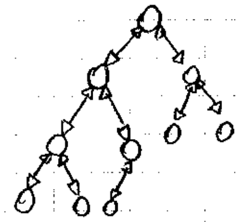
16.6.08

(1)

מבני נתונים - שיעור

<u>תזכורת</u>



ראינו כיצד ניתן לממש ADT של תור עדיפויות באמצעות ערימות (בינאריות) אותן מימשנו באמצעות מערך

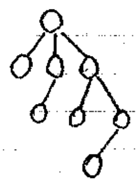
לחלופין, ניתן (אפילו טבעי) לממש ערימה כזו כמבנה מקושר, כאשר בכל צומת מוחזקים 3 מצביעים (לאב, לבן הימני ולבן השמאלי).

בשני המימושים כל השאילתות (minimum, size) מתבצעות ב-$\Theta(1)$ וכל העדכונים (insert, extract-min, decrese-key) מתבצעים ב-$O(\log n)$.

* נשים לב במקרה של עץ מקושר, כדי למצוא את המינימום של האיבר הבא בצורה יעילה, עלינו להחזיק רשימה נוספת.

מימוש נוסף של תור עדיפויות
<u>באמצעות ערימות בינומיות</u>

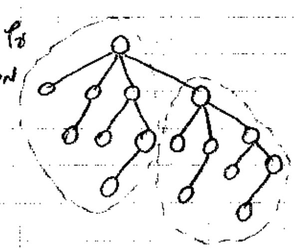
ערימה בינומית היא אוסף של עצים בינומיים. דוגמה לעץ בינומי:



עץ בינומי מדרגה 3

עץ בינומי מדרגה 4 מתקבל משני עצים בינומיים מדרגה 3 ע"י הפיכת שורש של אחד מהם לבן של השורש השני.

באופן כללי, עץ בינומי מדרגה k+1 מתקבל באופן הזה משני עצים בינומים מדרגה k.



עץ בינומי מדרגה 4

כמה קודקודים	עץ	דרגה
1		0
2		1
4		2
8		3

* הסיבה שהעץ נקרא עץ בינומי כיוון שבכל רמה ניתן לבטא את המקדמים של הבינום מרמה k. לדוגמה עבור 3, מספר הקודקודים בכל רמה הוא 1 3 3 1. פרט למקור של השם, אין לכך חשיבות.

תכונות של עצים בינומיים:

1. בעץ בינומי מדרגה n יש <u>בדיוק</u> 2^n צמתים.

2. (לפי משפט אחר) מספרי הצמתים ברמות של עץ בינומי מדרגה n הם המקדמים הבינומים. $\binom{n}{0},\binom{n}{1},\binom{n}{2},\dots,\binom{n}{n}$.

3. העומק של עץ בינומי מדרגה n הוא n.

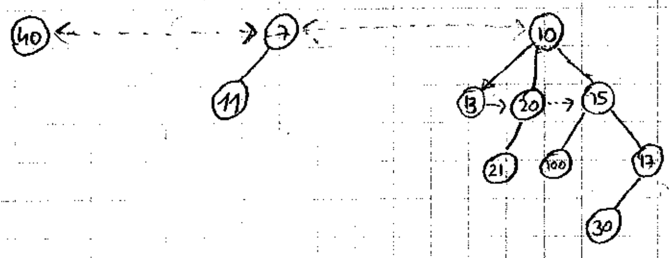
4. אם מוחקים את השורש של עץ בינומי מדרגה n הוא מתפרק ל-n עצים בינומים מדרגות $0,1,2,\dots,n-1$.

16.6.08 (2)

מבני נתונים - שיעור

ערימה בינומית, מבחינה מבנית, היא אוסף של עצים בינומיים מדרגות שונות כאשר בצמתים מאוחסן מידע ומתקיים תנאי הערימה בכל אחד מהעצים בנפרד: המפתח באב קטן מהמפתח בבן (בכל צומת).

דוגמה לערימה בינומית עם 11 צמתים:



* כאשר נתון מספר צמתים נתון, אנו יכולים להסתכל על הייצוג הבינארי שלו כדי לדעת את מבנה הערימה.

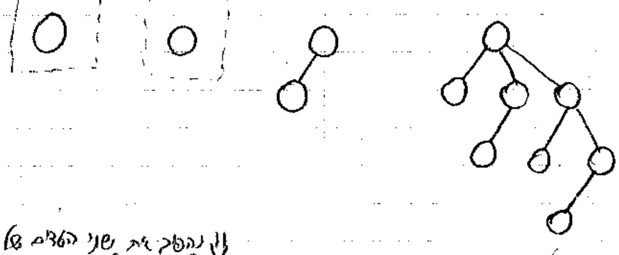
מימוש של ערימה בינומית יעשה ע"י מבנה מקושר. לכל צומת יהיו 2 מצביעים (אחד לבן השמאלי ביותר ואחד לבן הסמוך מימין). בכל צומת יהיה שדה שמחזיק את המפתח בצומת.
כמו כן, עבור שורשי העצים יהיה שדה שמחזיק את דרגת השורש ומצביעים שמחברים את השורשים לרשימה מקושרת (דו כיוונית).

בנוסף נחזיק משתנה size ומצביע לשורש עם המפתח המינימלי (שהוא כמובן המפתח המינימלי בערימה כולה).

ניגש למימוש תור עדיפות באמצעות ערימה בינומית.

$\Theta(1)$ — טריוויאלי: יצירת תור עדיפויות ריק.
השאילתות: size, minimum

הוספה insert:



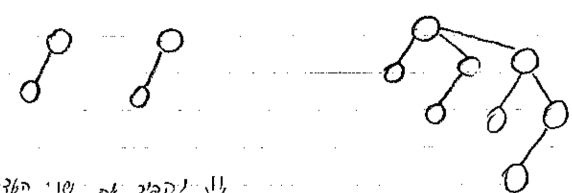
נוסף המעבר ל-11 | נוסף המעבר ל-12 ע"י כלות חוקית

10: 1 0 1 0

11: 1 0 1 1

12: 1 1 0 0

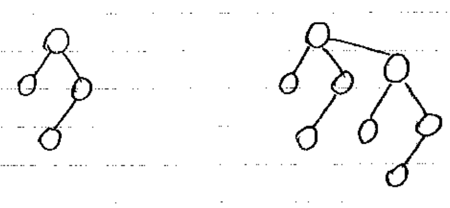
נצטרך את שני העצים של 1 לעץ של 2



1. יוצרים עץ חדש מדרגה 0 ומכניסים לצומת היחיד בו את המפתח החדש.

2. כל עוד יש שני עצים מאותה דרגה k, משלבים אותם לעץ חדש עם דרגה k+1 שהשורש שלו הוא השורש עם המפתח הנמוך יותר מבין 2 העצים. (ומעדכנים אם יש צורך את המצביע לשורש עם המפתח המינימלי בערימה).

נצטרך את שני העצים של 2 לעץ של 3

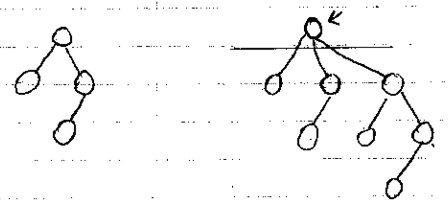


כך עכשיו מבנה הערימה חוקי

כל איטרציה של הלולאה 2 מתבצעת בזמן $\theta(1)$. מספר האיטרציות תלוי בהצגה הבינרית של n והוא לכל היותר $\log n$ ולכן בסה"כ insert יקח $O(\log n)$.

extract-min

$n=12$



מבנה לפעולות חשבון
על מס' בינריים

$12: \cancel{1}100$

100 ← העצים הנותרים

111 ← שארית העץ שהשורש שלו הוסר

הערימה לפני הפעולה מורכבת מעצים מהדרגות הבאות:

0 2 3 4 7

מורידים את השורש של העץ מדרגה 4 והעץ מתפרק לעצים מדרגות 0,1,2,3 ועכשיו יש לנו עצים מהדרגות הבאות

העצים שנשארו → 0 2 3 7

העצים שנוספו בעקבות הפירוק → 0 1 2 3

נראה את הדמיון לחיבור בינארי

0 2 3 7 →
0 1 2 3 →

$$
\begin{array}{rcccccccccl}
 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & \\
 & & & & & 1 & 1 & 1 & 1 & \\
 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & (130) \\
+ & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & (15) \\
\hline
 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 &
\end{array}
$$

⇓

1
: 2 3 7
1 2 3

⇓

2
2 3 7 ⇒ 2 3 7 ⇒ 2 3 4 7
2 3
3
3

מימוש של extract-min

1. סלק את השורש עם המפתח המינימלי (ושמור את המפתח כדי להחזיר אותו) כתוצאה מכך מתפרק העץ שפרקנו את השורש שלו ל-k עצים מדרגות $0,1,\dots,k-1$.
   דרגת השורש שפרקנו

2. עבור i מ-0 עד $\log n$:

   אם יש 2 או 3 עצים מדרגה i, שלב 2 מהם לעץ בדרגה $i+1$.
   הסיבוכיות כמובן $O(\log n)$.

הערה: צריך גם לעדכן את המצביע לשורש עם המפתח המינימלי וזה לבד ייקח $O(\log n)$ זמן.

מבני נתונים- שעור

<u>decrease-key</u> בסך הכל צריך לבצע heapify-up בעץ הבינומי
בעץ הבינומי שבו נמצא המפתח שרוצים לשנות.

זה כמובן לוקח $O(\log n)$.
(כי העומק של כל עץ בערימה הוא לכל היותר $\log n$).

האם יש יתרון לערימות בינומיות על ערימות בינאריות?

<u>חסרון</u>: ערימות בינומיות מסורבלות יותר.

<u>תיקו</u>: אותם זמני ריצה (על חסמים תיאורטיים).

<u>יתרון</u>: ניתן למזג 2 ערימות בינומיות בזמן $O(\log n)$ (תהליך דומה לזה
שראינו במסגרת extract-min) אם משתמשים בערימות בינומיות
ואילו עם ערימות בינאריות, מה שניתן לעשות הוא פשוט לבנות
(<u>מחדש</u>) ערימה בינארית מאיברי 2 הערימות - לוקח $O(n)$.

<u>ערימות פיבונאצ'י</u>

ערימת פיבונאצ'י היא אוסף של עצים שמתקבלים מאוסף של קודקודים מבודדים ע"י
הפעלת <u>כל</u> סדרה של פעולות שילוב והפרדה שמקיימות את המגבלות הבאות:

(א) אם רוצים לשלב 2 עצים לעץ אחד, השורשים של שני העצים צריכים להיות
מאותה דרגה ואז אחד מהשורשים הופך לבן של השני.

(ב) מותר לנתק קשר בין אבא לבן מאחד העצים
אבל כאשר אנו מאבדים את הבן <u>בפעם השניה</u>, חייב
להתבצע גם ניתוק של האבא מאביו.
הופך להיות שורש עצמאי

(*) דרגת שורש של עץ היא מספר הבנים שתחתיו ישירות.