים נקבל מיון לינארי.

נבצע את המיון ב d שלבים, כאשר בשלב ה i נמיין את המספרים לפי הספרה ה i (מהסוף) בלבד. בכל שלב נשתמש במיון מניה. מכיוון שנשתמש במיון מניה יציב, כל שלב לא הורס את תוצאות השלבים הקודמים.

לדוגמא:

$b=10$

$d=3$

329
657
457
839
436
720
355

שלב 1 (ספרה אחרונה):
720
355
436
657
457
329
839

שלב 2 (ספרה אמצעית):
720
329
436
839
355
657
457

שלב 3 (ספרה ראשונה):
329
355
436
457
657
720
839

<u>תרגיל</u>

נתונים n מספרים שלמים בתחום $[0,(n^2-1)]$. הצע דרך יעילה למיינם.

- נשים לב כי כל המספרים מהתחום, כשמוצגים בבסיס n, הינם בעלי שתי ספרות לכל היותר.

א. נעבור לבסיס n.
להעברת מספר x לייצוג בבסיס n (נסמנו a_1a_0) נבצע:

$$\left.\begin{array}{l} a_0 \leftarrow x \bmod n \\ a_1 \leftarrow \left\lfloor \frac{k}{n} \right\rfloor \end{array}\right\} O(1)$$

ב. נפעיל מיון בסיס.

$$d = 2 \quad \text{(מקסימלי)}$$
$$b = n$$
$$O\big(2(n+n)\big) = O(n)$$

<u>**עצי חיפוש**</u>

<u>בעיית המילון</u>
מילון מאחסן אוסף רשומות מהטיפוס (מפתח, אינפורמציה).

<u>פעולות:</u>
- הכנסת איבר למילון.
- הוצאת איבר מהמילון.
- חיפוש איבר.
- מציאת מינימום.
- מציאת מקסימום.
- מציאת עוקב ל x.
- מציאת קודם ל x.

<u>עצי חיפוש</u> – משפחה של מימושים למבנה הנתונים האבסטרקטי "מילון".

<u>רשימה מקושרת לא ממויינת</u>

א. הוסף איבר $O(1)$. אם רוצים למנוע כפילויות $O(n)$.
ב. הוצא איבר $O(n)$.
ג. מצא איבר $O(n)$.

<u>מערך ממויין</u>

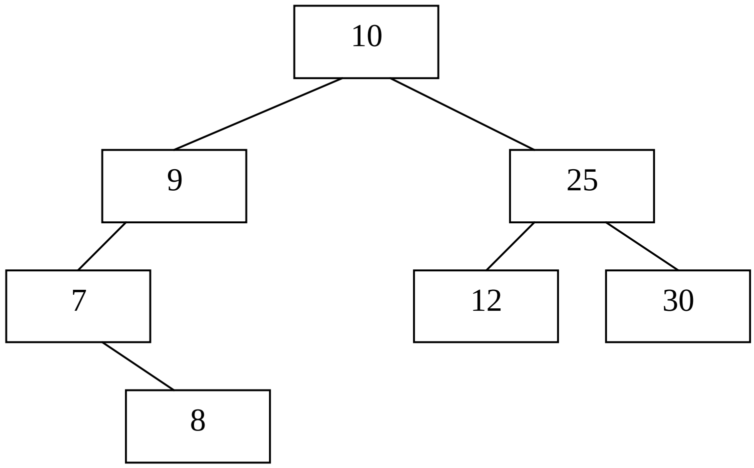
א. הוסף איבר $O(n)$.
ב. הוצא איבר $O(n)$.
ג. מצא איבר $O(\log n)$.

<u>עץ חיפוש בינארי</u>

המפתח בצומת x גדול או שווה לכל המפתחות בתת העץ השמאלי שלו וקטן או שווה לכל המפתחות בתת העץ הימני.

בניית עץ החיפוש: $n \cdot O(h)$.
inorder $O(n)$ - נותן את רשימת האיברים בסדר עולה.

$O(h)$ - גובה העץ.
$O(\log n) \le h \le O(n)$
$O(\log n)$ במקרה הממוצע.



7,8,9,10,12,25,30

בהינתן עץ חיפוש בינארי אנו מקבלים את האיברים ממויינים ב $O(n)$.

<u>חיפוש מפתח</u> - 4 אפשרויות:

1. קיבלנו NULL – המפתח לא נמצא.
2. המפתח שווה – מצאנו.
3. המפתח הרצוי קטן מהשורש – הולכים לבן השמאלי.
4. המפתח הרצוי גדול מהשורש – הולכים לבן הימני.

מצא איבר: עץ מאוזן - $O(h)$. עץ כלשהו במקרה הגרוע - $O(n)$.

<u>הוסף איבר</u>:

1. חפש. עץ מאוזן - $O(h)$. עץ כלשהו במקרה הגרוע - $O(n)$.
2. הוסף אם לא קיים.

סה"כ: עץ מאוזן - $O(h)$. עץ כלשהו במקרה הגרוע - $O(n)$.

<u>מציאת עוקב ל x</u>

נבחן שני מקרים:

א. אם ל x יש בן ימני: y, אזי העוקב הוא המינימום של תת העץ ששורשו y.

ב. אם ל x אין בן ימני – מטפסים מ x למעלה, כל עוד עוברים מבן ימני לאביו. הצומת הראשונה ש x נמצא בתת העץ השמאלי שלה היא העוקב ל x.

סיבוכיות: $O(h)$.

<u>מציאת קודם ל x</u>

נבחן שני מקרים:

א. אם ל x יש בן שמאלי: y, אזי הקודם הוא המקסימום של תת העץ ששורשו y.

ב. אם ל x אין בן שמאלי – מטפסים מ x למעלה, כל עוד עוברים מבן שמאלי לאביו. הצומת הראשונה ש x נמצא בתת העץ הימני שלה היא הקודם ל x.

סיבוכיות: $O(h)$.

<u>הוצא איבר</u>:

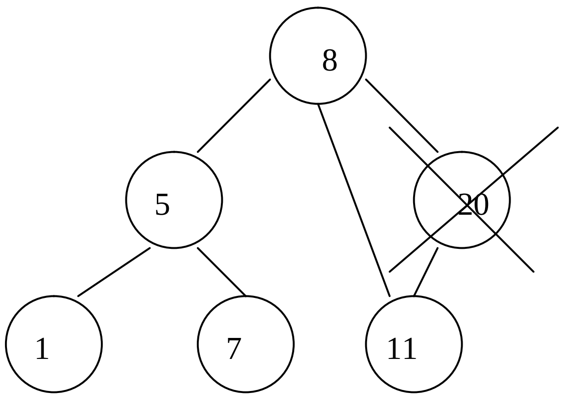
1. חפש.
   - i. לא נמצא - $O(1)$.
   - ii. מצאנו עלה - $O(1)$.
   - iii. הורה לילד אחד. מחברים את הנכד עם הסבא. הנכד יחובר לסבא בצד בו היה מחובר האב. $O(1)$.
   - iv. הורה ל 2 ילדים. כל עץ מורכב משני רבדים. רובד אחד הוא המבנה והרובד השני הוא התוכן. ניתן להחליף את הצומת בבן הכי קטן בתת העץ הימני או בבן הכי גדול בתת העץ השמאלי (הולכים ימינה או שמאלה עד שאין לצומת בן ימני או שמאלי בהתאמה). לאחר מכן:
     - א. החל תוכן של x ב y.
       1. העתקת תוכן.
       2. העתקה מבנית - מבטלים את y ומחברים את המצביעים שלו למצביעים של x.
     - ב. בטל מבנה y - $O(1)$ (אפשרות ii או iii).

סיבוכיות (מציאת y במורד העץ) - עץ מאוזן - $O(h)$. עץ כלשהו במקרה הגרוע - $O(n)$.

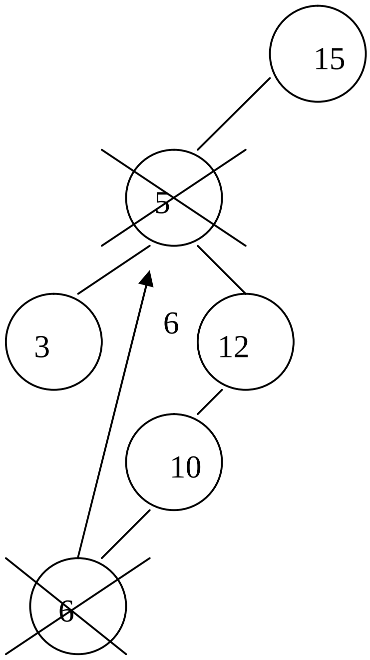
<u>הסבר אחר למחיקת איבר x</u>

נחפש את x. נבחן 3 מקרים:

1. אם ל x אין בנים, נמחק את x.
2. אם ל x יש בן אחד, מוחקים אותו ושמים באבא שלו מצביע לבן של x במקום המצביע ל x.

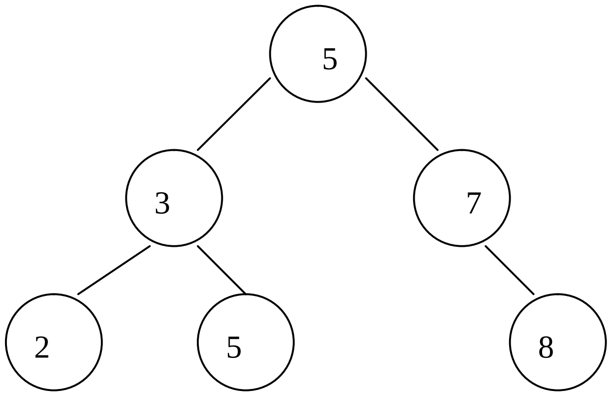


3. אם ל x יש שני בנים: נמצא את העוקב (או הקודם) של x. ידוע שלעוקב אין שני בנים. לכן נוציא אותו ממקומו לפי אחד משני הסעיפים הקודמים. נחליף את y ב x.

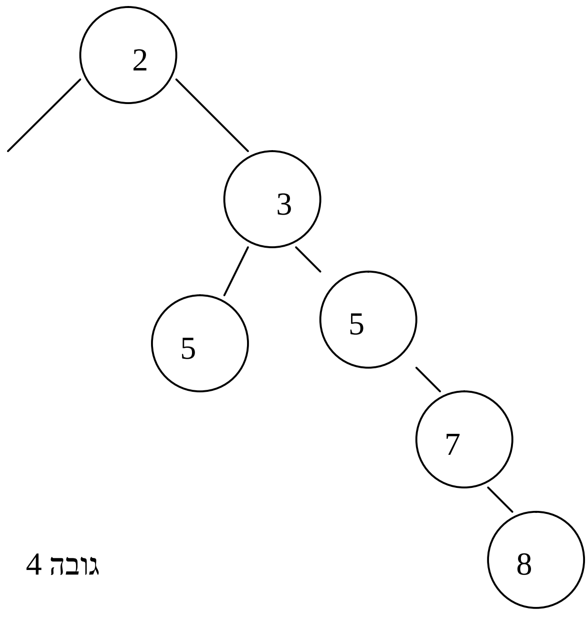


סה"כ: $O(h)$.

<u>מהו h?</u>



גובה 2



גובה 4

$O(\log n) \le h \le O(n)$

כשמכניסים את האיברים על פי הסדר: $O(n)$.

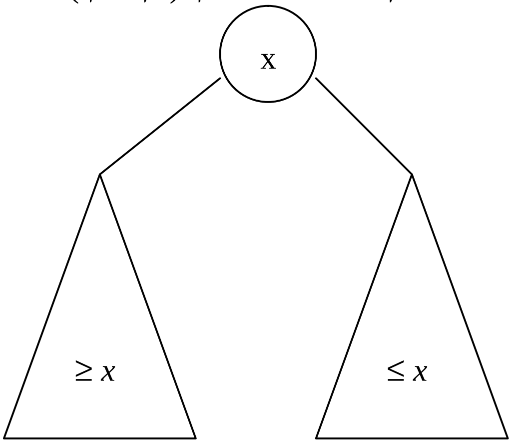
כשמכניסים בצורה אקראית: $O(\log n)$.

הממוצע הוא $O(\log n)$.

הערה: בחיפוש בעצים אם הרשומות כבדות מדי, נעשה את כל הפעילויות עם המפתח של הרשומה ועם מצביע על תוכן הרשומה. נשים את הרשומה בצד ובעץ יהיו שדות רק עם המפתח והמצביע על הרשומה.

<u>שאלה</u>

נתון עץ חיפוש בינארי. רוצים לבנות עץ בינארי שיראה כך (עץ הפוך):



פתרון: עוברים על כל הצמתים ברקורסיה ומחליפים את הצדדים של הבנים.

<u>שאלה</u>

תארו מחיקת צומת כאשר אין תחזוק של מצביע לאבא.

פתרון:

במקרה שיש רק בן אחד:

$O(h)$ חיפוש.

$O(h)$ חיפוש הכולל תחזוקת מצביע לאבא.

במקרה שיש שני בנים:

$O(h)$ חיפוש כולל תחזוקת אביו של x.

$O(h)$ חיפוש כולל תחזוקת אביו של y.

$O(1)$ החלפתם.

<u>שאלה</u>

תאר אלגוריתם לא רקורסיבי יעיל המבצע inorder על עץ חיפוש בינארי.

פתרון:

```
inorder(x)
{
    x<-min(x);
    while (x!=NULL)
    {
        print(x);
        x<-succesor(x);
    }
}
```

$\boxed{O(h)}$

$O(n)\cdot O(h)$

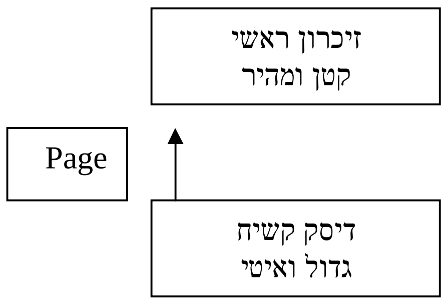
<u>שאלה</u>

כתבו אלגוריתם המקבל עץ בינארי ובודק האם הוא עץ חיפוש בינארי וגם מחזיר את הערך המינימלי והמקסימלי שלו.

תשובה:

```
ret,min,max<-is_binary(x)
{
	if (x->left!=NULL)
	{
		(ret,min,left_max)<-is_binary(x->left)
		if ret==FALSE
			return(FALSE,NULL,NULL)
	}
	else  // x->left==NULL
	{
		min=x->key
		left_max=x->key
	}
	if (x->right!=NULL)
	{
		(ret,right_min,max)<-is_binary(x->right)
		if ret==FALSE
			return (FALSE,NULL,NULL)
	}
	else
	{
		max=x->key;
		right_min=x->key
	}
	if right_min<x->key OR left>x->key
		return FALSE
	return (TRUE,min,max)
```

<u>עץ 2-3-4</u>



<u>B Tree</u> – עץ מאוזן שכל אחד מצמתיו מוגבל לגודל ה Page (גודל הזיכרון שאפשר להסיע כל פעם מהזיכרון המשני לראשי).

תכונות:

1. עץ מושרש.
2. כל צומת מכיל את האינפורמציה:
   - מספר מפתחות $k-1$

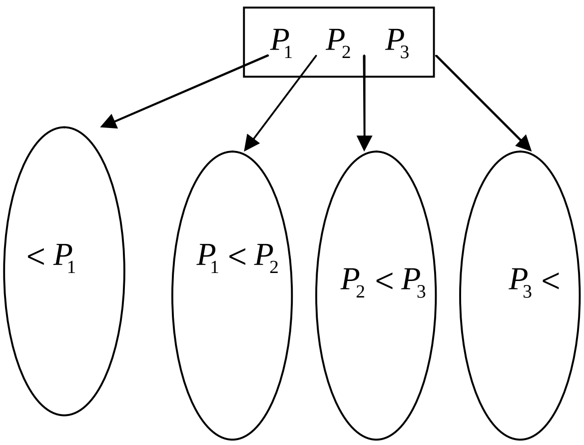
- המפתחות עצמם בסדר לא יורד.
- האם הצומת הוא עלה.
- אם הצומת פנימי, מצביעים לילדים (k).
- המפתחות הם מפרידי תחומים (ערכי הילדים הם בין ערכי המפתחות).
- כל העלים הם באותו עומק.
- לכל צומת יש בין t ל $2t$ בנים, כלומר בין $t-1$ ל $2t-1$ מפתחות.

3. גובהו $\log_t n$, כלומר נמוך יותר מעץ חיפוש בינארי $t \ge 2$. אם t קבוע $O(\log n)$. סיבוכיות הפעולות: $O(t \cdot h)$.

<u>עץ 2-3-4 (מקרה פרטי של עץ B)</u>

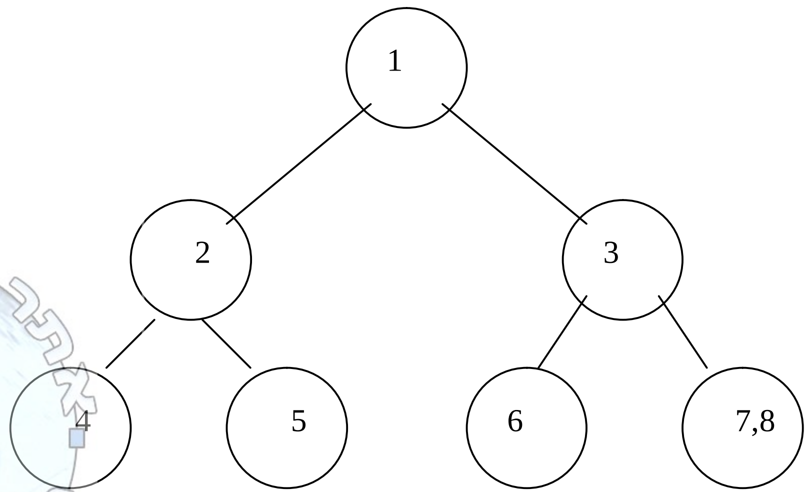
לכל צומת יש 2, 3 או 4 ילדים.

לכל צומת אם יש מפתח אחד יש 2 תתי עצים, אם יש שני מפתחות יש 3 תתי עצים ואם יש שלושה מפתחות יש 4 תתי עצים.



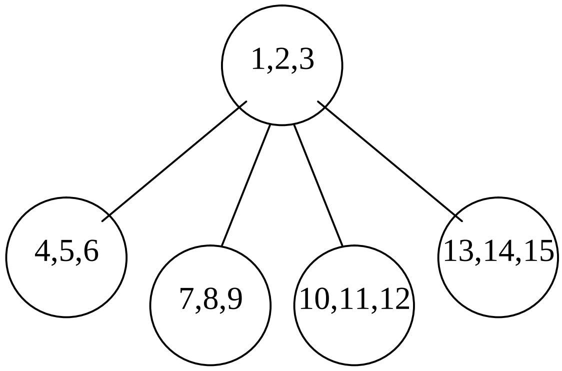
בעצי 2-3-4 כל העלים באותה רמה.

עץ 2-3-4 הדליל ביותר יהיה עץ בינארי (המספרים הם האינדקסים):



$2^i - 1$ מינימום הצמתים כש i הוא הגובה.

נסתכל על העץ העמוס ביותר:



<u>הוספה</u>

נבצע חיפוש לאיבר:

נניח שאין כפילות – הגענו לעלה.

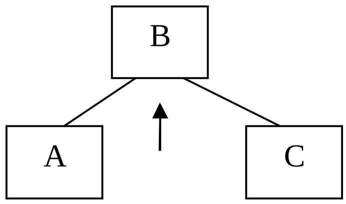
א. בעלה יש מקום – נוסיף.
ב. בעלה אין מקום – פיצול.

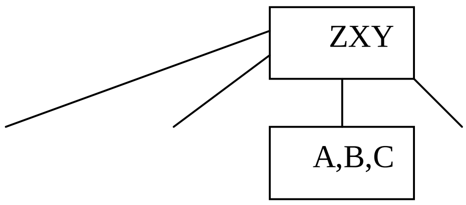
<u>פיצול</u>

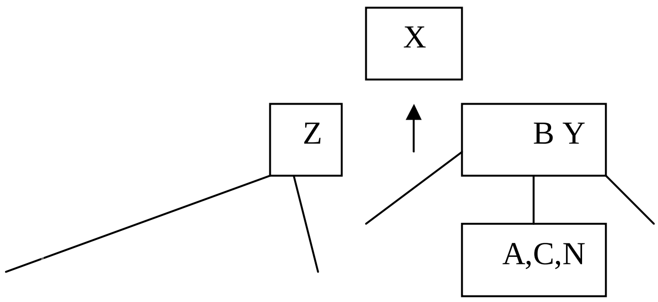
נפצל את הצומת שבו 3 מפתחות.

היו לנו 3 מפתחות: A,B,C.

נעביר את הצומת האמצעי להורה:



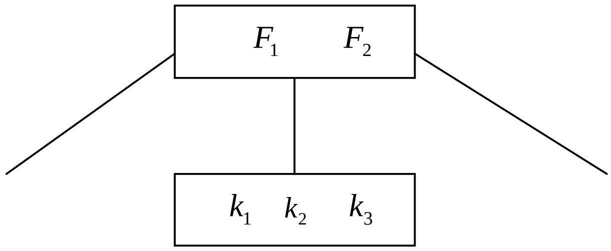




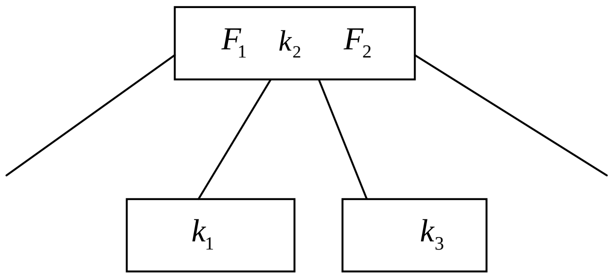
כשאנחנו מטפסים למעלה יש כמה אפשרויות:

א. יש מקום.
ב. אין מקום – מפצלים ועולים למעלה.
ג. יצרנו שורש חדש.

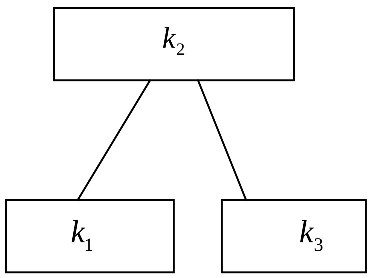
הסבר נוסף לפיצול צומת מלא:



המפתח החיצוני (k_2) עובר לאבא ושני המפתחות שנשארו מתפצלים לשני בניו.



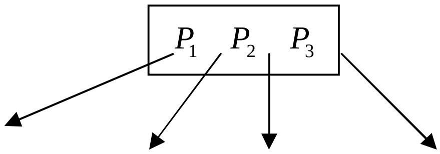
אם אין לצומת אבא, ניצור צומת חדש עם k_2 וגובהו של העץ גדל ב 1.



שיטה אחרת:
כשנטייל למטה, כל פעם כשנגיע לצומת עם 3 מפתחות נפצל אותו. לאחר מכן כשנגיע למקום המתאים נדע שלאיבר החדש יהיה בטוח מקום למעלה.

כדי להקטין את המקום הרב בזיכרון שתופס כל צומת (עלות הסעה גבוהה של הבלוק אל הזיכרון), נאכסן במקום כל רשומה בצומת רק את המצביע לרשומה ואת המפתח.

P_i - מחזיק את מפתח הרשומה ואת המצביע אל הרשומה שנמצאת במחסן.



מקסימום מפתחות = מקסימום צמתים X 3.

מקסימום מפתחות לעץ ברמה i = $4^{i+1}-1$.

אם גובה העץ הוא h, מספר המפתחות הוא $4^{h+1}-1 \ge n \ge 2^{h+1}-1$.

$$n \ge 2^{h+1}-1$$
$$n+1 \ge 2^{h+1}$$
$$\log(n+1) \ge h+1$$
$$\log(n+1)-1 \ge h \ge \log_4(n+1)-1$$

<u>ביטול מפתח</u>

א. מצא.
   1. אין.
   2. עלה.
   3. בצומת פנימי.

ב. בטל.
   1. כלום.
   2. פירוט מייד.
   3. נחליף בעוקב (ונטפל בעלה – ראה מקרה 2).

<u>מקרה 2 – המפתח נמצא בעלה:</u>

– לאחר שהוצאנו את המפתח, נשארו מפתחות בעלה.

– העלה נשאר ריק.

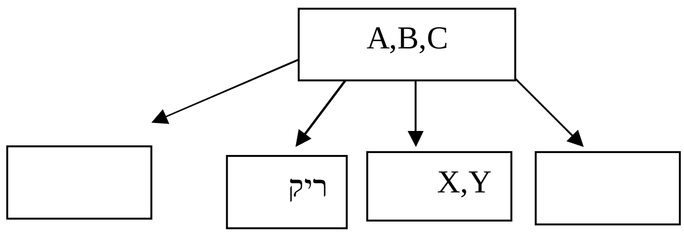
א. עזרה מאח.

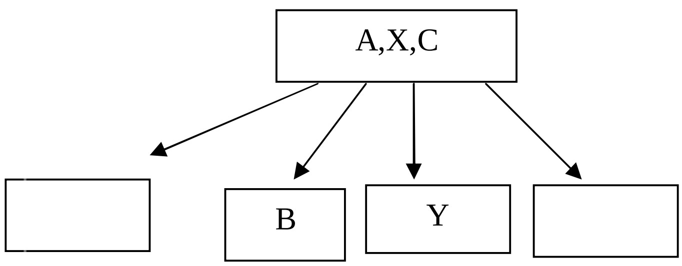
ב. עזרה מההורה.

ג. הבעיה עולה רמה.

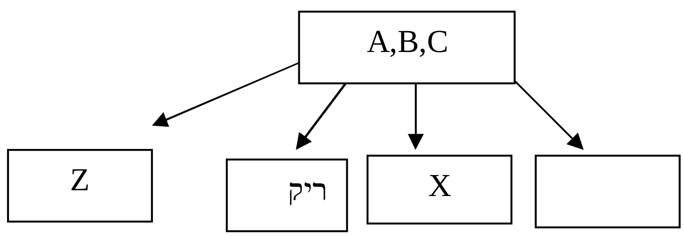
א. עזרה מאח:

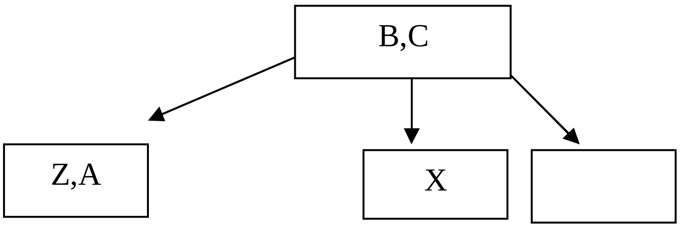
הורה:



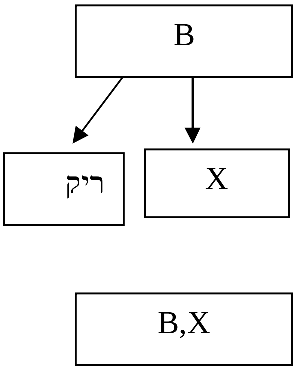


ב. עזרה מהורה:

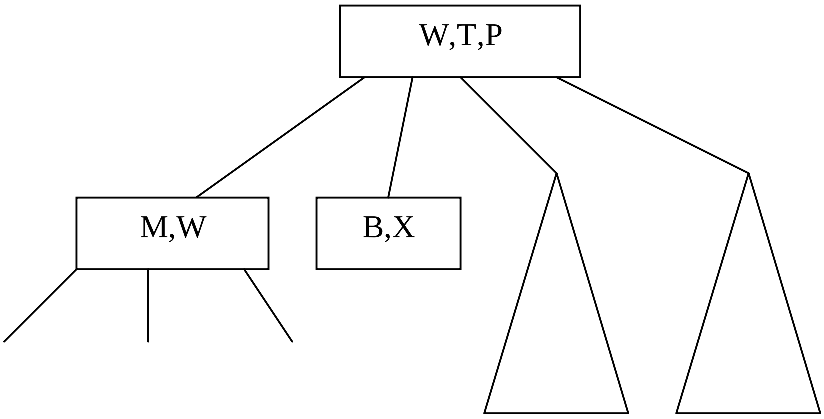


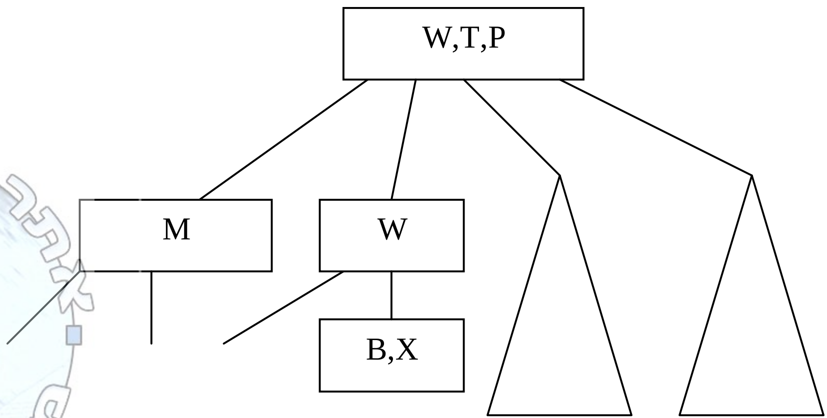


ג. הבעיה עולה רמה:

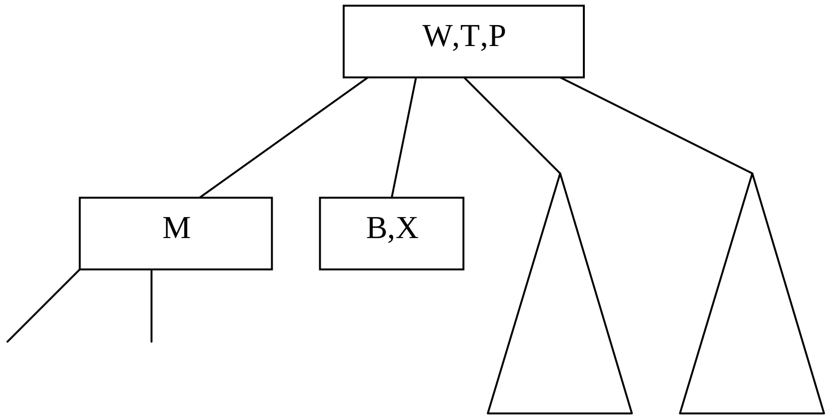


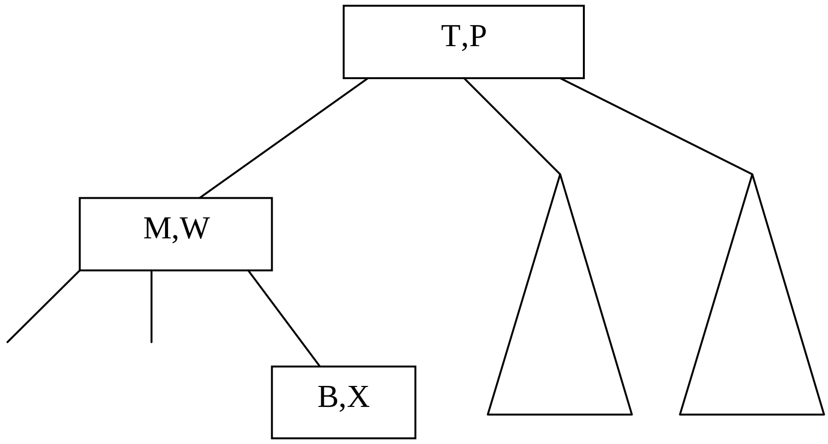
עכשיו יש שוב 3 אפשרויות:
עזרה מאח:



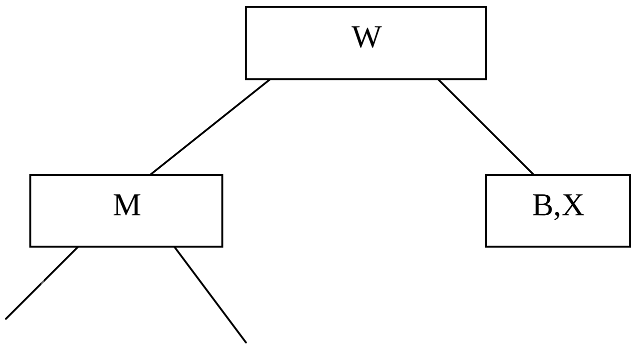


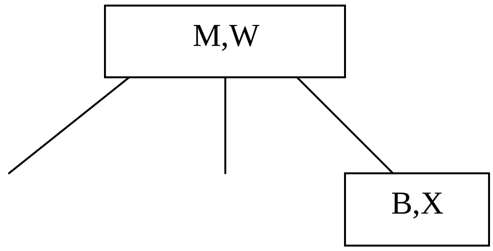
עזרה מהורה:





הבעיה עולה רמה:



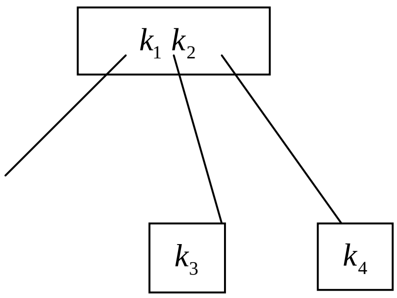


כל אחד משלבי הביטול (לא כולל החיפוש) יקח $O(1)$. במקרה הגרוע נצטרך להוריד רמה מהעץ, כלומר $O(\log n)$.

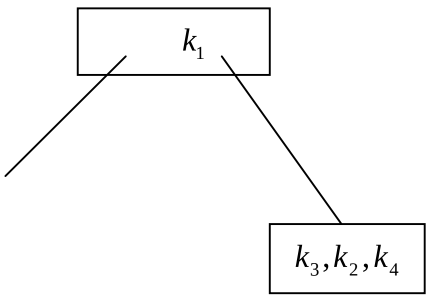
<u>הסבר נוסף למחיקת x</u>

א. אם x נמצא בצומת פנימי, נחליפו בעוקב / קודם שלו ונמחק את העוקב / קודם.

- אם לקודם יש יותר ממפתח אחד
  $\Leftarrow$ נחליף את הקודם k_4 ב k_2 (ונמחק את k_2). את הוצאת k_4 נפעיל רקורסיבית עד העלה.
- אם לקודם יש מפתח אחד ולעוקב יש יותר ממפתח אחד
  $\Leftarrow$ החלף את k_5 ב k_2 ונטפל בהוצאת k_5 רקורסיבית עד העלה.
- אם גם לעוקב וגם לקודם יש רק מפתח אחד:

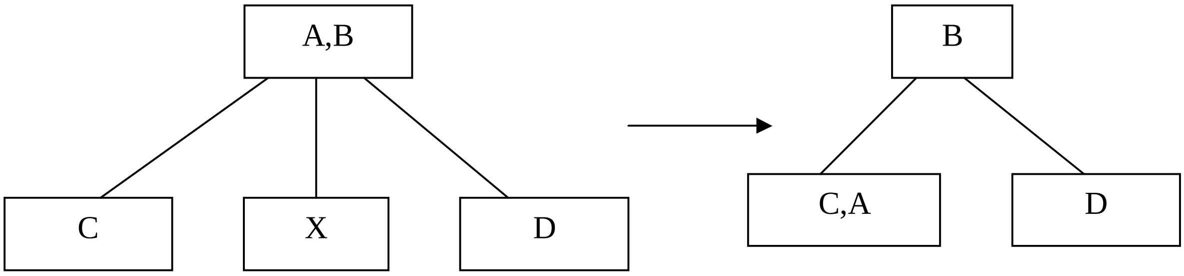


$\Leftarrow$ נמזג את k_2 עם k_3,k_4 ונמחק את k_2 רקורסיבית:

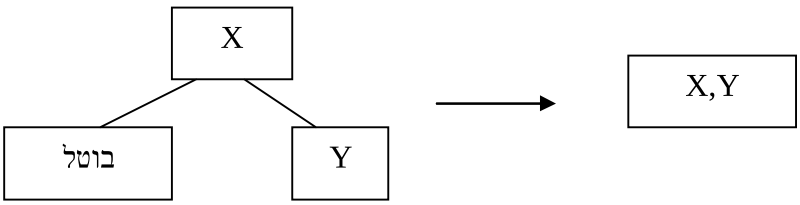


ב. מחיקה מעלה.

- אם בעלה יש יותר ממפתח אחד, נמחק את x.
- אם בעלה יש מפתח אחד:
  1. יש ל x אח צמוד שיש לו יותר ממפתח אחד.
  2. אם להורה של x יש יותר ממפתח אחד:

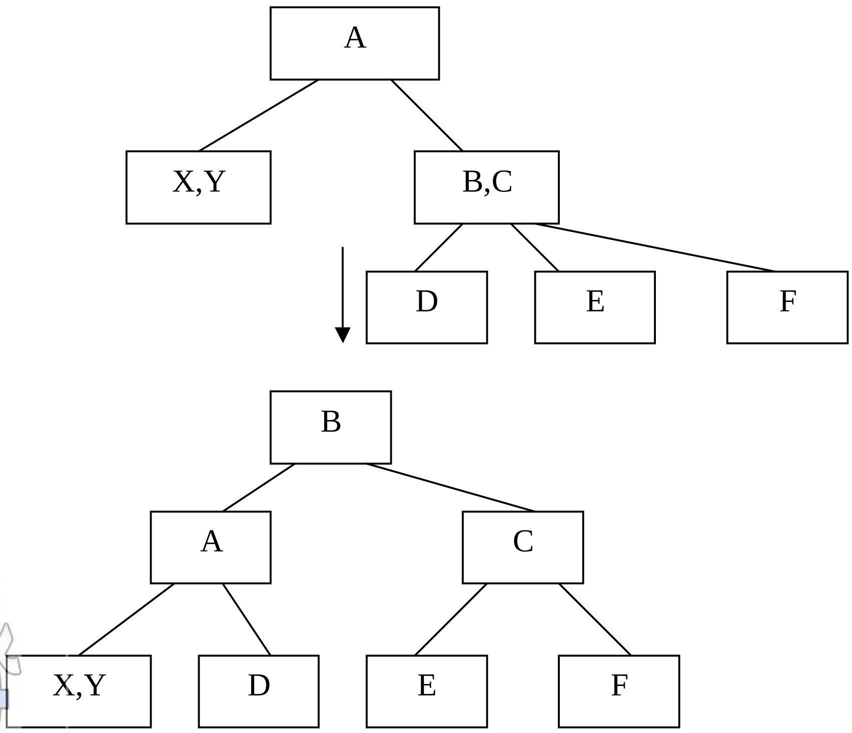


3. גם לאבא וגם לאח יש רק מפתח אחד:

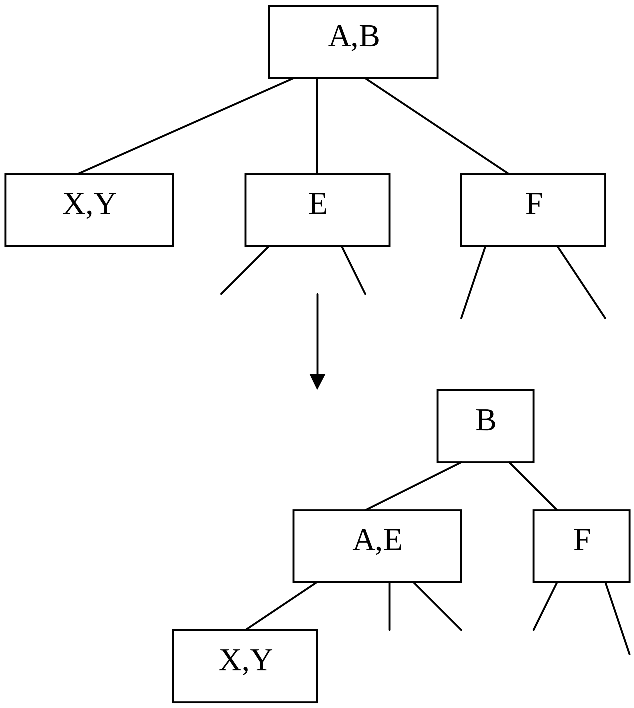


מחברים את האח (היחיד) לאבא. אם זה השורש – סיימנו. אחרת – קיבלנו עלה בעומק שווה מכל העץ ולכן ממשיכים:

- עזרה מאח (לאח של האבא יש יותר ממפתח אחד):



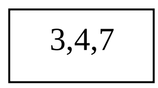
- עזרה מהורה. לאחים של האבא יש רק מפתח אחד, אבל לאב של האבא יש יותר:



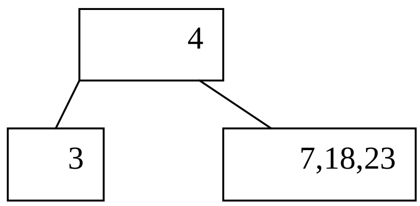
<u>דוגמא לבניית עץ</u>

3,7,4,18,23,5,2,15,14,9

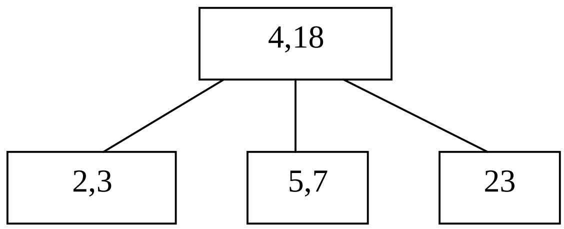
שלב 1:



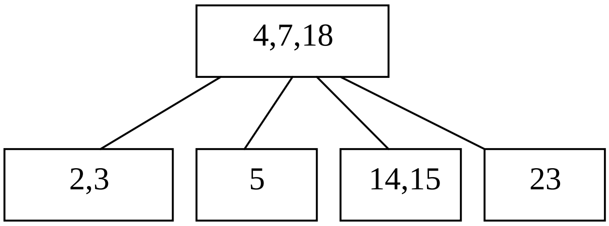
שלב 2:



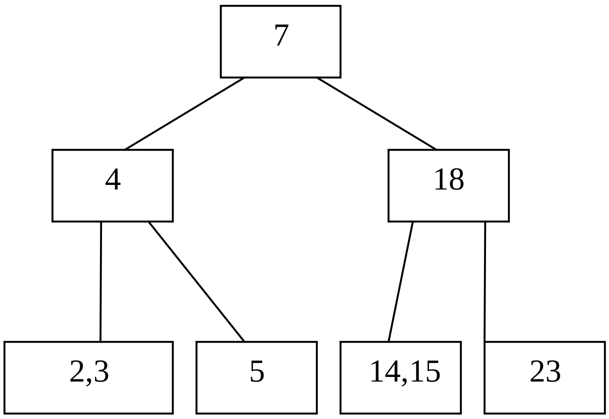
שלב 3:



שלב 4:



שלב 5:

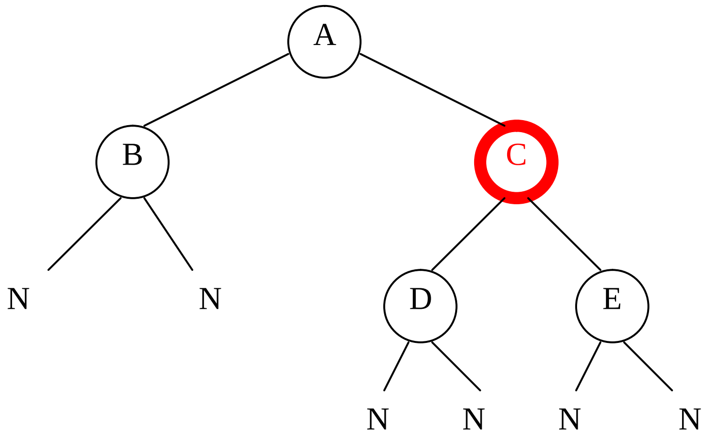


<u>מינימום (של תת עץ שראשו x)</u>

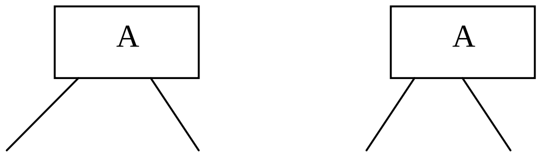
מתחילים מ x. עוברים לבן השמאלי (T_0) שוב ושוב עד שמגיעים לעלה. המפתח השמאלי בעלה הוא המינימום.

<u>עץ אדום שחור</u>

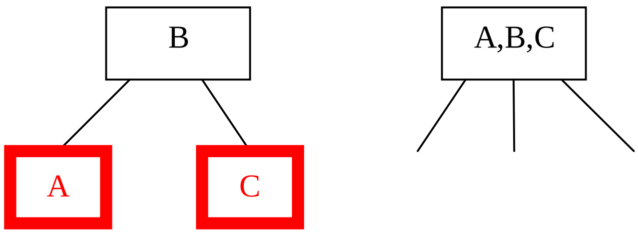
- עץ חיפוש בינארי מאוזן.
- לכל צומת צבע (אדום / שחור).
- השורש – שחור.
- NULL – שחור.
- לצומת אדום שני ילדים שחורים.
- מכל צומת z המסלולים מ z לעלים מכילים מספר זהה של צמתים שחורים.



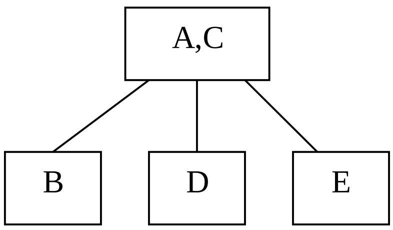
עץ אדום שחור הוא תרגום של עץ 2-3-4



כל צומת של 2-3-4 יתורגם לצומת אחד שחור וכל השאר יהיו אדומים



נתרגם את העץ שבדוגמא הראשונה לעץ 2-3-4:



בתרגום של עץ אדום שחור לעץ 2-3-4, כל העלים בעץ 2-3-4 יהיו באותה רמה.
כשמתרגמים עץ 2-3-4 לעץ אדום שחור, לכל היותר גובה העץ האדום שחור יהיה פי שניים מעץ 2-3-4.
בתוך עץ אדום שחור גובהו של תת עץ של צומת אחד יהיה לכל היותר פי שניים מגובהו של תת עץ אחר של אותו צומת.

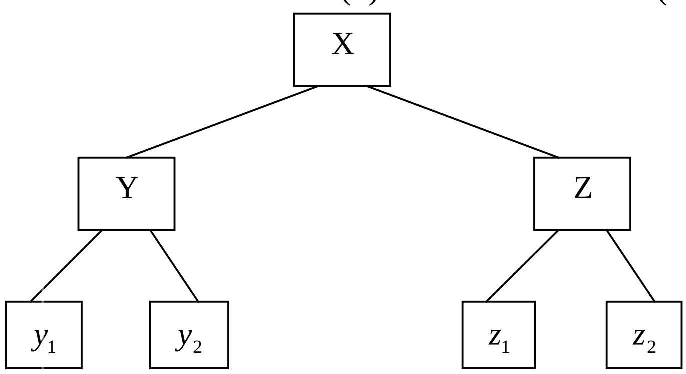
נראה שגובה עץ אדום שחור הוא לכל היותר $2\log(n+1)$. $\frac{h}{2} \le Bh(x) \le h$.

$Bh(x)$ - מספר השחורים במסלולים של תת העץ של x (לא כולל x).

נוכיח באינדוקציה שבתת העץ ששורשו x לפחות $2^{Bh(x)}-1$ מפתחות:

$Bh(x)=0$ - מספר המפתחות הוא $2^0-1=0$.

נניח עבור $Bh(x)=k$ ונוכיח עבור $Bh(x)=k+1$.



טענה: בתת העץ ששורשו z (y) לפחות 2^k-1 מפתחות.
לכן בתת העץ ששורשו x יש לפחות $2(2^k-1)+1=2^{k+1}-1$ מפתחות.

שאלה: מהו $Bh(x)$ הגדול ביותר של עץ המכיל n מפתחות?

$$2^{\frac{h}{2}}-1\le 2^{Bh(x)}\le n$$

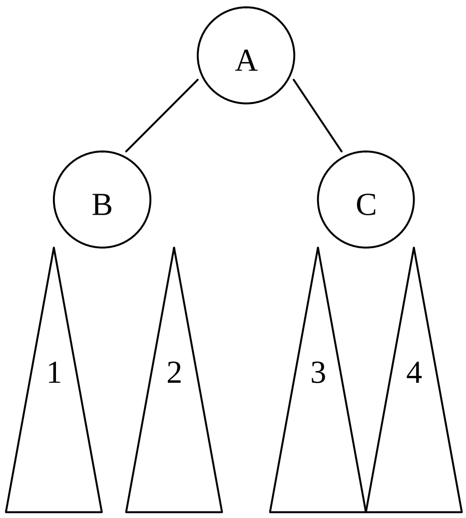
$$2^{\frac{h}{2}}-1\le n$$

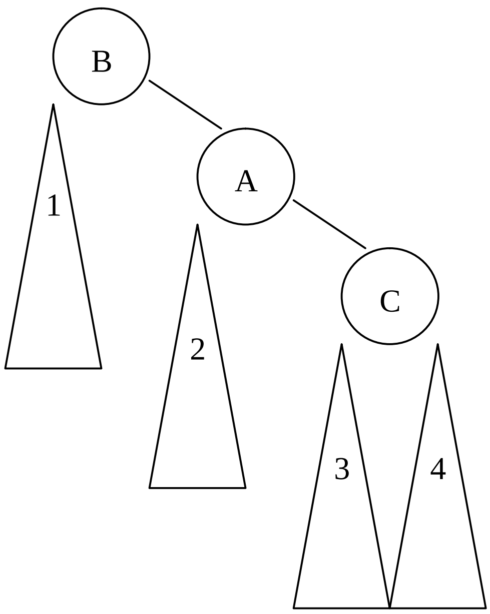
$$2^{\frac{h}{2}}\le n+1$$

$$\frac{h}{2}\le \log(n+1)$$

$$h\le 2\log(n+1)$$

שני עצים זהים:

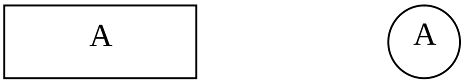




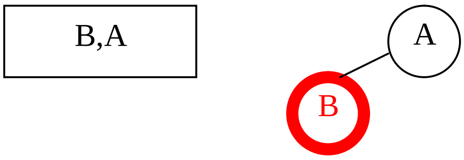
<u>חיפוש</u>: אותו דבר כמו בעצים רגילים – חיפוש עד שמגיעים לעלים.

<u>הוספה</u>:
א. הוסף לפי חוקי עץ חיפוש בינארי – שמים את האיבר במקום הריק.
ב. צבע את הצומת באדום.
ג. תקן את העץ במידת הצורך.

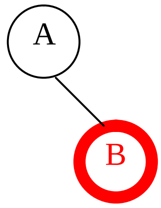
מקרה 1:



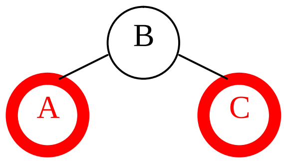
מקרה 2:

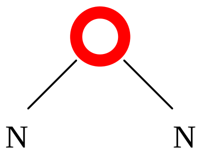


מקרה 3:

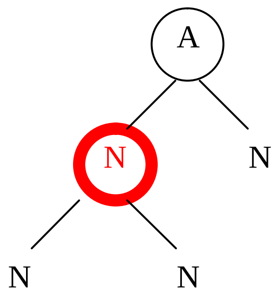


מקרה 4:

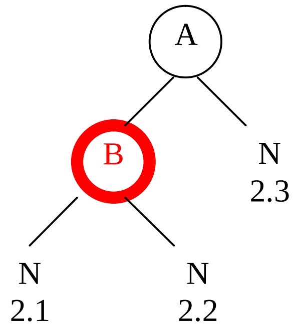




1: אין מה לשנות.



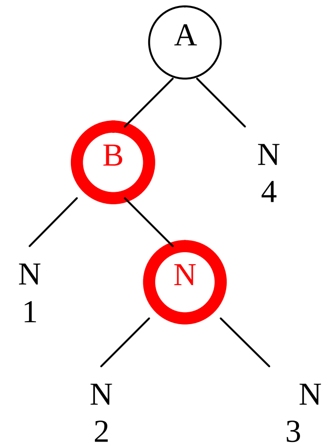
2: נחלק לשלושה מקרים.



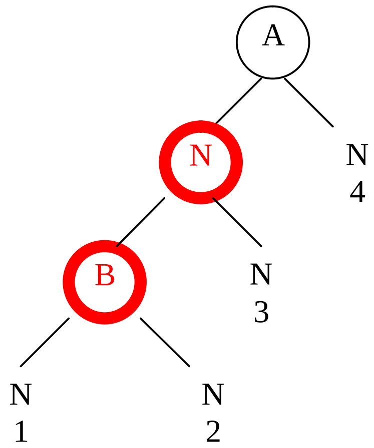
2.3: מוסיפים אדום וזהו.

2.2:

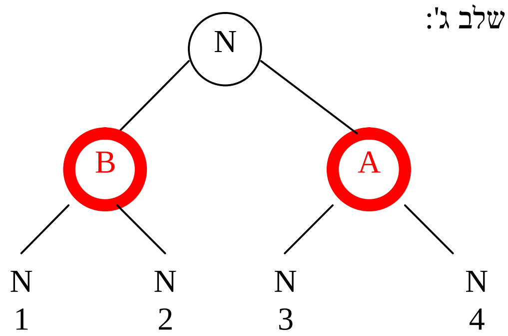
שלב א':



שלב ב':
(מעבירים
למצב 2.1)

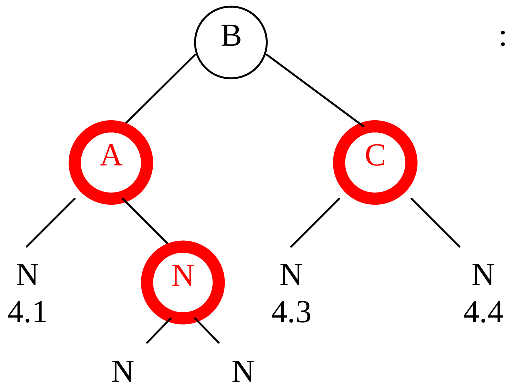


שלב ג':

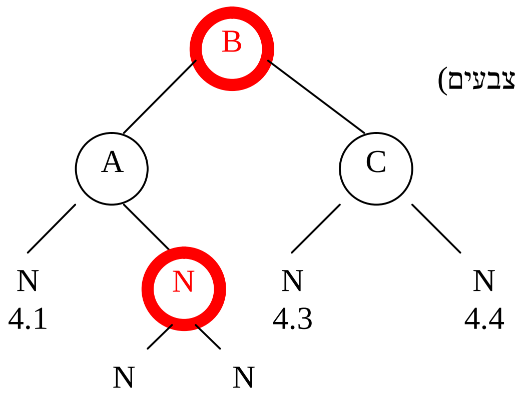


4:

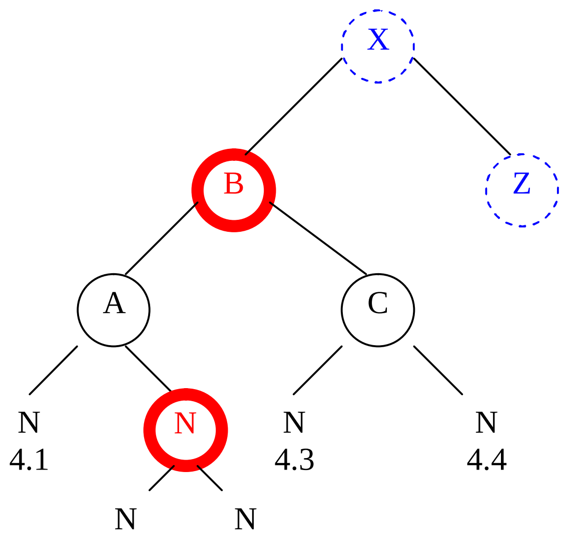
שלב א':



שלב ב':
(החלפת צבעים)



העלינו את B למעלה ע"י צביעתו באדום. כעת נותר לנו X, ההורה שלו.



אם X שחור, אז סיימנו. אם X אדום ישנן שתי אפשרויות:

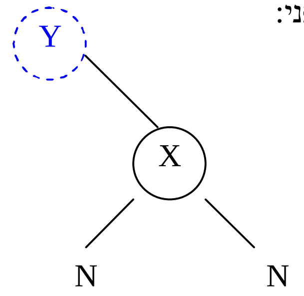
א. Z שחור – מקרה 2.1
ב. Z אדום – מקרה 4.

<u>ביטול:</u>

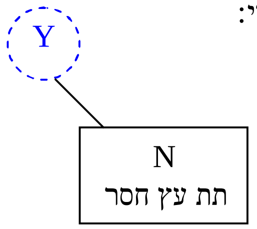
1. בטל לפי חוקי ביטול של עץ חיפוש בינארי.
2. עדכן את העץ (הצומת שבוטל הוא עלה או הורה לבן יחיד).
   א. הצומת שבוטל הוא אדום – סיימנו.
   ב. הצומת שבוטל הוא שחור:

דוגמאות לשתי מחיקות:

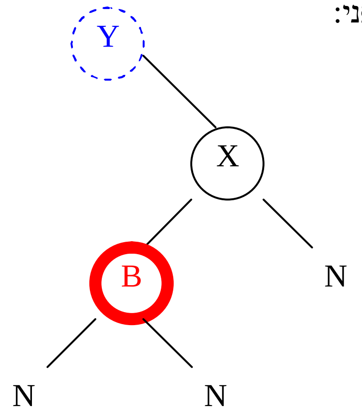
לפני:



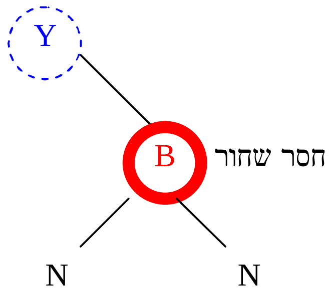
אחרי:



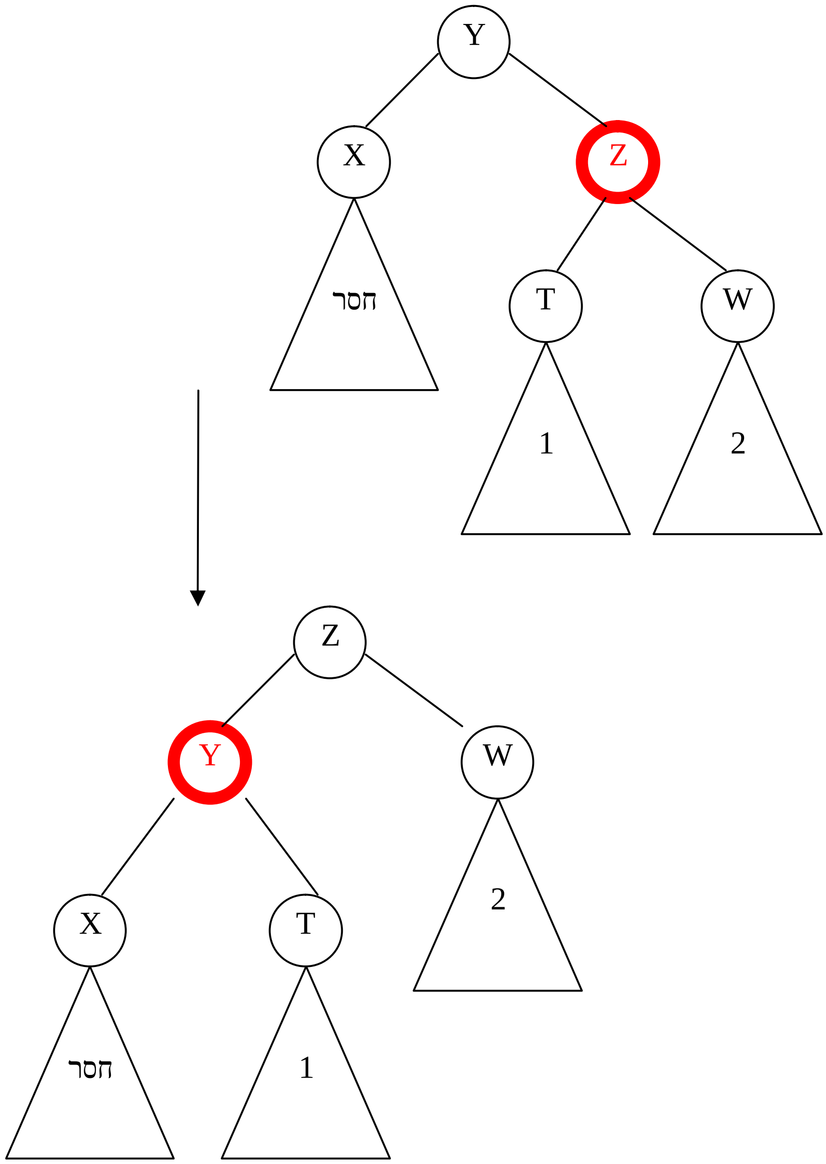
לפני:

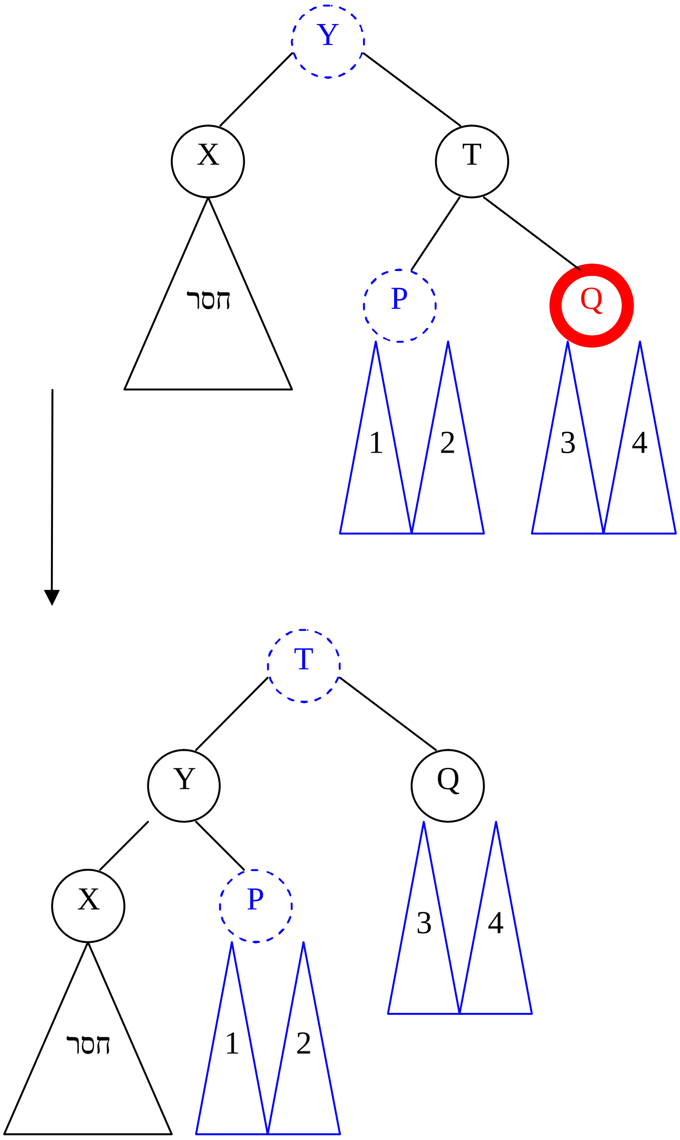


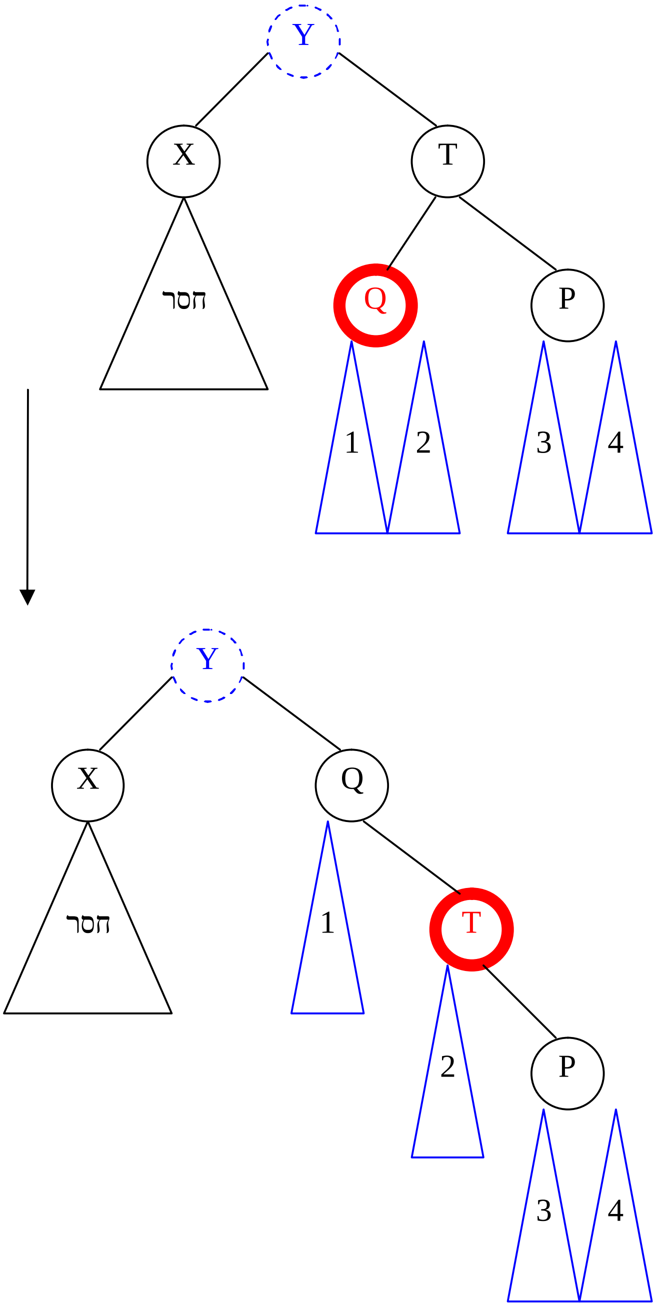
אחרי:

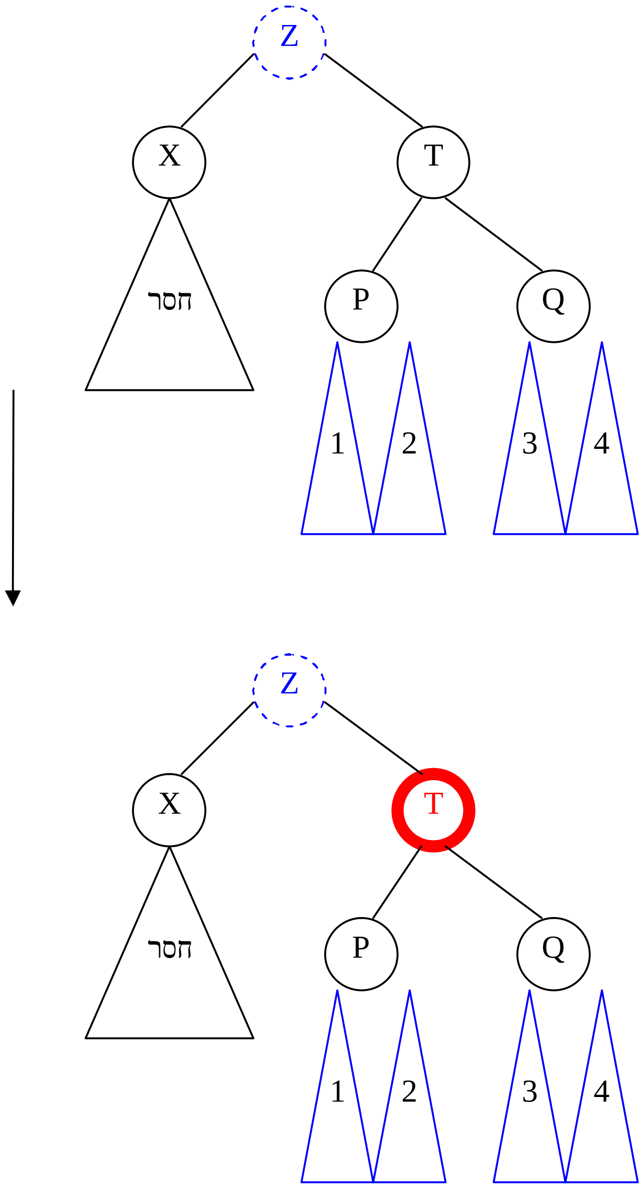


כלל: עץ חסר עם שורש אדום – צובעים את השורש בשחור וגמרנו.
לכן אנו מניחים שהשורש של העץ הוא שחור (אחרת היינו צובעים אותו).


Y
X
Z
חסר
T
W
1
2
Z
Y
W
2
X
T
חסר
1


Y
X
T
חסר
P
Q
1
2
3
4
T
Y
Q
X
P
3
4
חסר
1
2


Y
X
T
חסר
Q
P
1
2
3
4
Y
X
Q
חסר
1
T
2
P
3
4



צבענו את T באדום. עכשיו Z שחור או אדום. אם Z אדום נצבע בשחור וגמרנו. אם Z שחור נעלה עוד רמה.

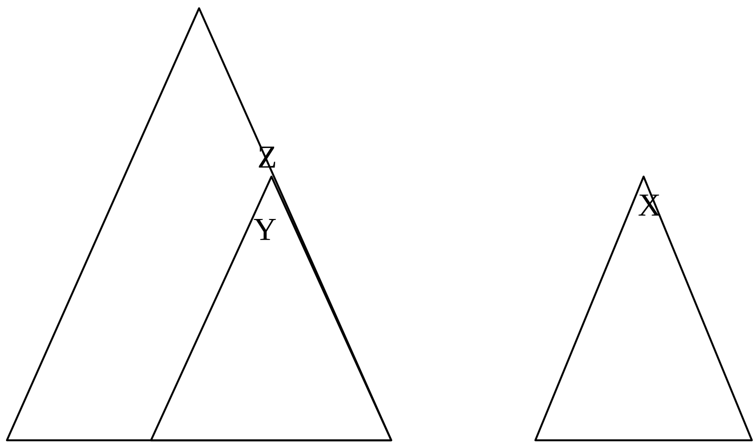
<u>מחיקה</u>

א. בטל לפי כללי עץ חיפוש בינארי.
ב. בוטל אדום – גמרנו.
ג. בוטל שחור – נוצר תת עץ חסר.
   1. שורש אדום – נצבע בשחור וגמרנו.
   2. שורש שחור – תת עץ חסר.
      a. אח אדום – רוטציה.
      b. ניסיון לקבל עזרה מאח.
      c. עזרה מהורה / העברת הבעיה למעלה.

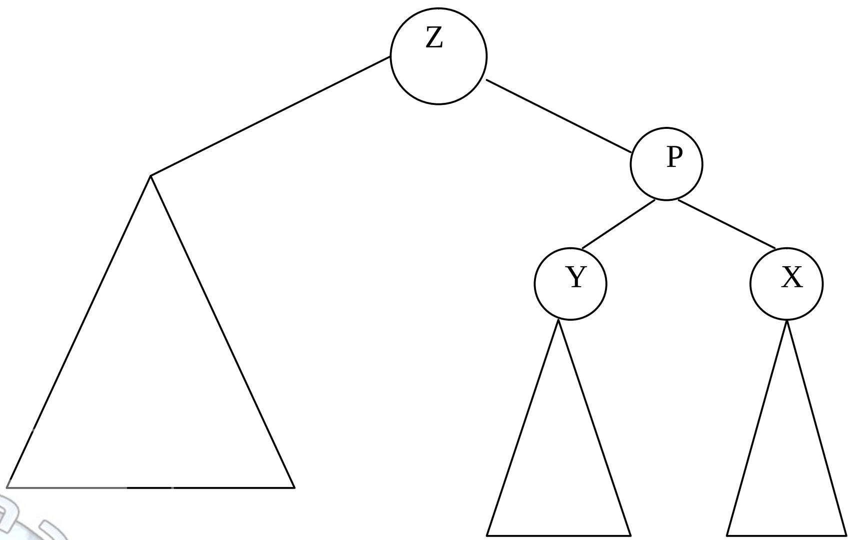
איחוד של עצי אדום שחור

שופכים כל עץ למערך ומקבלים את האיברים ממויינים (ממיינים אותם בעזרת שלב אחד של מיון מיזוג). $O(n)$. לאחר מכן בונים את העץ. האיבר האמצעי יהיה השורש ושני האיברים האמצעיים משני צדדיו יהיו הבנים. אם בשורה התחתונה לא נוכל למלא את כל האיברים, נצבע אותה באדום.

המקרה המעניין הוא כאשר האיברים של עץ אחד קטנים מאלה של העץ השני.
נמדוד את הגובה של העץ הנמוך מביניהם - $Bh(x)$. נחפש את תת העץ השני שהגובה שלו הוא אותו דבר, כלומר $Bh(x)$. נסמן את השורש שלו ב Y.



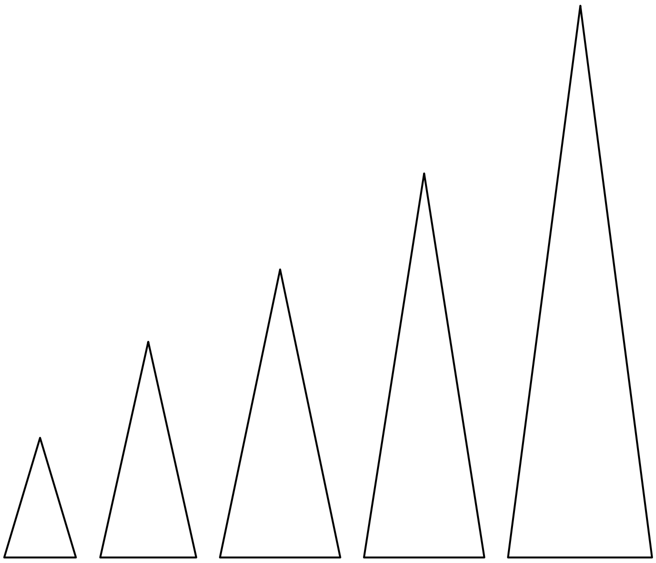
נחפש את ההורה השחור של הצומת שגובהו $Bh(x)$. נסמן אותו ב Z. נאחד את העצים באופן הבא:



סיבוכיות: $O(\log n)$.

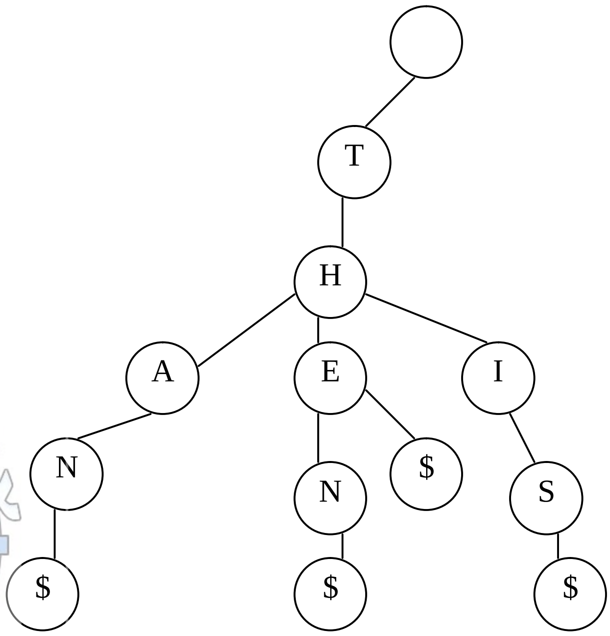
פיצול עץ אדום שחור

עוברים על גובה העץ וגוזמים תתי עצים. כל תת עץ במעלה העץ הגדול הוא גדול יותר מהקודם. גם הערכים של שורשי העצים גדלים במעלה העץ. לאחר שגוזמים את העצים, אנו יודעים מה הגובה של כל אחד מהם. לכן לא צריך למדוד את הגובה של כל אחד מהם בנפרד (מה שלוקח כל פעם $\log n$ באיחוד של שני עצים). נאחד את העצים זה עם זה. עבודה על איחוד כולם ביחד תיקח $\log n$.



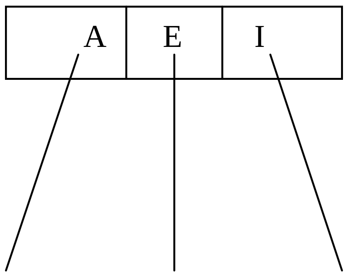
עץ מילים TRIE

THEN
THE
THIS
THAN

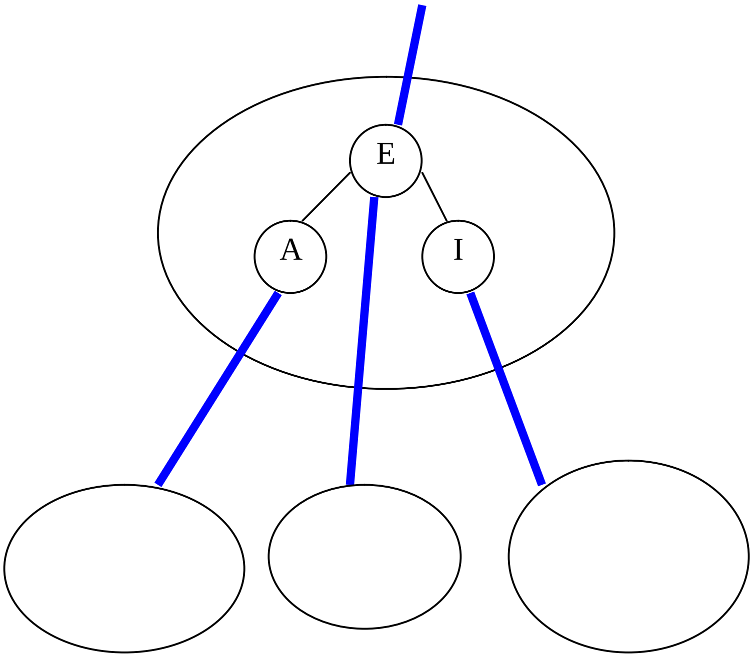


שיטות אפשריות לאכסן אותיות בצמתים:

1. מערך עם רשימה מקושרת לכל אות.



2. עץ אדום שחור. בכל צומת יהיה לנו עץ אדום שחור.



חיפוש, הוסף, בטל: אורך המילה כפול $\log\Sigma$

Σ הוא מספר האותיות ב א"ב.

$T = t_1..t_n$, אלף בית בגודל 20.

המילון מכיל את כל הסייפות של המחרוזת T.

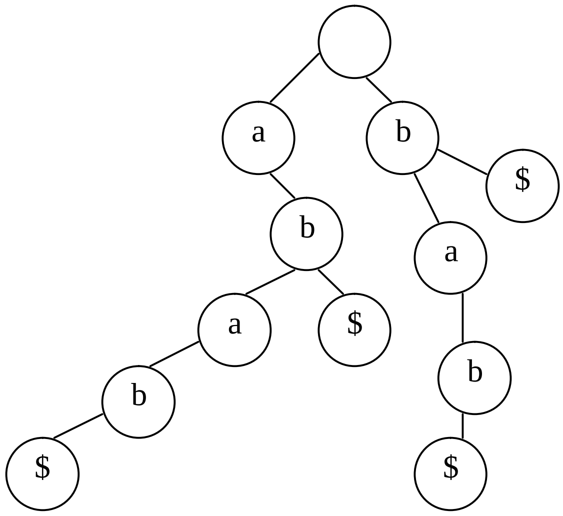
$$T = t_1 t_2 t_3 t_4 \$ = abab\$$$

$t_4\$$

$t_3 t_4\$$

$t_2 t_3 t_4\$$

$t_1 t_2 t_3 t_4\$$



תכנות דינאמי

<u>הרעיון הכללי</u>
פותרים כל תת בעיה פעם אחת בלבד ושומרים בטבלה. תכנון מלמטה למעלה (מהבעיות הכי קטנות ועד לבעיה המקורית), כאשר פתרון של תת בעיה כלשהי מסתמך באופן רקורסיבי על פתרונות לתת בעיות קטנות יותר.

<u>דרך הפתרון</u>
צריך להגדיר את הפתרון באופן רקורסיבי, כלומר להראות כיצד פתרון הבעיה תלוי בפתרון בעיות קטנות יותר.

דוגמא מהספורט: ניצחון ב 4 מתוך 7 משחקים.
תוצאות אפשריות: 4:0, 4:1, 4:2, 4:3.
i - מספר המשחקים שעלינו לנצח.
j - מספר המשחקים שעל הרעים לנצח.

תנאי עצירה ברקורסיה:
$P(i,j)=1 \qquad i=0$
$P(i,j)=0 \qquad j=0$

לא יתכן $i=j=0$

נוסחת הרקורסיה: $P(i,j)=\dfrac{P(i-1,j)+P(1,j-1)}{2}$
הבעיה היא ששיטה זו בזבזנית בחישובים.

חישוב באמצעות טבלה:

i / j	0	1	2	3	4
0	X	1	1	1	1
1	0	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{7}{8}$	$\frac{15}{16}$
2	0	$\frac{1}{4}$	$\frac{1}{2}$		
3	0			$\frac{1}{2}$	
4	0				$\frac{1}{2}$

השלב הראשון הוא לראות שזוהי בעיה שניתנת לפתרון באמצעות רקורסיה.
השלב השני הוא לראות שהפתרון בזבזני.
השלב השלישי הוא להעביר את הפתרון לתכנות דינאמי באמצעות טבלה, שאינו בזבזני.

<u>אלגוריתם חמדן</u>

אלגוריתם שפועל מבלי לראות את כל התמונה.

<u>אלגוריתם של נתינת עודף</u>

סוגי מטבעות: 1,5,10,25.
המטרה: לתת כמה שפחות מטבעות.

לדוגמא: 63: $3\cdot 1$, $1\cdot 10$, $2\cdot 25$.

נניח שסוגי המטבעות הם: 1,5,11.
עבור 15 אי אפשר לבנות אלגוריתם חמדני. האלגוריתם החמדן יתן לנו $4\cdot 1$, $1\cdot 11$. האלגוריתם הלא חמדן יתן לנו $3\cdot 5$.

<u>תת סדרה משותפת ארוכה ביותר (LCS (Longest Common Subsequence</u>

בהינתן שתי מחרוזות, האם הן דומות?

חיפוש מחרוזת קטנה בתוך מחרוזת גדולה:

A 1____________________________ n

B 1__________________ m

השוואת מחרוזות למחרוזת:

A ____________________________

B ____________________________

אחת האפשרויות לפתרון בעיה זו היא לבדוק כמה אותיות באותו מקום זהות ולהחליט על אחוז ההתאמה שעבורו הן יהיו דומות.

הבעיה היא במקרה שיש אותיות דומות, רק לא באותו מקום:
abcd
axbc

תת מחרוזת: קבוצה חלקית של אותיות מהסדרה ששומרות על הסדר המקורי. ניתן לייצג כל מחרוזת באמצעות מספר בינארי: כל אות שנמצאת תהיה 1 וכל אות חסרה תהיה 0.

$$A = a_1, \ldots, a_n$$

$$A = abcdb$$

$$a \;\; c \;\; b$$

10101

לכל מחרוזת בגודל n יש לכל היותר 2^n תתי מחרוזות.

לכל שתי מחרוזות A ו B נמצא את אורך תת המחרוזת המשותפת הארוכה ביותר.

לדוגמא:

$abcd$

$aabb$

תת המחרוזת המשותפת היא ab.

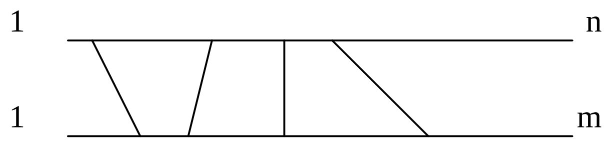
עבור

$abcd$

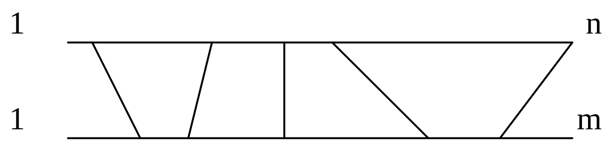
$aacbb$

תת המחרוזות המשותפות הן ac וגם ab.

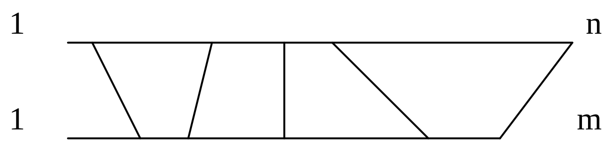
נבנה אלגוריתם שמוצא את תת המחרוזת הגדולה ביותר המשותפת.
נמצא את סט הקווים המקסימלי שמחבר אותיות זהות ולא מצטלב:



נניח שקיבלנו את הפתרון האופטימלי ושתי האותיות האחרונות שוות. אם אין קו ביניהן, אז לפחות לאחת מהן יש כבר קו.



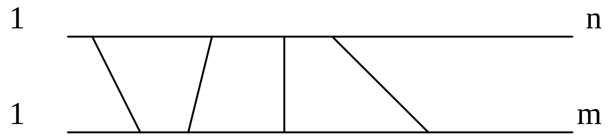
אם נזיז את הקו, לא יהיה נזק:



לכן ניתן להניח ששני האחרונים שווים. זאת משום שאם לא יהיה קו ביניהן אז נזיז את הקו.

לכן בפתרון הרקורסיבי נמחק את שתי האותיות השוות שקו נמתח ביניהן ונוסיף 1 למונה.

מה קורה כשהאותיות האחרונות אינן שוות?



יכול להיות שאין להן קוים, לאחד יש קו או לשני יש קו.
במקרה זה נפעיל את הרקורסיה פעמיים: פעם אחת בלי האות האחרונה במחרוזת הראשונה ופעם אחת בלי האות האחרונה במחרוזת השניה. תנאי העצירה יהיה כשאחת המחרוזות תהיה ריקה.

נקרא לפונקציה LCS: Longest Common Subsequence

1. אם $a_n = b_m$: $lcs(1...n,1...m-1)+1$.
אם $a_n \neq b_m$:
2. $LCS(1...n-1;1...m)$
3. $LCS(1...n-1;1...m-1)$
תנאי עצירה: אם $n=0$ או $m=0$, $LCS()=0$.

נייצג את תתי המחרוזות המשותפות של החלקים השונים בעזרת טבלה:

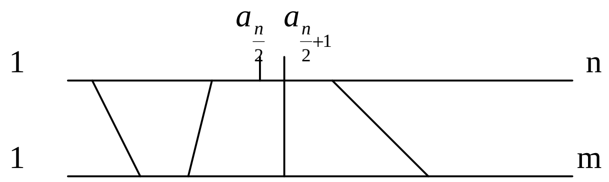
0	b_1						b_m
a_1							
a_m							אורך תת המחרוזת המשותפת המקסימלית

זמן ריצה: $O(m \cdot n)$.

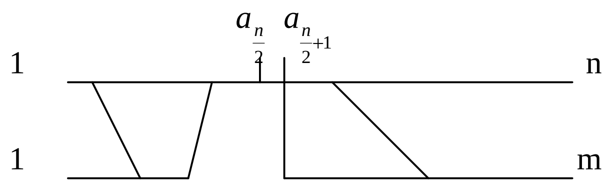
המקום בזיכרון שאנו משתמשים בו בטבלה הוא $n \cdot m$.

ניתן לעשות זאת בצורה יותר יעילה: עבור חישוב כל שורה צריכים רק את השורה שמעליה. לכן אם m ו n אינם שווים, ניקח את השורה באורך הקטן יותר. הבעיה היא שכך נאבד את היכולת לחזור ולמצוא את תת המחרוזת עצמה שהיא באורך שמצאנו.

ישנו פתרון לבעיה זו - נסתכל על הפתרון האופטימלי:



חילקנו אותו לשני חלקים. החלוקה יוצרת מצב בו ה LCS האופטימלי הוא ה LCS של צד אחד + ה LCS של הצד השני. נוריד את הנקודות שלא מחוברות בין שני החלקים במחרוזת השניה:



מחשבים מימין ומשמאל את ה LCS ומאחדים את שתי התוצאות.
כדי לחשב את התוצאות מחלקים כל חלק שוב לשניים (בדומה לדוגמא שראינו בשיעור הראשון).
סיבוכיות הזמן נשארה $O(n \cdot m)$, אבל סיבוכיות המקום היא $O(n)$.

<u>הסבר נוסף לפתרון בעיית LCS</u>

רישא X_j של X : $X_1,...,X_j$

נגדיר תת בעיה של LCS: מצא LCS עבור זוג רישות של X ו Y.

<u>2 שלבים</u>:
1. מצא אורך LCS.
2. מצא LCS

נגדיר טבלה C כך ש:
אורך ה LCS של Y,X $\rightarrow C[i,j]$
* $C[m,n] \leftarrow$ אורך ה LCS של X ו Y.

$$C[i,j] = \begin{cases} C[i-1,j-1]+1 & if \;\; X[i] == Y[j] \\ else \;\; \max(C[i,j-1], C[i-1,j]) \end{cases}$$

תנאי עצירה:

$C[0,0] = 0$

$C[0,j] = 0$

$C[i,0] = 0$

X_0, Y_0 הן מחרוזות ריקות.

כאשר פותרים את $LCS[i,j]$ צריך להסתכל על 2 מקרים.
1. $X[i] == Y[j] \leftarrow$ ה LCS של X_i, Y_j בהכרח גדול ב 1 מה LCS X_{i-1}, Y_{j-1}.
2. אחרת $\leftarrow$ ניקח LCS ארוך יותר מבין האופציה של השמטת $X[i]$ ובין השמטת $Y[j]$.

```
LCS(X,Y)
      m=length(X);
      n=length(Y)
      for i=1 to m C[i,0]=0 // X0 מקרה מיוחד
      for j=1 to n C[0,j]=0 // Y0 מקרה מיוחד
      for i=1 to m
            for j=1 to n
                  if (X[i]==Y[j])->C[i,j]=C[i-1,j-1]+1
                  else C[i,j]=max(C[i-1,j],C[i,j-1])
```

<u>דוגמא</u>:

$X = ABCB$

$Y = BDCAB$

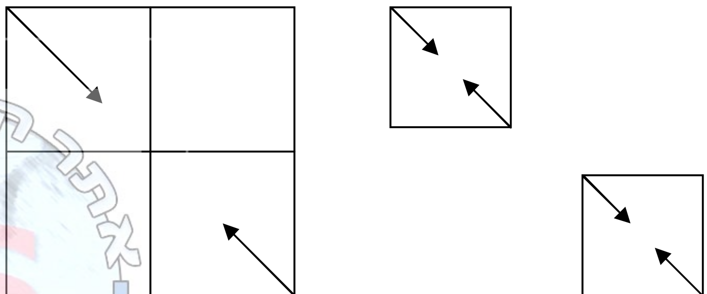
			B	D	C	A	B
		0	1	2	3	4	5
	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1
B	2	0	1	1	1	1	2
C	3	0	1	1	2	2	2
B	4	0	1	1	2	2	3

<u>שחזור המחרוזת</u>:

משווים בין שתי האותיות האחרונות בתת המחרוזת והולכים בטבלה לפי ההשוואה. החיצים האלכסוניים מייצגים את האותיות השוות.

			B	D	C	A	B
		0	1	2	3	4	5
	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1
B	2	0	1	1	1	1	2
C	3	0	1	1	2	2	2
B	4	0	1	1	2	2	3

<u>מציאת LCS והמסלול בסיבוכיות מקום של $O(n)$</u>



מחפשים את ההתאמה האמצעית, ע"י מעבר בו זמנית משני הצדדים, עד שמוצאים את האמצע. לאחר מכן עושים זאת שוב בשני החלקים. סיבוכיות הזמן תהיה $O(n^2)$ וסיבוכיות המקום תהיה $O(n)$ (ע"י שימוש בשיטה המקוצרת של שמירת 2 שורות בלבד בכל חיפוש).

<u>בעיית התרמיל (גרסת 0-1)</u>

תרמיל יכול לסחוב לכל היותר W ק"ג.
S – קבוצת כל n האלמנטים.
עבור כל אלמנט i, משקלו w_i ותועלתו b_i.
w_i, b_i, W מספרים שלמים אי שליליים.

<u>הבעיה</u>:
איזו תת קבוצה של S להכניס לתרמיל כך ש $W \geq \sum w_i$ ונמקסם את $\sum b_i$ מקסימלי.

<u>פתרון נאיבי</u>
2^n קומבינציות אפשריות לפתרון. נעבור על כולן ונמצא את הקומבינציה הטובה ביותר. $O(2^n)$ - אקספוננציאלי.

<u>ניסיון ראשון לפתרון בעזרת תכנון דינאמי</u>
נגדיר $S_k = \{1,...,k\}$.
פתרון עבור S_k הוא תת קבוצה של S_k שנותנת פתרון אופטימלי.

<u>השאלה</u>: האם ניתן להגדיר פתרון ל S_n בעזרת פתרונות לתתי בעיות S_k?
תשובה: לא.

דוגמא:

תועלת	משקל	אלמנט #
3	2	1
4	3	2
5	4	3
8	5	4
10	9	5

$W = 20$

$S_4 = \{1,2,3,4\}$
$\sum w_i = 14$
$\sum b_i = 20$
$S_5 = \{1,3,4,5\}$
$\sum w_i = 20$
$\sum b_i = 26$

S_4 היא לא תת קבוצה של S_5. ל S_5 יש פתרון יותר טוב מזה שמסתמך על הוספת איבר לקבוצה S_4 הקיימת.

<u>ניסיון שני</u>

W= המשקל הכולל המדוייק עבור כל S_k.

נגדיר טבלה B : $B[k,w]$ = סכום התועלות המקסימלי.

k - גודל תת הבעיה אותה פותרים (עבור S_k).

w - המשקל המדוייק עבור S_k.

בלי k הנוכחי

$$B[k,w]=\begin{cases} B[k-1,w] & w_k > w \\ else\ \max\begin{cases} B[k-1,w] \\ B[k-1,w-w_k]+b_k \end{cases} \end{cases}$$

עם k הנוכחי

כלומר: תת הסדרה של S_k הנותנת את התועלת המקסימלית תחת סכום משקלים w היא:

א. לא נכלול את האלמנט k כי $w_k > w$.

ב. נבדוק האם כדאי לכלול את האלמנט ה k:

- כדאי אם הפתרון האופטימלי עבור S_{k-1} שלו משקל כולל $w-w_k$ + הוספת האלמנט ה k.
- לא כדאי – יש פתרון טוב יותר ב S_{k-1} עבור w.

<u>דוגמא:</u>

$n=4$

$W=5$

האלמנטים:

(w_i, b_i)

(2,3)

(3,4)

(4,5)

(5,6)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4		
3	0					
4	0					

S_2 :	S_1 :
$i=2$	$i=1$
$b_i=4$	$b_i=3$
$w_i=3$	$w_i=2$
$w=1$	$w=1$
$w-w_i=-2$	$w-w_i=-1$
$B[2,1]=0$	$B[1,1]=0$
$i=2$	$i=1$
$b_i=4$	$b_i=3$
$w_i=3$	$w_i=2$
$w=2$	$w=2$
$w-w_i=-1$	$w-w_i=0$
$B[2,2]=3$	$B[1,2]=3$
$i=2$	$i=1$
$b_i=4$	$b_i=3$
$w_i=3$	$w_i=2$
$w=3$	$w=3$
$w-w_i=0$	$w-w_i=1$
$B[2,3]=3$	$B[1,2]=3$

סיבוכיות: $O(n\cdot W)$.

מימוש גרפים

$G(V,E)$

על הצמתים יש תויות ולפעמים גם על הקשתות. ישנם גרפים מכוונים וגרפים לא מכוונים.

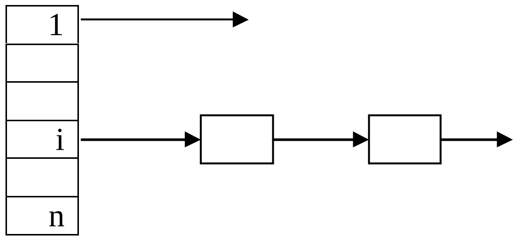
<u>שיטה 1 – בניית טבלה.</u>

נניח שיש n צמתים ו m קשתות. גודל הטבלה יהיה n^2 (גודל הטבלה תלוי בצמתים).

1					j				n
i					X				
n									

הבעיה היא שאם אנו רוצים להודיע הודעה לכל השכנים אנו צריכים לעבור על הרבה תאים.

<u>שיטה 2 – מערך עם מצביעים.</u>



עכשיו המקום בזיכרון הוא $O(n+m)$.

הבעיה היא שסיבוכיות מציאת הקשת היא $O(m)$. היתרון הוא שמסירת הודעה לכל השכנים היא מהירה יותר.

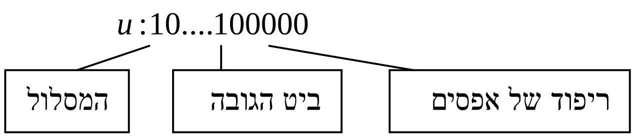
<u>מציאת האב הקדמון הנמוך ביותר LCA</u>

נתון עץ - יש לענות על השאילתה הבאה: בהינתן 2 צמתים בעץ, מצא את האב הקדמון הנמוך ביותר של שני הצמתים.

<u>מציאת LCA של עץ בינארי שלם ב $O(1)$</u> (לאחר pre processing של $O(n)$)

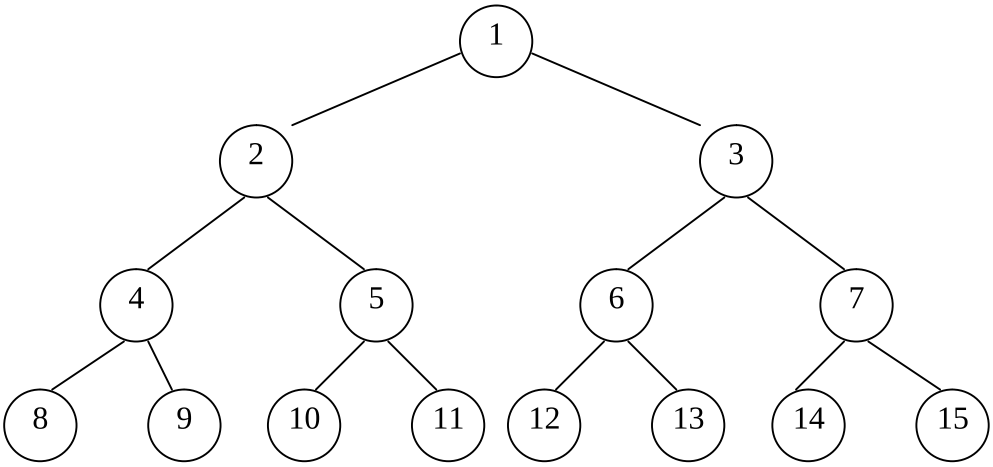
הנחות חשובות
1. אם n שווה למספר הספרות בייצוג בינארי כלשהו, אזי פעולות אריתמטיות, לוגיות והשוואתיות על מספרים באורך של לא יותר מ $\log n$ ביטים ניתן לבצע ב $O(1)$.
2. מציאת הביט השמאלי (הימני) שהוא '1' במספר באורך $\log n$ ביטים ניתן לבצע ב $O(1)$.

קידוד
כל צומת u יקבל קידוד באורך $\log n$ ביטים שיבטא באופן ייחודי את המסלול מהשורש עד ל u.
- התחל בביט השמאלי. הביט ה i מייצג את כיוון הקשת ה iית על המסלול מהשורש ל u. נסמן '0' אם הקשת היא שמאלה, '1' אם הקשת היא ימינה.
- אם במסלול ל u יש k קשתות, אזי הביט ה $k+1$ יהיה '1', ביט הגובה.
- שאר הביטים המשלימים עד ל $\log n$ ביטים יהיו '0'.



נקודד עץ בינארי שלם ב $O(n)$.

דוגמא:



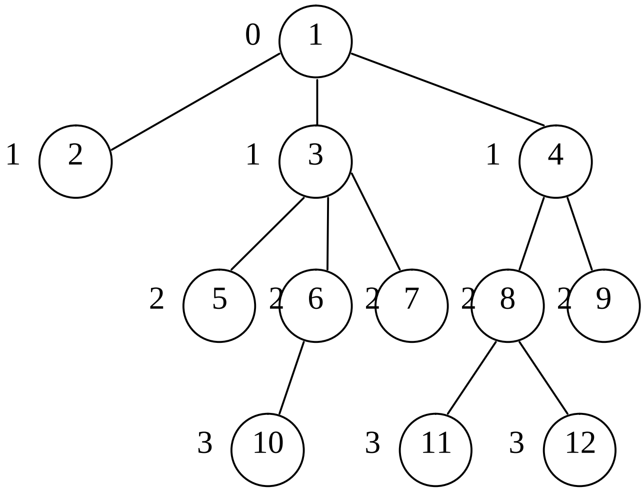
נמצא את האב המשותף של 12 ו 14.

12 1100

14 1110

האב המשותף הוא 3 וזוהי הרישא המשותפת הגדולה ביותר של שני המספרים (11).

נחזור לעץ כללי. נרשום ליד כל צומת את העומק שלו.



נרשום את סדר המעבר על האיברים כשעוברים על כל העץ:

1,2,1,3,5,3,6,10,6,3,7,3,1,4,8,11,8,12,8,4,9,4,1

נרשום מתחת לכל צומת את הרמה שלו:

1	2	1	3	5	3	6	10	6	3	7	3	1	4	8	11	8	12	8	4	9	4	1
0	1	0	1	2	1	2	3	2	1	2	1	0	1	2	3	2	3	2	1	2	1	0

לכל איבר יש את הפעם הראשונה שהוא מופיע ואת הפעם האחרונה שהוא מופיע.

B – התחלה.

E – סוף.

$B_i \le E_i$ (הם שווים כאשר האיבר הוא עלה).

לכל שני איברים יש שתי אפשרויות:
א. אף צומת הוא לא אב קדמון של הצומת האחר:
1. $E_i < B_j$
2. $E_j < B_i$

ב. אחד הצמתים הוא האב הקדמון של האחר:
$$B_i < B_j \le E_j < E_i$$
$$B_j < B_i \le E_i < E_j$$

כשנעים מצומת אחד אל הצומת השני התא בעל הרמה הנמוכה ביותר הוא האב הקדמון המשותף.

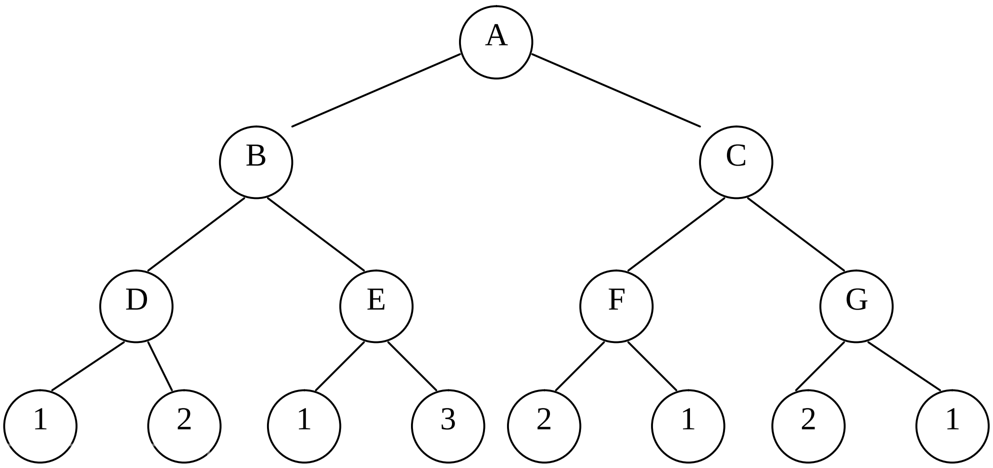
בהינתן שני צמתים עלינו לזהות באיזה מקרה אנחנו.
במקרה ב' אנו יודעים את התשובה.
במקרה א' אנחנו צריכים לדעת אם זוהי אפשרות 1 או 2. אנו סורקים את האזור ובודקים מהי הרמה הנמוכה ביותר בין שני הצמתים.

שלב א': חישוב העומק של כל צומת.
שלב ב': יוצרים שני מערכים: מערך הטיול בעץ ובמקביל מערך העומקים של שני הצמתים.
שלב ג': לכל צומת מחשבים את ההתחלה והסוף שלה במערך.
שלב ד': בהינתן צומת, אנו בודקים באיזה מן המקרים אנחנו נמצאים.

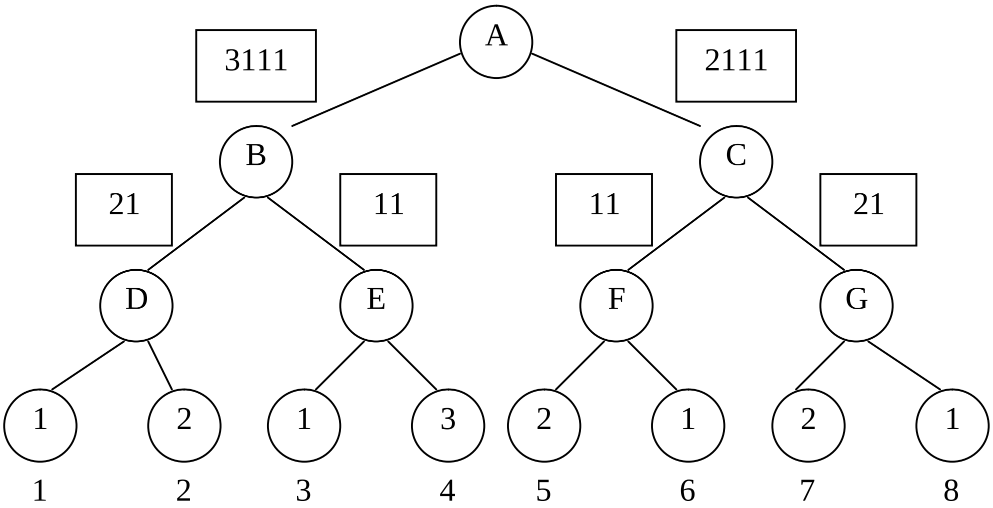
הבעיה היא שהאלגוריתם לוקח $O(n)$.

שיטה לפתור את הבעיה ב $O(1)$ עם הכנה של $O(n\log n)$:

בונים על האינדקסים של הצמתים עץ בינארי מלא. הצמתים מכילים את הרמות:

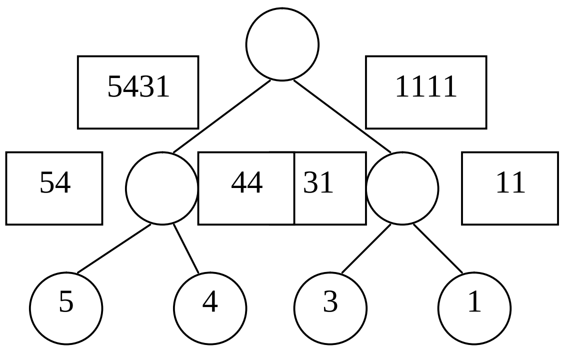


בונים שני מערכים לכל צומת, כל אחד לתת העץ השמאלי או הימני של הצומת. במערך הימני אנו עוברים מימין לשמאל על מערך העלים ובמערך השמאלי משמאל לימין.
גודל המערכים כמספר העלים בכל תת עץ של צומת. אנו ממלאים אותם בצורה הבאה:
מקום i במערך שמאל מכיל את המינימום בין i הילדים הימניים של תת העץ השמאלי של הצומת (וההיפך במערך ימין).



כל איבר במערך הוא המינימום מבין האיברים באותו קטע של המערך משמאל או מימין.

אפשרות אחרת: כל תת עץ מדווח על כל תת המערך:

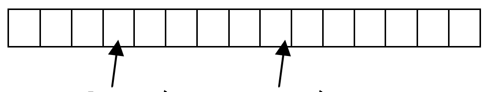


כל צומת סורק את תת העץ שלו וממלא את המערך שלו.

פעולות:

א. מציאת הקצוות לכל צומת $O(n\log n)$. יתכן אפילו ב $O(n)$.
ב. חישוב המערכים עבור כל צומת ומיקום כל עלה במערכים.

ניקח מערך ונחלק אותו לחלקים בגודל $\log n$. נמצא את המינימום בין שני מקומות בו.



ניקח קודם כל קבוצות בגודל $\log n$ ונמצא את המינימום בהן. נסרוק כל קבוצה ונמצא את המינימום בה.

עכשיו יש לנו $\dfrac{n}{\log n}$ קבוצות שיש לנו את המינימום שלהן.

הכנת מבנה הנתונים:

$$O\left(\frac{n}{\log n}\cdot\log\frac{n}{\log n}\right)=O(n)$$

את ערכי המינימום של הקבוצות נשים בעץ כמו שראינו קודם.

עדיין נשארה בעיית הקצוות (חלקים קטנים מ $\log n$ בין שני המקומות שעלינו למצוא את המינימום ביניהם).

נבצע את אותו האלגוריתם בדיוק על החלק בגודל $\log n$. נחלק אותו לחלקים שכל אחד מהם בגודל של $\log\log n$.
נמצא את המינימום בכל אחד מהחלקים בגודל $\log\log n$.
מציאת המינימום בכל החלקים: $O(\log n)$.
סה"כ הכנות בסיבוב השני: $O(n)$.

בסיבוב השלישי נשתמש בשיטה שונה:
גודל הקטע: $\log\log n$.
למערך שלנו יש תכונה מיוחדת. הוא מייצג עומק של צמתים. המספר הבא בתור במערך הוא העומק של הצומת הבא ולכן הוא או גדול ב 1 מהמקום הנוכחי או קטן ב 1 ממנו.
לכן ניתן לרשום את המערך בצורה הבאה:

5	+1	-1	+1	-1	+1	+1						

המינימום של המערך השונה הבא יהיה באותו אינדקס בדיוק:

7	+1	-1	+1	-1	+1	+1						

נבנה מערך דו מימדי בגודל $(\log\log n)^2$ בו בכל תא יהיה האינדקס המינימלי בין כל שני אינדקסים:

j\i	1	2	3	4	5
1					
2					
3					
4					
5					

הטבלה תחושב בזמן של $O((\log\log n)^2)$.

ישנן $2^{\log\log n} = \log n$ סדרות כאלו.
$$\log n(\log\log n)^2 = O(n)$$

נמצא עבור כל קטע כזה את טבלת המינימום השייכת לו. נעשה זאת בעזרת עץ מילים של – ו +. בסוף כל חלק של העץ יהיה מצביע לטבלה המתאימה. לכן הזמן שיקח לנו למצוא את המשפחה עבור כל הקטעים הוא $O(n)$.

<u>הסבר נוסף למציאת LCA(u,r)</u>

נסמן את הקידוד של כל צומת u ב $path(u)$.

$$path(u)\ \ XOR\ \ path(r) = result$$

ריפוד (0), ביט גובה (1), מספר הקשתות הזהות = $path(lca(u,v))$

לדוגמא: $lca(1001,1011)=1001 XOR 1001 = 0010$. נוריד את האפסים בהתחלה, ניקח את החלק מה 1 והלאה, נוסיף 1 ואחריו אפסים: 1010. וזהו הקידוד של האב הקדמון המשותף הנמוך ביותר. אם הסיבית השונה הראשונה היא סיבית הגובה, נשים משמאלה 0 (ואחר כך נוסיף את ה 1 ואת האפסים).

הגדרה: Range – minima problem

<u>בעיה</u>: בהינתן מערך בעל n מספרים ממשיים נרצה להפעיל אלגוריתם preprocessing שיאפשר לענות על השאילתא הבאה ב $O(1)$: בהינתן חלק רצוף של המערך, מיהו הערך המינימלי בו?

<u>מקרה פרטי</u>: ידוע שההפרש בין כל שני איברים עוקבים הוא 1.

<u>הרדוקציה</u>
נראה כי בהנחה שיודעים לפתור את המקרה הפרטי של range minima, ניתן לפתור את בעיית ה LCA.
כדי למצוא $lca(u,v)$ נמצא את ההופעה הראשונה של v,u. כך נגדיר את הטווח למציאת ה range minima.

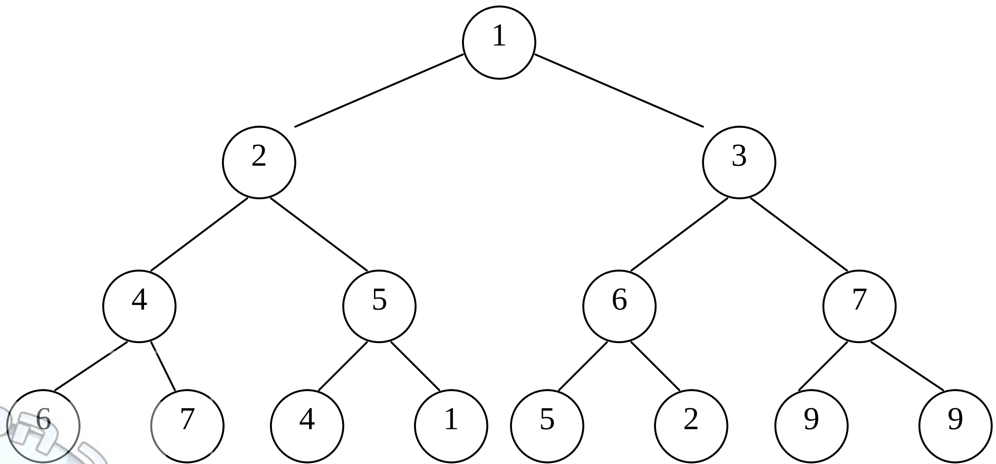
א. אם u הוא אב קדמון של v, אזי כל הצמתים שנבקר בין u ל v הם בתת העץ של u, כלומר לא יהיה בטווח אף צומת שהרמה שלו נמוכה מזו של u.
ב. אחרת: כל הצמתים שנבחר בהם הם בתת העץ של $lca(u,v)$, כולל $lca(u,v)$. חייבים לעבור במסלול אוילר בין ההופעה הראשונה של u וההופעה הראשונה של v.

<u>פתרון למקרה הפרטי של range minima</u>

שתי פרוצדורות ל pre processing:
1. Proc I $O(n\log n)$.
2. Proc II $O(2^n)$.

Proc I
n הוא חזקה של 2. בנה עץ בינארי שלם שעליו הם המספרים במערך A (כלומר $2n-1$ צמתים). שמור וחשב עבור כל צומת פנימי את כל ה prefix minima וכל ה suffix minima שלו.

דוגמא:
$\{6,7,4,1,5,2,9,9\}$



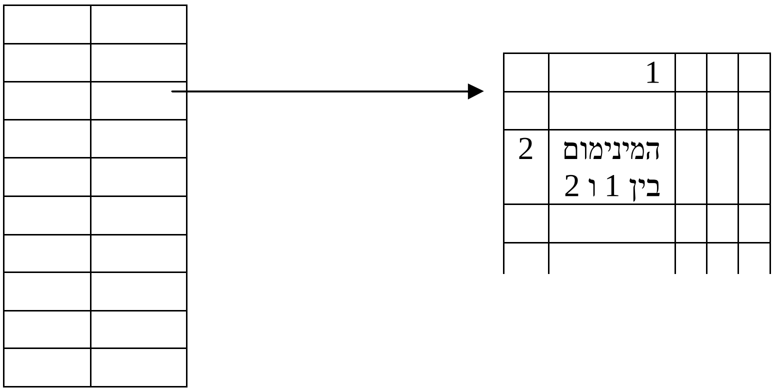
עבור צומת 1:
Prefix minima: $\{6,6,4,1,1,1,1,1\}$
Suffix minima: $\{1,1,1,1,2,2,9,9\}$.

עבור צומת 3:
$Pm = \{5,2,2,2\}$
$Sm = \{2,2,9,9\}$

<u>סיבוכיות:</u>
כל חישוב prefix minima או suffix minima הוא ב $O(1)$.
בכל רמה נחשב ונשמור רשימות בגודל $O(n)$.
מספר רמות (עץ בינארי שלם) $O(\log n)$.
סה"כ: $O(n \log n)$.

<u>Proc II</u> $O(n^2)$

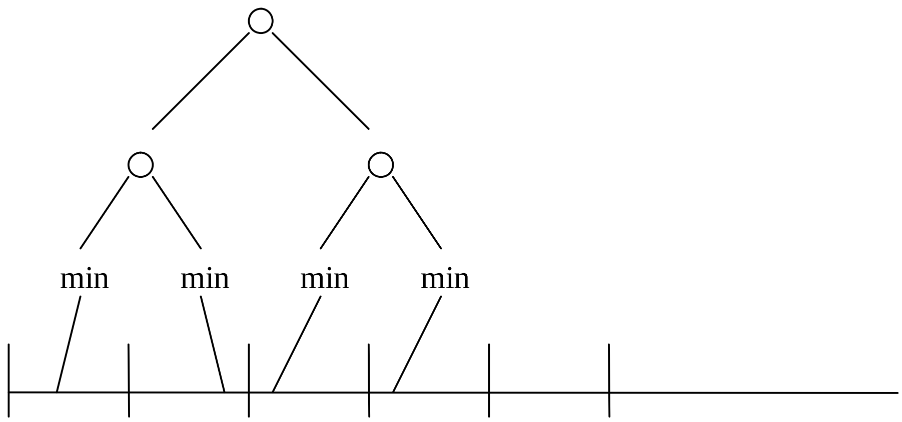
1. נניח שהאיבר הראשון ברשימה הוא 0.
2. נבנה טבלה בעלת 2^{n-1} כניסות (עבור 2^{n-1} רשימות L אפשריות). כל כניסה היא בעצם מצביע לטבלה הקשורה ל L ספציפית.
3. הטבלה עבור L ספציפית מאכסנת את כל התשובות לכל הטווחים הקיימים (קיימים $O(n^2)$ טווחים אפשריים. סה"כ טבלאות: $n^2 \cdot 2^2 = O(2^n)$.



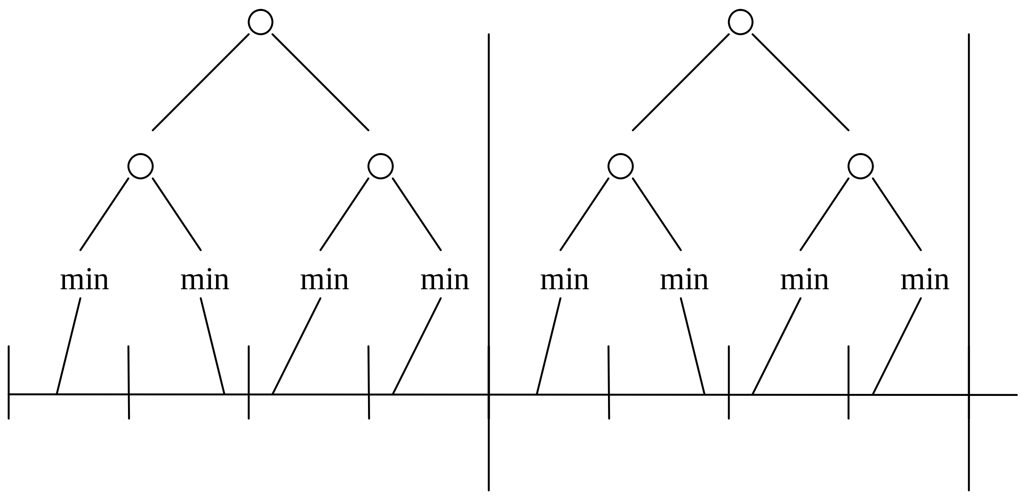
<u>שאילתת restricted range minima</u>
$O(1) \leftarrow$ ניגש לכניסה המתאימה לקלט המתאים.
$O(1) \leftarrow$ ניגש לכניסה בטבלת הטווחים המתאימה לטווח.

<u>בהינתן Proc I, Proc II נראה כיצד לבנות אלגוריתם preprocessing ב $O(n)$</u>
3 שלבים:

1. נחלק את L לקבוצות בגודל $\log n$. לכל קבוצה נמצא מינימום. ישנם $\frac{n}{\log n}$ מינימומים. נפעיל עליהם את Proc I על מערך בגודל $\frac{n}{\log n}$.



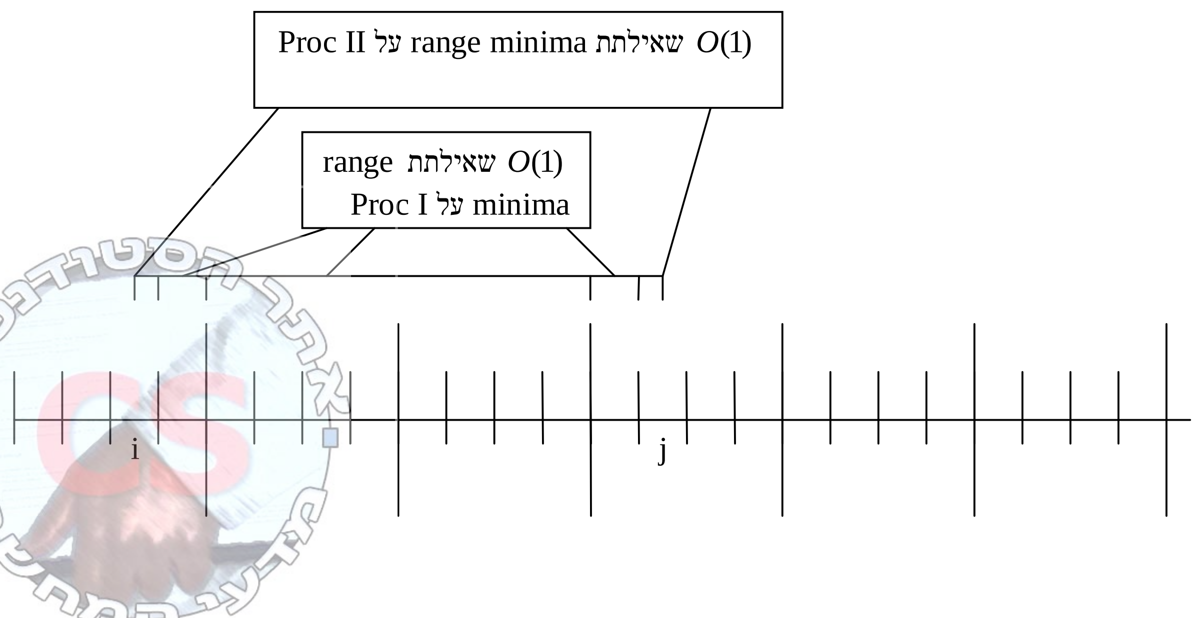
2. כל קבוצה בגודל $\log n$ נחלק למחיצות בגודל $\log \log n$. לכל קבוצה נמצא מינימום. בכל קבוצה ישנם $\frac{\log n}{\log \log n}$ מינימומים. נפעיל עליהם את Proc I על מערך בגודל $\frac{\log n}{\log \log n}$.



3. על כל אחת מהמחיצות בגודל $\log \log n$ נפעיל את Proc II.

<u>שאילתת range minima ב $O(1)$ $[i, j]$</u>

נבטא את הטווח $[i, j]$ ע"י 5 תתי טווחים:



<u>סיבוכיות preprocessing</u>

1.

- חלוקה ל $\frac{n}{\log n}$ קבוצות. מציאת מינימום בכל קבוצה $\leftarrow O(1)$. מקום $O\left(\frac{n}{\log n}\right) \leftarrow$ שמירת המינימומים.
- הפעלת Proc I על מערך בגודל $\frac{n}{\log n}$.

$$O(m\log m) = O\left(\frac{n}{\log n}\log\left(\frac{n}{\log n}\right)\right) \le O\left(\frac{n}{\log n}\log n\right) = O(n)$$

2.

- חלוקת כל קבוצה מתוך $\frac{n}{\log n}$ קבוצות למחיצות בגודל $\log\log n$ ומציאת המינימום $\leftarrow O(n)$. מקום: $\frac{n}{\log\log n}$ מינימומים.
- הפעלת Proc I על $\frac{n}{\log n}$ מערכים בגודל $\frac{\log n}{\log\log n}$.

$$O(m\log m) = \frac{n}{\log n}\cdot O\left(\frac{\log n}{\log\log n}\log\left(\frac{\log n}{\log\log n}\right)\right) \le \frac{n}{\log n}\cdot O\left(\frac{\log n}{\log\log n}\log\log n\right) = O(n)$$

3. $\frac{n}{\log\log n}$ מחיצות. בכל מחיצה $\log\log n$ איברים.

$$2^m = 2^{\log\log n} = \log n\cdot\frac{n}{\log\log n} = O(n)$$

<u>שאלה</u>

נתון מערך A ובו n איברים.
השאילתא: בהינתן שני אינדקסים מצא את האיבר בעל הערך המינימלי.

<u>שיטה 1</u>

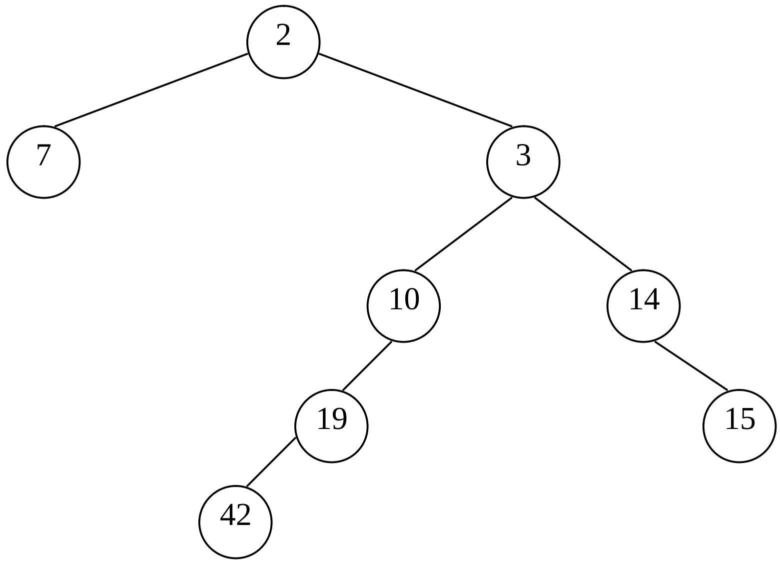
הביצוע יהיה דומה לזה של מציאת אב קדמון משותף. ההבדל הוא שלא נוכל להשתמש בטבלת ה + וה -.
לכן נשתמש בשיטה של העצים עד שנסיים (k שלבים).
הכנה $O(kn)$.

שאילתא $O\left(2k + \underbrace{\log\ldots\log}_{k\ times} n\right) = O(\log^* n)$

<u>שיטה 2 – עצים קרטזים</u>

עצים בינארים.
בהינתן מערך $a_1\ldots a_n$.
שורש העץ a_{root} שווה לערך המינימלי.
הבן השמאלי הוא הערך המינימלי ב $a_1\ldots a_{root-1}$.
הבן הימני הוא הערך המינימלי ב $a_{root}\ldots a_n$.

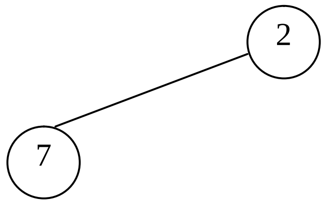
דוגמא:
7,2,42,19,10,3,14,15.



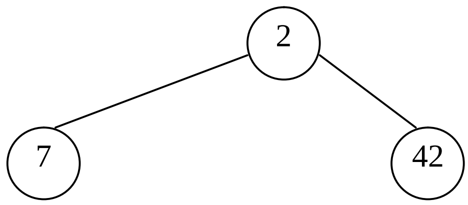
שיטה לחישוב סיבוכיות:
יש לנו מדרגות. אנחנו נמצאים בתחתית גרם המדרגות. יש לנו n שלבים. בכל שלב אפשר לעלות מדרגה אחת או לרדת כמה מדרגות שאנו רוצים. אנחנו משלמים 1 על כל ירידה וכל עליה של כל מדרגה.
סיבוכיות: $O(n)$. אנחנו יכולים לרדת רק את מספר המדרגות שעלינו.

עלינו לבנות עץ בו האב הקדמון המשותף הנמוך ביותר של כל שני איברים יהיה הערך המינימלי ביניהם.

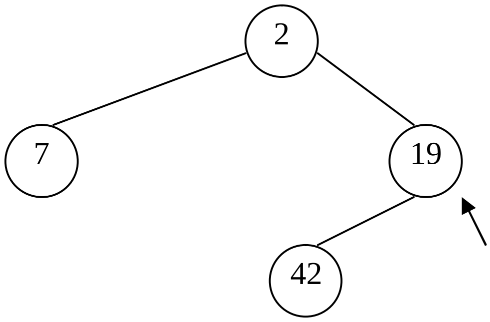
בניית העץ שבדוגמא:
נכניס את 7. לאחר מכן נשווה את 2 ל 7. 2 קטן ממנו ולכן הוא יהיה האב שלו.



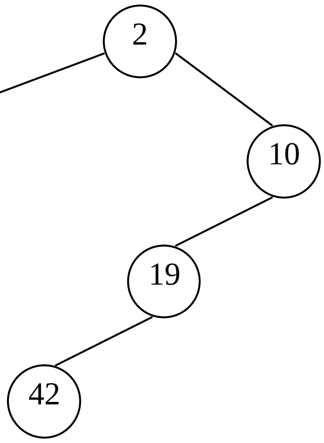
נכניס את 42:



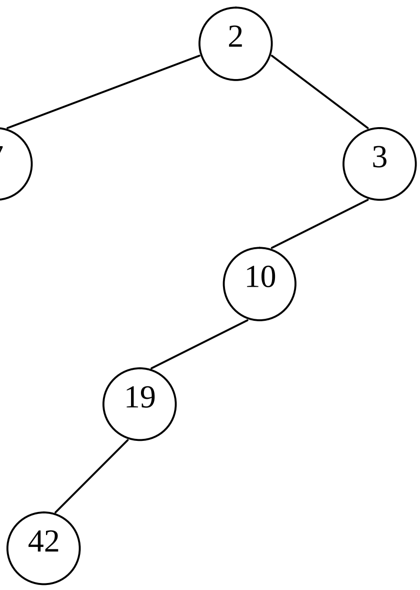
נכניס את 19:



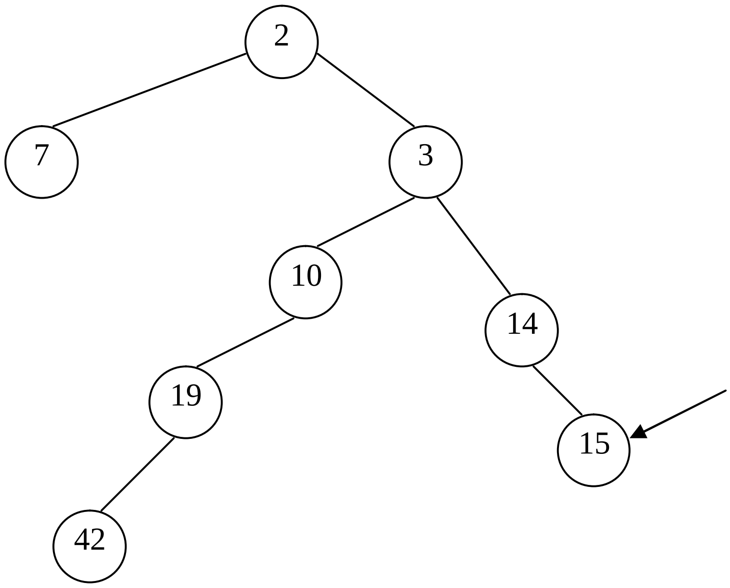
נכניס את 10. נשווה אותו ל 19 ול 2:



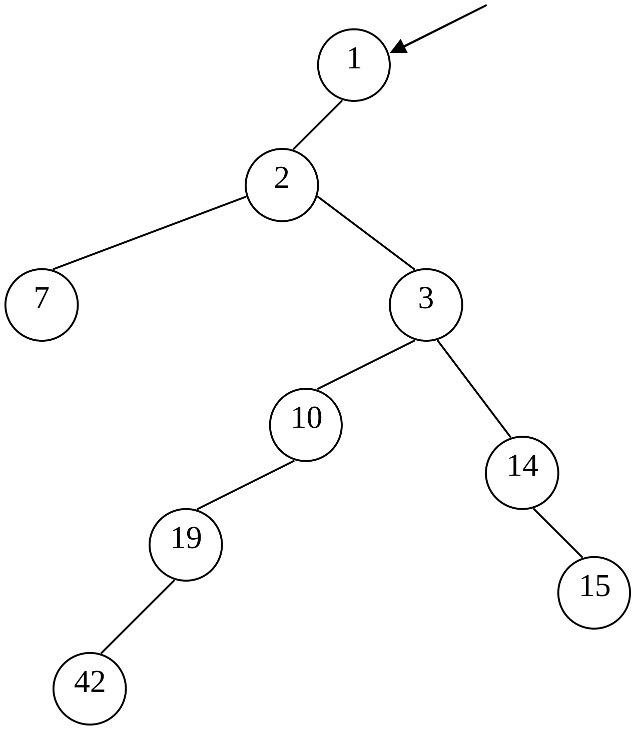
נכניס את 3:



14 ו 15:



נניח שיש גם 1. הוא משווה את עצמו מהמצביע עד השורש והופך לשורש בעצמו. עכשיו המצביע הוא על 1:



שלב 1: בונים עץ קרטזי $O(n)$.

שלב 2: מריצים את הבניה של אב קדמון משותף $O(n)$.

שאילתא: $O(1)$.

שני מקרים:

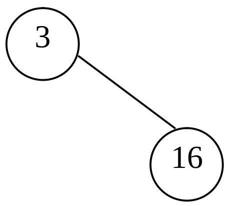
מקרה A – האיבר החדש יותר גדול. בודקים את האיבר שהתווסף לפני האיבר החדש. רואים שהאיבר שלנו יותר גדול ולכן מוסיפים אותו כבן. העומק בו יהיה גדול ב 1.

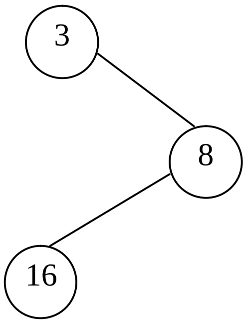
מקרה B – האיבר החדש יותר קטן. העומק של האיבר החדש יותר קטן.

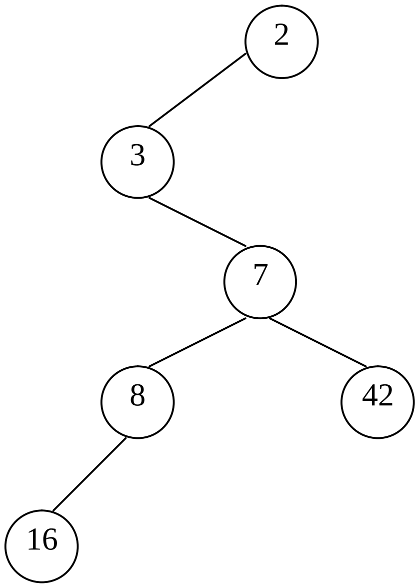
מספר הפעולות שביצענו בכל שלב הוא (הפרש העומקים)O.

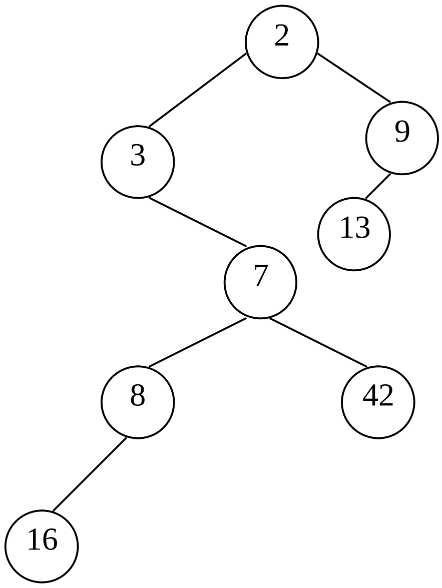
אנו רואים שסיבוכיות האלגוריתם היא הסיבוכיות של אלגוריתם המדרגות. בכל שלב ניתן לעלות דרגה אחת או לרדת כמה דרגות.

דוגמא:
3,16,8,7,42,2,13,9









מספר פעולות בכל הוספה:

1	1	1	1	1	3	1	1
3	16	8	7	42	2	13	9

העומק של כל צומת ברגע שהכנסנו אותו:

1	2	2	2	3	1	2	2
3	16	8	7	42	2	13	9

Hash Tables – טבלאות גיבוב

מבנה נתונים המאפשר פעולות חיפוש, הוצאה והכנסה ב $O(1)$ בממוצע.

מיעון ישיר - כאשר טווח המפתחות האפשריים, d, הוא קטן ואז נוכל לבנות מערך בגודל d כך שכל מפתח i ישב באינדקס i במערך.

כאשר מספר המפתחות המאוחסנים בפועל קטן יחסית לטווח האפשרי $d >> n$, מיעון ישיר מהווה בזבוז מקום. אז נשתמש בטבלת גיבוב שגודלה נמצא ביחס ישיר למספר המפתחות המאוחסנים בפועל. כל מפתח i ישב באינדקס $h(i)$

פונקיית הגיבוב

הפונקציה h:

- זמן החישוב $O(1)$
- טווח הפונקציה הוא האינדקסים במערך. המקור הוא עולם המפתחות.
- דטרמיניסטית – בכל פעם שנפעיל את h על מפתח k תצא אותה תוצאה.
- המיפוי צריך להיות אחיד ככל האפשר (ללא התנגשויות). עבור כל מפתח נקבל תא נפרד במערך.

<u>פתרונות לבעיית המיפוי האחיד</u>

<u>פתרון I: Chaining – שרשור.</u>
כל האיברים שמתמפים לאותו אינדקס ישורשרו לרשימה מקושרת עבור אותו אינדקס במערך.
לדוגמא:
קבוצה בגודל 10
51,17,15,81,92,87,12,55,45,35
$h(k) = k \bmod 10$

0
1 $\rightarrow$ 51 81
2 $\rightarrow$ 92
3
4
5 $\rightarrow$ 15
6
7 $\rightarrow$ 17 87
8
9

<u>הכנסת איבר</u> $O(1)$: הוספה לראש הרשימה המקושרת.
<u>חיפוש איבר</u>: במקרה הגרוע $O(n)$. המקרה הממוצע תלוי באורך הממוצע של הרשימה המקושרת.
<u>הוצאת איבר</u>: במקרה הגרוע $O(n)$. המקרה הממוצע תלוי באורך הממוצע של הרשימה המקושרת.

<u>ניתוח הסיבוכיות של שרשור</u>
הנחת הפיזור האחיד הפשוט: h מפזרת באופן אחיד, כלומר ההסתברות שאיבר נתון ימופה לתא מסויים שווה עבור כל m התאים.
m – גודל הטבלה.
n – מספר המפתחות בשימוש.

a - מקדם העומס = $\frac{n}{m}$: מספר איברים ממוצע ברשימה מקושרת.

משפט
בשיטת השרשור, תחת הנחת הפיזור האחיד הפשוט, חיפוש כושל ממוצע הוא $O(1+a)$.

חישוב פונקציית ה hash

סריקת כל הרשימה המקושרת

<u>משפט</u>
בשיטת השרשור, תחת הנחת הפיזור האחיד הפשוט, חיפוש מוצלח אורך $O(1+a)$.

הוכחה:
- נניח שמפתח נכנס בראש הרשימה.
- נאמר שבזמן החיפוש ישנם n מפתחות שהוכנסו בסדר $k_1..k_n$.
  - מהו זמן חיפוש ממוצע של k_i?

- אחרי מפתח זה נוספו $n-i$ מפתחות נוספים. לכן בממוצע גודל הרשימה משמאל למפתח k_i הוא $\frac{n-i}{m}$. לכן זמן החיפוש הממוצע של k_i הוא $1+\frac{n-i}{m}$.

זמן חיפוש ממוצע למפתח כלשהו:

$$\frac{1}{n}\cdot\sum_{i=1}^{n}\left(1+\frac{n-i}{m}\right)=\frac{1}{n}\cdot\sum_{i=1}^{n}(1)+\frac{1}{n}\cdot\sum_{i=1}^{n}\left(\frac{n-i}{m}\right)=1+\frac{1}{n\cdot m}\sum_{i=1}^{n-1}(i)=1+\frac{n(n-1)}{2}=1+\frac{n-1}{2m}=$$

$$=1+\frac{\frac{n}{m}}{2}-\frac{1}{2m}=1+\frac{a}{2}-\frac{1}{2m}=O(1+a)$$

אנו מניחים ש $n=O(m)$ ולכן $a=O(1)$

מכאן שפעולת חיפוש (והוצאת איבר) לוקחת $O(1)$ בממוצע עבור שיטת השרשור.

<u>פתרון II: Open Addressing (מיעון פתוח)</u>

- כל האיברים מאוחסנים בטבלת ה hash עצמה. $n\le m$ $a\le 1$

עבור כל מפתח k סדרת התאים הנבדקת היא:

$h(k,i)$

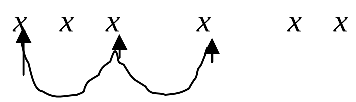
i - מספר הבדיקה.

$h(k,0),h(k,1),...,h(k,n-1)$

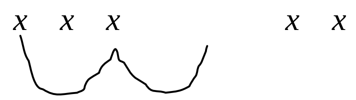
הפונקציה מנסה להכניס את המפתח לתא. אם התא תפוס היא מופעלת שוב וכך עד שנמצא תא פנוי.

<u>בעיית המחיקות</u>

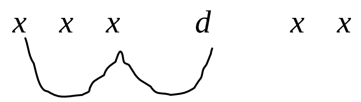
המערך לפני המחיקה: אנו מחפשים את האיבר עד שמוצאים אותו.



מצאנו את האיבר. הבעיה היא ברגע שמוחקים אותו: הפונקציה נעצרת ברגע שיש תא ריק. ישנם עוד איברים בהמשך, אך סדרת הבדיקות לא תגיע לאיבר.



לכן נסמן את התא כתא בו נמחק איבר:



כעת לצורך קריאה התא תפוס, אך לצורך כתיבה הוא פנוי.
עניין זה פוגע בסיבוכיות מציאת האיבר כשיש מחיקות.

<u>הנחת הגיבוב האחיד</u>

סדרת בדיקות היא כל תמורה של הסדרה $\{0,1,...,m-1\}$. סה"כ יש $m!$ תמורות.
לכל מפתח k יש הסתברות שווה לכל תמורה להיות סדרת הבדיקות שלו.

<u>3 קירובים לגיבוב אחיד:</u>
1. בדיקה לינארית.
2. בדיקה ריבועית.
3. גיבוב כפול.

<u>1. בדיקה לינארית</u>
בהינתן $h:U\rightarrow\{0,1,..,m-1\}$, h פונקציית גיבוב רגילה.
שיטת הבדיקה הלינארית משתמשת בפונקציית הגיבוב:
$$h(k,i)=\left(h^{/}(k)+i\right)\bmod m$$
$$h^{/}(k)=k\bmod m$$
לדוגמא: $m=10$
57,12,37,19,17,62,53

0	17
1	
2	12
3	62
4	53
5	
6	
7	57
8	37
9	19

הבעיה עם שיטה זו היא שהפיזור אינו אחיד.

<u>2. בדיקה ריבועית</u>
$$h(k,i)=\left(h^{/}(k)+c_1 i+c_2 i^2\right)\bmod m \quad c_1,c_2\neq 0$$
המרחקים בין התאים תלויים במספר הבדיקה באופן ריבועי.
$$h^{/}(k_1)=h^{/}(k_2)\Rightarrow h(k_1,i)=h(k_2,i)$$

השלמה – תרגול 6:

<u>3. גיבוב כפול</u>
$$h(k,i)-\left(h_1(k)+ih_2(k)\right)\bmod m$$

- כדי להשיג פיזור טוב:
  - o m יהיה חזקה של 2 ו h_2 תייצר מספרים אי זוגיים.
  - o m תהיה ראשונית ו h_2 תחזיר ערכים קטנים מ m .

דוגמא:
$$h_1(k)=k\bmod m$$
$$h_2(k)=1+k\bmod m$$
$m^{/}=m-1$ או $m^{/}=m-2$

משפט

בהנחת גיבוב אחיד בשיטת Open addressing מתקיים $a<1$.

זמן ממוצע של חיפוש כושל הוא לכל היותר $\frac{1}{1-a}$ בדיקות.

זמן ממוצע של חיפוש מוצלח הוא לכל היותר $\frac{1}{a}\ln\left(\frac{1}{1-a}\right)$ בדיקות.

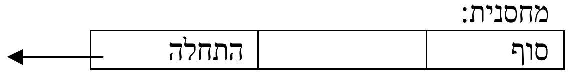
מכיוון ש $a<1$ $\quad n<O(n) \leftarrow$ סיבוכיות מוצלחת של $O(1)$.

שיטות גיבוב טובות

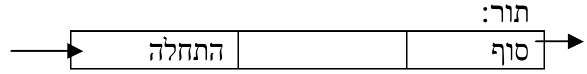
1. שיטת החילוק $h(k)=k \bmod m$. במקרה ש m הוא חזקה של 2 שיטה זו אינה טובה.
2. שיטת הכפל $h(k)=\lfloor m(kA) \bmod 1 \rfloor$. A ניתן לבחירה שלנו.
3. גיבוב אוניברסלי. מראש מכינים קבוצה של פונקציות גיבוב ובזמן ריצה בוחרים באופן אקראי פונקציית גיבוב מתוך הקבוצה.
   נשתמש בקבוצת פונקציות אוניברסליות H:
   עבור כל זוג מפתחות שונים x,y מספר פונקציות הגיבוב שיגרמו להתנגשות $h(x)=h(y)$
   הוא בדיוק $\frac{H}{m}$. כלומר הסיכוי להתנגשות בין x ל y הוא $\frac{1}{m}$.

תורים דו כיווניים

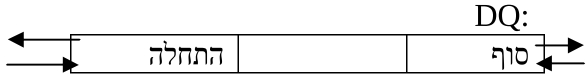
מחסנית:



תור:



DQ:



פעולות:

- מצא מינימום.
- הוסף בהתחלה.
- הוסף בסוף.
- הוצא בהתחלה.
- הוצא בסוף.

שאילתות

זמן תגובה לשאילתא – נחשב את המקרה הגרוע.
נחשב את זמן התגובה ל n שאילתות.
מתוך כך נחשב את הזמן ה"ממוצע" לשאילתא.

ניתן למצוא את המינימום ב $O(1)$ בכל פעם ע"י תחזוק עץ קרטזי. בממוצע הוספה והוצאה יקחו $O(1)$.

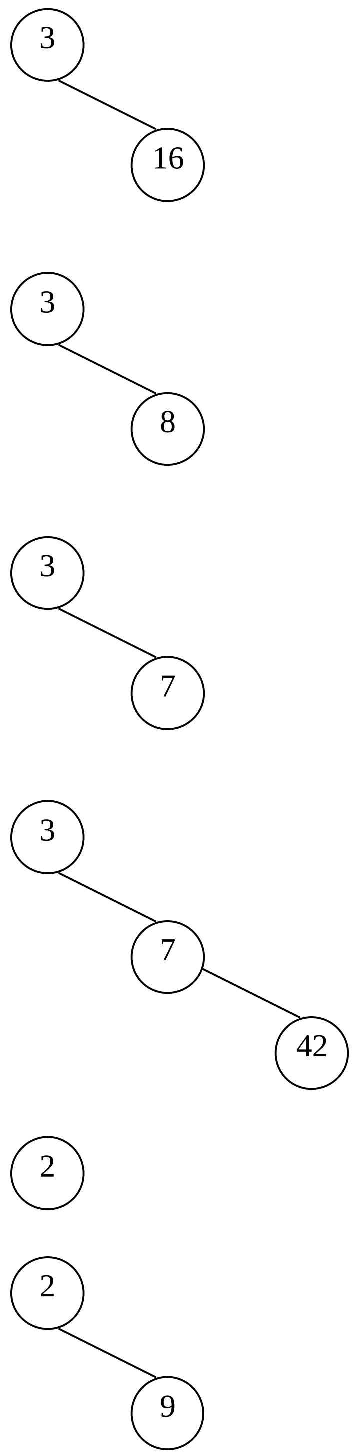
בעץ הקרטזי יהיו שני מצביעים, אחד לאיבר האחרון שהכנסנו ואחד לאיבר שבהתחלת הרשימה (האיבר הראשון).

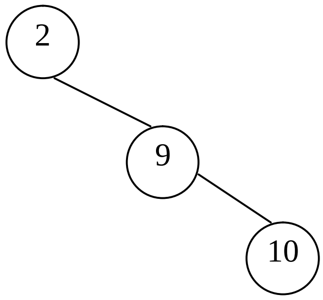
הכנסת האיבר בצד שמאל של העץ תהיה בדיוק כמו בצד ימין, אך הפוך.

הבעיה תהיה עם מחיקת איבר. תהיה בעיה לתחזק את העץ לאחר המחיקה, כלומר לדעת מה היה המינימום לפני המחיקה.

נסתכל על הדופן הימני של העץ הקרטזי. הוא בעצם רשימה מקושרת שבראשה נמצא המינימום ואחריו נמצאים המינימומים המקומיים.

דוגמא:
3,16,8,7,42,2,9,10





לכן אם מוציאים מישהו מהתור אנו צריכים רק לבדוק את ראש הרשימה המקושרת. אם הוא קטן מהראש מורידים את הראש. אם הוא גדול ממנו, מוסיפים אותו לרשימה. בממוצע כל פעולה תהיה ב $O(1)$.

מציאת מינימום במחסנית: נשתמש במחסנית נוספת. נוסיף למחסנית איבר חדש רק אם הוא יותר קטן מהמינימום.

מציאת מינימום בתור בו מוסיפים בזנב ומבטלים בסוף: נשתמש ברשימה דו כיוונית. נוסיף את הערך החדש בכל פעם לסוף הרשימה. נוריד את כל האיברים מתחילת הרשימה הגדולים ממנו. $O(n)$ לכל פעולה במקרה הגרוע.

<u>שאלה למחשבה</u>
ישנן בחירות עם n מצביעים ו m מועמדים. יש לנו רק $O(1)$ זיכרון נוסף לצורך חישוב התוצאה. האם יש מנצח (מעל 50%)?
הערה: במדעי המחשב הנחת העבודה היא שבכל משתנה אין יותר מ $O(\log n)$ ביטים.

תשובה: מוציאים זוגות של הצבעות למועדים שונים זה מזה. כל ההצבעות שישארו בסוף יהיו של המועמד שאולי ניצח (אם יש מנצח אז זה הוא). לאחר מכן סופרים את ההצבעות שהיו לו. אם הן מעל 50% מהצבעות, הוא ניצח.
אנו מתחזקים שני מונים: כמות ומועמד. כל פעם שאנו נתקלים במועמד מגדילים את הכמות. כל פעם שנתקלים במועמד אחר מורידים את הכמות. אם הכמות מגיעה לאפס, המועמד יהיה התא הבא בתור. לאחר מכן סופרים את הקולות ובודקים אם הם מעל ל 50%.