אוניברסיטת בר אילן

מבני נתונים
89 - 120

תרגולים

(חלקי)

מרצה: פרופ' שמואל טומי קליין
נכתב ונערך ע"י: גלעד אשרוב

סמסטר ב', תש"ע

הערות כלליות. המסמך מכיל סיכומי תרגולים שניתנו במהלך הסמסטר (סמסטר ב', תש"ע). המסמך חלקי, כלומר, אינו מכיל את כל התרגולים שניתנו במהלך הסמסטר. אני מקווה שבשנים הקרובות בע"ה אשלים את התרגולים החסרים.... אשמח לקבל הערות ותיקונים למייל: `asharog@cs.biu.ac.il`. המסמך נכתב בלאטך לעברית (LaTeX).

תודות וקצת קרדיטים. בהזדמנות זו ארצה להודות למתרגל השני ־ צבי קופולביץ', למרצה ־ פרופ' טומי קליין, למתרגלים הקודמים בקורס ־ נטלי שפירא, שי שלומאי ודודי בן חמו. כמו־כן, אודה להילה זרוסים, יאיר דומב וארז וייסברד (שישבו לקרוא כמה דפים כדי לבדוק שהם כתובים בסדר). תודה נוספת למיטל לוי, שתירגלה בעבר באוניברסיטת ת"א, שהפנתה אותי לחומר חשוב.

השיעור על אסימפטוטיקה מבוסס על תרגולו של דודי בן־חמו, השיעור על *splay trees* מבוסס על שיעור של פרופ' אומן, והשיעור על *perfect hash* מבוסס על הרצאתו של פרופ' לוינשטיין. השיעור על *skip list* מבוסס על הרצאתו של *Eric Demaine*, הניתנת לצפייה ברשת.

הספר שעליו עבדנו רוב הסמסטר הוא הספר "מבוא לאלגוריתמים" של קורמן, ריבסט ולייזרסון.

12 ביוני 2010 גלעד אשרוב

תוכן עניינים

תרגול 1
ניתוח זמני ריצה

תאריך עדכון אחרון: 12 ביוני 2010[1]

מוטיבציה. נתחיל את דיוננו בדוגמא. נתאר שני אלגוריתמים לחישוב סידרת פיבונאצ'י. כלומר, האלגוריתם מקבל כקלט מספר n, וצריך להחזיר $fib(n)$ כאשר הפונקציה fib מוגדרת בצורה הבאה:

$$fib(n) = fib(n-1) + fib(n-2) \qquad \text{and} \qquad fib(1) = fib(2) = 1$$

נתבונן בשני האלגוריתמים הבאים לפתרון הבעיה:

```
int fib2(int n) {
    int preResult = 1, result = 1,
        temp = 0;
    for (int i=2; i<n; i++) {
        temp = result;
        result += preResult;
        preResult = temp;
    }
    return result;
}
```

```
int fib1(int n) {
    if (n == 1 || n == 2) {
        return 1;
    }
    return fib1(n-1) + fib1(n-2);
}
```

איזה אלגוריתם הינו **יעיל** יותר? איך מודדים יעילות של אלגוריתם? על זאת ננסה לענות בשיעור זה.

במבני נתונים ובאלגוריתמיקה, אנו נדרשים להציג פתרונות לבעיות המוגדרות היטב. בכדי להעריך את טיב הפתרון, או בכדי להעריך עד כמה הפתרון אופטימלי, נרצה להעריך את מספר הפעולות שהמחשב מבצע בפתרון שהצענו. נרצה לבטא את מספר הפעולות שהמחשב נדרש לבצע כפונקציה של הקלט לבעיה.

בכדי להעריך זמן ריצה, אנו נדרשים למספר הנחות מקלות. ראשית, אנו מתבוננים במכונה שבה מעבד יחיד, והפעולות אותן אנו סופרים הן פעולות בסיסיות: השמה, פעולות אריתמתיות (+, -, *, /), פעולות בוליאניות, השוואה, גישה לתא במערך וכו'. אנו מניחים שזמן הביצוע של כל פעולה שכזו הוא קבוע.

כאמור, נדבר תמיד על מספר הפעולות כפונקציה של אורך הקלט לתוכנית. נעיר כי אלגוריתם כללי לפתרון בעיה צריך לעבוד לכל קלט, ולפיכך דרישה זו הגיונית. לצרוך העניין, ברור כי מיון 5 מספרים הינו קל ומהיר יותר ממיון 1000 מספרים; אם־כן, אנו מודדים את טיב האלגוריתם כמספר הפעולות אותו הוא מבצע כפונקציה של אורך הקלט שלו, ולא כפונקציה של מספר הפעולות שביצע בפועל בריצה ספציפית.

נעיר כי אלגוריתם יכול לעבוד זמן שונה עבור שני קלטים מאותו האורך. לדוגמא, אם נרצה למיין את הסדרה $1,2,3,5,4$, לעומת הסידרה: $5,2,3,1,4$, סביר (אך לא הכרחי) שאותו אלגוריתם מיון יעבוד "קשה יותר" (זמן רב יותר) עבור הסדרה השנייה - שכן הראשונה כמעט וממויינת. לפיכך, על פי רוב, אנו נחשב את מספר הפעולות של האלגוריתם על הקלט הגרוע ביותר שהאלגוריתם יכול לקבל ($worst\ case\ analysis$).

[1] נכתב ע"י גלעד אשרוב. הסיכום נכתב בעיקירו על בסיס תרגוליו של דודי בן חמו, 2005, והספר "מבוא לאלגוריתמים" של קורמן, לייזרסון, ריבסט, שטיין - ותורגם לעברית ע"י האוניברסיטה הפתוחה.

נעיר כי לפעמים עורכים ניתוחים על קלט ממוצע לבעיה ($average\ case\ analysis$), וסיבוכיות אלגוריתם יכולה להיות שונה בשני המקרים. ברוב המקרים קשה יותר לנתח מהי הסיבוכיות עבור הקלט הממוצע מאשר סיבוכיות על המקרה הגרוע ביותר.

בכל בעיה שנדבר עליה, נציין במפורש מהו גודל הקלט; לדוגמא, במיון מספרים - נדבר על מספרים הקלטים, כלומר - n. לעומת זאת, אם נרצה לנתח אלגוריתם למכפלת שני מספרים, נדבר על מספר הסיביות הנצרכים לייצוג כל אחד מן המספרים. אם הקלט הוא גרף, נתאר את מספר הפעולות כפונקציה של מספר הקשתות בגרף, או מספר הקודקודים[2].

זמן ריצה. זמן ריצה של אלגוריתם על קלט מסויים הוא מספר פעולות היסוד המבוצעות. נרצה לחשב את סיבוכיות הזמן של אלגוריתם באופן מתמטי כך שנתעלם מאספקטים "טכנולוגיים" כגון מהירות המחשב שעליו מריצים את האלגוריתם (שכן, ברור שכאשר יוצא מחשב חדש מהיר המהיר פי שניים, האלגוריתם שלנו ירוץ מהר פי שניים). "סיבוכיות הזמן" תתאר את סדר הגודל של הפעולות הנדרשות, ותתעלם מקבועים. לצורך פשטות החישוב, נתאר את מושג ה"אסימפטוטיקה", העוזר לנו ל"הפטר" מכל הקבועים.

סיבוכיות זיכרון. לעיתים, נרצה למדוד את כמות הזיכרון שבו האלגוריתם משתמש. שוב, נרצה להתעלם מאספקטים "טכנולוגיים", ולכן גם פה נשתמש באסימפטוטיקה.

1.1 אסימפטוטיקה

אסימפטוטיקה הינה הערכה של קצב גידול של פונקציה. מה שנותר בחישוב הוא רק האיבר המשמעותי ביותר. למעשה, הדבר שקול לשאלה - למה שואף זמן הריצה כשגודל הקלט שואף לאינסוף.

1.1.1 חסם אסימפטוטי עליון - O

בד"כ, נרצה לחסום את זמן הריצה "מלמעלה". נניח תוכנית A רצה בזמן $f(n)$ עבור קלט מאורך n. אם $f(n) = O(g(n))$, אזי מספר הפעולות שהתוכנית עושה הוא סדר גודל של $g(n)$ פעולות לכל היותר. באופן מפורש יותר, שתי ההגדרות הבאות שקולות:

הגדרה 1.1 נאמר ש - $f(x) = O(g(x))$ אם ורק אם קיימים שני קבועים, $c > 0$, $x_0 \geq 0$ כך שלכל $x \geq x_0$ מתקיים:

$$|f(x)| \leq c \cdot |g(x)|$$

הגדרה 1.2 נאמר ש - $f(x) = O(g(x))$ אם ורק אם קיים קבוע $c > 0$ כך ש:

$$\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| \leq c$$

כאשר אנחנו אומרים "זמן הריצה הוא $O(n^2)$" הכוונה היא שזמן הריצה במקרה הגרוע ביותר, לקלט הגרוע ביותר, חסום ע"י $n^2 \cdot c$ כאשר c הוא איזשהו קבוע. אנחנו בעצם מסתכלים על המקרה הגרוע ביותר.

בנוסף, לעיתים רושמים ביטויים כגון: $n^2 + 5n + 2 = n^2 + O(n)$, כלומר, הופכים את $5n + 2$ ל - $O(n)$. בד"כ כשרושמים ביטויים מסוג זה, מתכוונים שישנו איזשהו $O(n)$ בביטוי שלא ממש מעניין; הגודל המעניין הוא n^2.

יש כאלו שמגדירים את $O(g(x))$ כמשפחה של פונקציות המקיימות $f(x) = O(g(x))$. באופן מפורש יותר:

$$O(g(n)) = \{f(n) \mid \exists n_0, c \geq 0 \text{ such that } \forall n \geq n_0 : \ 0 \leq f(n) \leq c \cdot g(n)\}$$

במקרה כזה, נאמר כי $f(n) \in O(g(n))$ ולא כמו בהגדרה הקודמת $f(n) = O(g(n))$. אנו נעבוד לפי שתי ההגדרות הראשונות בלבד.

[2] בקורסים מתקדמים בתיאוריה של מדעי המחשב, כגון חישוביות וסיבוכיות, אנחנו מחשבים את סיבוכיות האלגוריתם כפונקציה של מספר הביטים הנדרשים לייצוג כל הקלט. על כל פנים, בקורס זה אנו "מרמים" קצת, ומגדירים בכל פעם מהו הפרמטר בקלט שאליו אנחנו מייחסים את הסיבוכיות.

דוגמאות:

1. נניח שזמן ריצה של אלגוריתם A הוא $f(n) = 10n^2 + 5n$. אזי, ברור כי $f(n) = O(n^2)$. זאת מכיוון שקיים קבוע $c = 15$ כך שלכל n (קיים $x_0 = 0$, כך שלכל $n > x_0$) מתקיים:

$$10n^2 + 5n < 10n^2 + 5n^2 = 15n^2.$$

ולכן, לפי הגדרה 1.1 נקבל כי $f(n) = O(n^2)$. בעזרת הגדרה 1.2 נקבל את אותה התוצאה; נחשב:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{10n^2 + 5n}{n^2} = 10$$

ברור כי 10 הינו קבוע.

נציין כי אותה הפונקציה, $f(n)$ מקיימת: $f(n) = O(n^2 \log n),\ f(n) = O(n^3),\ f(n) = O(n!)$ וניתן לחשוב על עוד דוגמאות נוספות (למעשה, כל פונקציה שגדולה מ - n^2).

2. נתבונן בפונקציה $f(n) = n \log n^5 + 6n$. נראה כי $f(n) = O(n \log n)$:

$$\lim_{n\to\infty} \frac{f(n)}{n \log n} = \lim_{n\to\infty} \frac{n \log n^5 + 6n}{n \log n} = \lim_{n\to\infty} \frac{5n \log n}{n \log n} + \lim_{n\to\infty} \frac{6n}{n \log n} = 5 + \lim_{n\to\infty} \frac{6}{\log n} = 5$$

ולכן, לפי הגדרה 1.2 אנו מקבלים כי $f(n) = O(n \log n)$.

3. **טענה:** $n^2 \neq O(n)$

הוכחה: נוכיח בעזרת כל אחת מההגדרות. נתחיל עם הגדרה 1.1: נניח בשלילה שקיים c וקיים n_0 כך שלכל $n > n_0$ מתקיים:

$$n^2 \leq c \cdot n \Rightarrow n \leq c$$

בסתירה לכך שהנוסחא מתקיימת לכל $n > n_0$.

כעת, נראה הוכחה נוספת על סמך הגדרה 1.2:

$$\lim_{n\to\infty} \left| \frac{n^2}{n} \right| = \lim_{n\to\infty} |n| \longrightarrow \infty \geq c$$

לכל קבוע $c > 0$. ■

1.1.2 חסם אסימפטוטי תחתון - Ω

לעיתים נרצה לדבר על חסם תחתון לבעיה מסויימת. לדוגמא, ברור שמיון של n מספרים דורש לפחות n פעולה (סתם לבדוק אם המספרים ממויינים עולה סדר גודל של n פעולות). נרצה לחסום את זמן הריצה מלמטה. באופן מפורש יותר, שתי ההגדרות הבאות שקולות:

הגדרה 1.3 נאמר ש - $f(x) = \Omega(g(x))$ אם קיים קבוע $c > 0$, וקיים x_0 כך שלכל $x \geq x_0$ מתקיים:

$$0 \leq c \cdot g(x) \leq f(x)$$

הגדרה 1.4 נאמר ש - $f(x) = \Omega(g(x))$ אם קיים קבוע $c > 0$ כך ש:

$$\lim_{x\to\infty} \left| \frac{g(x)}{f(x)} \right| \leq c$$

דוגמאות:

1. **טענה:** $3n^2+5=\Omega(n)$.

הוכחה: נקח $c=1$. צ.ל n_0 כך שלכל $n\ge n_0$ מתקיים:

$$3n^2+5>1\cdot n\Rightarrow 3n^2-n+5>0$$

הנ"ל פרבולה "מרחפת" ולכן מספיק לקחת $n_0=1$. ∎

2. **טענה:** $3n^2+5=\Omega(n^2)$.

הוכחה:

$$\lim_{n\to\infty}\frac{n^2}{3n^2+5}=\frac{1}{3}$$

∎

3. **טענה:** $3n^2+5n\neq\Omega(n^3)$.

הוכחה:

$$\lim_{n\to\infty}\left|\frac{n^3}{3n^2+5}\right|=\infty>c$$

לכל קבוע $c>0$. ∎

1.1.3 חסם הדוק אסימפטוטית - Θ

באופן אינטואיטיבי, נאמר ש - $f(n)=\Theta(g(n))$ אם מתקיים: $f(n)=O(g(n))$ וגם $f(n)=\Omega(g(n))$.
באופן פורמלי:

הגדרה 1.5 נאמר ש - $f(n)=\Theta(g(n))$ אם קיימים קבועים $c_1,c_2>0$, וקיים קבוע n_0 כך שלכל $n\ge n_0$ מתקיים:

$$0\le c_1g(n)\le f(n)\le c_2g(n)$$

הגדרה שקולה:

הגדרה 1.6 נאמר ש - $f(n)=\Theta(g(n))$ אם קיים קבוע $c>0$ כך ש:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=c$$

דוגמא. נראה ש - $3n^2+5=\Omega(n^2)$ לפי שתי ההגדרות.
לפי הגדרה 1.5, נקח $c_1=1$, $c_2=8$, ו - $n_0=0$. לכל $n>n_0$ מתקיים:

$$c_1\cdot n^2=n^2\le 3n^2+5\le 8n^2=c_2n^2.$$

בכדי להראות זאת לפי הגדרה 1.6, נקבל:

$$\lim_{n\to\infty}\frac{3n^2+5}{n^2}=3.$$

טענה 1.7 לכל שתי פונקציות $f(n)$, $g(n)$: $f(n)=\Omega(g(n))$ וגם $f(n)=O(g(n))$ אם ורק אם: $f(n)=\Theta(g(n))$.

ההוכחה נשארת לקורא כתרגיל (לא שיש יותר מדי מה להוכיח פה..).

1.1.4 הסימון - o

כפי שציינו, הסימון O מציין חסם עליון לפונקציה - לאו דווקא הדוק. כאשר אנו יודעים בוודאות שהחסם אינו הדוק, ניתן להשתמש ב - o (o "קטן"). לדוגמא, נתבונן בפונקציה $5n$. נקבל כי $5n = O(n)$ וגם $5n = O(n^2)$, אך החסם האחרון אינו הדוק אסימפטוטית. במקרה זה נרשום: $n = o(n^2)$. נדגיש כי $5n \neq o(n)$. באופן פורמלי, נקבל:

הגדרה 1.8 נאמר ש - $f(n) = o(g(n))$ אם לכל קבוע חיובי $c > 0$ קיים קבוע $n_0 > 0$ כך שלכל $n \geq n_0$ מתקיים:

$$0 \leq f(n) < cg(n)$$

הגדרה שקולה:

הגדרה 1.9 נאמר ש$f(n) = o(g(n))$ אם מתקיים:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

1.1.5 הסימון - ω

כפי שהגדרנו יחס שבין o קטן לבין O גדול, מציג עכשיו את היחס שבין Ω לבין ω. אנו משתמשים ב - ω בכדי לציין חסם תחתון שאינו הדוק אסימפטוטית. לדוגמא, אם הפונקציה היא n^2 אזי היא שייכת גם ל - $\Omega(n^2)$ וגם ל - $\Omega(n)$. החסם $\Omega(n)$ אינו הדוק, ולכן ניתן לרשום $n^2 = \omega(n)$, אך: $n \neq \omega(n)$.

הגדרה 1.10 נאמר כי $f(n) = \omega(g(n))$ אם לכל קבוע חיובי $c > 0$ קיים קבוע $n_0 > 0$ כך שלכל $n \geq n_0$ מתקיים:

$$0 \leq cg(n) < f(n)$$

הגדרה 1.11 נאמר ש$f(n) = \omega(g(n))$ אם מתקיים:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

1.1.6 השוואת פונקציות

למעשה, פונקציות $O, \Omega, \Theta, o, \omega$ הן מעין יחס גדול / קטן:

$$\begin{aligned}
f(n) = O(g(n)) &\quad\approx\quad f(n) \leq g(n) \\
f(n) = \Omega(g(n)) &\quad\approx\quad f(n) \geq g(n) \\
f(n) = \Theta(g(n)) &\quad\approx\quad f(n) = g(n) \\
f(n) = o(g(n)) &\quad\approx\quad f(n) < g(n) \\
f(n) = \omega(g(n)) &\quad\approx\quad f(n) > g(n)
\end{aligned}$$

כמובן שכל היחסים - $=, \leq, \geq, <, >$ הם אסימפטוטיים, ולא שיוויונים אמיתיים.
רבים מן היחסים המתקיימים בין מספרים ממשיים מתקימים גם בהשוואות אסימפטוטיות.

טרנזיטיביות.

- אם $f(n) = \Theta(g(n))$ וגם $g(n) = \Theta(h(n))$ אזי $f(n) = \Theta(h(n))$.
- אם $f(n) = O(g(n))$ וגם $g(n) = O(h(n))$ אזי $f(n) = O(h(n))$.
- אם $f(n) = \Omega(g(n))$ וגם $g(n) = \Omega(h(n))$ אזי $f(n) = \Omega(h(n))$.
- אם $f(n) = o(g(n))$ וגם $g(n) = o(h(n))$ אזי $f(n) = o(h(n))$.
- אם $f(n) = \omega(g(n))$ וגם $g(n) = \omega(h(n))$ אזי $f(n) = \omega(h(n))$.

רפלקסיביות.

- $f(n) = \Theta(f(n))$
- $f(n) = O(f(n))$
- $f(n) = \Omega(f(n))$

סימטריה. $f(n) = \Theta(g(n))$ אם ורק אם $g(n) = \Theta(f(n))$.

בנוסף, נשים לב:

$$\begin{aligned} \text{constants}: &\quad n^{\frac{1}{c+1}} < \frac{1}{n^c} < \frac{1}{\log n} < 1 < \ldots \\ \text{logarithms}: &\quad < \log n < (\log n)^k < \ldots \\ \text{``polynomials''}: &\quad < \sqrt{n} < n < n \log n < n^k < n^{k+1} < \ldots \\ \text{exponentials}: &\quad < 2^n < n! < n^n \end{aligned}$$

שאלה: בהינתן $f(x)$, $g(x)$, האם תמיד מתקיים־ $g(x) = O(f(x))$ או $f(x) = O(g(x))$?
תשובה: לא. נתבונן ב - $f(n) = n^{1+\sin n}$, וב - $g(n) = n$. לא ניתן להשוות בינהם אסימפטוטית.

שאלה: האם לכל פונקציה $f(n)$ קיימת פונקציה "סטנדרטית" ($n, n \log n, n^2, \ldots$) $g(n)$ כך ש - $f(n) = \Theta(g(n))$?
תשובה: לא. נתבונן ב - $f(n)$ המוגדרת באופן הבא:

$$f(n) = \begin{cases} n & n \text{ odd} \\ n^3 & \text{o.w} \end{cases}$$

תרגיל 1.12 הוכח: $\log(n!) = \Theta(n \log n)$

הוכחה:
מתקיים:

$$\log(n!) = \log(1 \cdot 2 \cdot \ldots \cdot n) = \log\left(\prod_{i=1}^{n} i\right) < \log\left(\prod_{i=1}^{n} n\right) = \log n^n = n \log n$$

ולכן $\log(n!) = O(n \log n)$.

נותר להראות כי קיים c עבורו: $\log(n!) > cn\log n$. נקבל:

$$\begin{aligned}
\log(n!) &= \log(1\cdot 2\cdot \ldots \cdot n) = \sum_{i=1}^{n}\log i = \sum_{i=1}^{\frac{n}{2}-1}\log i + \sum_{i=\frac{n}{2}}^{n}\log i \\
&> \sum_{i=\frac{n}{2}}^{n}\log i > \sum_{i=\frac{n}{2}}^{n}\log\frac{n}{2} = \frac{n}{2}\log\frac{n}{2} = \frac{n}{2}(\log n - \log 2) \\
&= \frac{n}{2}(\log n - 1) = \frac{n}{2}\log n - \frac{n}{2}
\end{aligned}$$

כאשר $n > 4$ נקבל כי: $\log n > 2$, ולכן $\frac{n}{4}\log n > \frac{n}{2}$. נקבל כי לכל ה־$n$ים המספיק גדולים:

$$\log(n!) > \frac{n}{2}\log n - \frac{n}{2} > \frac{n}{2}\log n - \frac{n}{4}\log n = \frac{n}{4}\log n$$

ולכן, כאשר $c = \frac{1}{4}$, ו ־ $n_0 = 4$, נקבל כי לכל $n > n_0$ מתקיים: $\log(n!) > c\cdot n\log n$, כלומר ־ $\log(n!) = \Omega(n\log n)$. נסיק אם כן כי $\log(n!) = \Theta(n\log n)$ ■

1.2 סיבוכיות קוד

ניתן ללמוד את סיבוכיות האלגוריתם מתוך מבט כללי על מבנה הקוד. לפי מספר הלולאות שהקוד מבצע, לפי קריאות רקורסיביות, וכו'.

דוגמא: נתבונן בקוד הבא:

```
for (unsigned u=0; u<n; ++u) {
    basic_step1;
    basic_step2;
}
```

ישנה לולאה שרצים עליה n פעמים, בכל פעם מבצעים 2 פעולות, ולכן סיבוכיות הקוד הינה ־ $2n$, כלומר ־ $O(n)$ (ולמעשה $\Theta(n)$).

דוגמא נוספת: נתבונן בקוד הבא:

```
for (unsigned u=0; u<10; ++u) {
    basic_step1;
    basic_step2;
}
```

מספר הפעולות שהאלגוריתם מבצע הוא:

$$\sum_{u=0}^{9} 2 = 20$$

ולכן, נקבל $O(1)$.

עוד דוגמא: נתבונן בקוד הבא:

```
for (int i = n; i > 0; --i) {
    for (unsigned j=0; j<n; ++j) {
        basic_step;
    }
}
```

$$\sum_{i=1}^{n}\sum_{j=0}^{n-1}1=\sum_{i=1}^{n}n=n^2$$

כלומר, $O(n^2)$.

ועוד אחת: נתבונן בקוד הבא:

```
for (int i = 1; i <= n; i*=2) {
    basic_step;
}
```

בכל איטרציה מבצעים בדיוק פעולה אחת. כמה איטרציות יש לנו? נעקוב אחרי i: הערכים אותם הוא מקבל: $1,2,4,8,\ldots,n$. יש לנו למעשה סידרה הנדסית עם הפרמטרים: $a_1=1$, $a_k=n$, $q=2$. לפי נוסחא לסידרה הנדסית: $a_k=a_1\cdot q^{k-1}$, כלומר: $n=1\cdot 2^k$, ולכן $k=\log n$. כלומר, מספר האיטרציות הוא $\log n$. מכיוון שבכל איטרציה מבצעים פעולה אחת, נקבל כי הסיבוכיות היא: $O(\log n)$.

מה קורה כאשר הלולאות תלויות אחת בשניה? לדוגמא, נתבונן בקוד הבא:

```
for (int i = 1; i <= n; i *= 2) {
    for (j = 1; j <= i; j++) {
        basic_step;
    }
}
```

במקרה זה לא נוכל סתם לכפול את הלולאה החיצונית בלולאה הפנימית; נתבונן קודם בניתוח **לא מדויק**: הלולאה החיצונית מתבצעת $\log n$ פעמים, הלולאה הפנימית מתבצעת במקרה הגרוע ביותר n פעמים, ולכן נקבל בסה"כ - $O(n\log n)$. **הנ"ל אינו חסם הדוק!**

בכדי לקבל חסם הדוק, נחשב לפי סיגמאות. נשים לב כי i מקבל ערכים $1,2,4,8,\ldots,n$, לפיכך, נגדיר משתנה k שירוץ מ - 0 ועד ל - $\log n$, ונגדיר את i להיות 2^k. נקבל:

$$\sum_{k=0}^{\log n}\sum_{j=1}^{i}1=\sum_{k=0}^{\log n}\sum_{j=1}^{2^k}1=\sum_{k=0}^{\log n}2^k=2^{\log n+1}-1=2\cdot 2^{\log n}-1=2n-1=\Theta(n)$$

כלומר, הסיבוכיות היא לינארית.

תרגול 3
ניתוח לשיעורין

תאריך עדכון אחרון: 12 ביוני 2010.

ניתוח לשיעורין ($amortized\ analysis$) הוא טכניקה לניתוח זמן ריצה לסדרת פעולות, אשר מאפשר קבלת חסמי זמן ריצה נמוכים יותר מאשר חסמים המתקבלים כאשר מניחים את המקרה הגרוע ביותר בכל פעולה. בדרך כלל אנחנו מתייחסים למספר פעולות, כאשר חלק מהפעולות יקרות יותר, וחלקן ־ זולות יותר. בניתוח לשיעורין אנו מדברים על עלות כל פעולה בממוצע, עבור כל רצף של פעולות אפשרי של שימוש במבנה הנתונים. אנו מדברים על המקרה הגרוע ביותר; כלומר, לוקחים בחשבון את הסידרה הגרועה ביותר של פעולות שיכולה להיות.

ניתוח לשיעורין היא איזושהי אסטרטגיה לניתוח כל רצף של פעולות המראה שממוצע העלות לכל פעולה הוא קטן, למרות שישנן פעולות יחידות בתוך הרצף שעלולות להיות יקרות. נעיר שלמרות שאנו מדברים על ממוצעים, אנו לא מערבים הסתברויות בניתוח.

המטרה היא חישוב חסם עליון הדוק ככל שניתן לעלות של רצף כלשהו של n פעולות. נניח עלות הפעולה ה ־ i היא c_i, המטרה היא לחשב $T(n) = \sum_{i=1}^{n} c_i$. אזי עלות כל פעולה בממוצע היא $T(n)/n$.

נראה שלוש שיטות לניתוח לשיעורין ־ שיטת הצבירה, שיטת החיובים (שיטת ה"בנק") ושיטת הפוטנציאל. שיטת הפוטנציאל היא החשובה ביותר.

3.1 דוגמא ־ מחסנית

נתבונן במחסנית התומכת בפעולות הבאות:

- $\mathrm{push}(x)$ ־ מכניס את x למחסנית. עלות ־ $O(1)$.
- $\mathrm{pop}(x)$ ־ להוציא את האיבר האחרון שנכנס (אם יש כזה). עלות ־ $O(1)$.
- $\mathrm{multi\text{-}pop}(x)$ ־ מוציאה את כל האיברים מהמחסנית. עלות: אם יש k איברים כרגע במחסנית אזי ־ $O(k)$.

נממש (בפסאדו קוד) את $\mathrm{multi\text{-}pop}()$:

```
multi-pop()
    while not isEmpty()
        pop()
```

נשים לב שכל הפעולות עולות $O(1)$, מלבד הפעולה multi-pop. בפרט, הפעולה הנ"ל נראית יקרה ־ היא לינארית במספר האיברים במחסנית. כך, אם ישנם n איברים במחסנית, עלותה ־ $O(n)$ (מה שנשמע רע מאוד ביחס ל ־ $O(1)$). אבל ־ האם אנחנו לא מחמירים מדי כשאנו מנתחים את הפעולה בצורה כזו? כדי לחדד את השאלה, נתבונן בניתוח השגוי הבא. נשאל עבור סידרה של n פעולות ־ מהי העלות $worst\ case$ לסידרה. כעת, יכולים לטעון (בטעות) שכל פעולה מסוג multi-pop עולה $O(n)$ בכל פעם, ולכן בסה"כ רצף של n פעולות עלול להגיע ל ־ $O(n^2)$. אך, ניתוח זה לא הדוק.

אצלינו, בכדי שיהיו n איברים במחסנית, צריכים להיות n פעולות push לפני כן. אם נתונים לנו רצף של פעולות מסוימות, נניח ־ n פעולות push ולאחריהן פעולת multi-pop ־ עבור n פעולות ה־push אנחנו משלמים n בסה"כ, ועל פעולת multi-pop אנו משלמים n. כלומר בסה"כ אנו משלמים עבור רצף $n+1$

הפעולות $2n$, מה שאומר שבממוצע לכל פעולה אנו משלמים 2. כלומר, העלות הממוצעת לכל פעולה ־ היא $O(1)$.

כעת נראה שיטות ואסטרטגיות לחישוב של ניתוח לשיעורין.

3.2 שיטת הצבירה

בשיטת הצבירה אנו מראים שלכל n, סדרה של n פעולות צורכת זמן של $T(n)$ לכל היותר. נקבל אם כן, שממוצע זמן של כל פעולה, או העלות לשיעורין של כל פעולה, הינה $T(n)/n$. זוהי העלות לשיעורין של כל פעולה גם כאשר הסידרה מכילה סוגים שונים של פעולות.

בדוגמת המחסנית ננתח כמה עולה n פעולות על המחסנית. יש שלוש פעולות אפשריות: `push`, `pop`, `multi-pop`. אמנם `multi-pop` עולה $O(n)$ במקרה הגרוע, אך נשים לב שכל עצם ניתן לשלוף רק פעם אחת עבור כל פעם שהוא נדחף למחסנית. לכן, מספר הקריאות ל `pop` (גם אלו שבתוך המימוש של `multi-pop`) הוא לכל היותר מספר הקריאות ל ־ `push`. מכיוון שאנו מתבוננים בסידרה מאורך n, אין יותר מ ־ n קריאות ל ־ `push`, ולפיכך מספר הקריאות הכולל ל ־ `pop` (גם אלו שבתוך `multi-pop`) הוא לכל היותר n. נקבל כי העלות הכוללת היא לכל היותר $2n$, ולכן בממוצע עלות כל פעולה היא 2, כלומר $O(1)$.

3.2.1 יותר פורמלי

הקבוצה שלי ביום רביעי ראתה את זה, וכמו־כן, הקבוצות של צבי. הורדתי את זה בשיעורים ביום ראשון בכדי להעביר את העיקר. אתם נדרשים לקרוא ולנסות להבין את ההוכחה.

בשיטת הצבירה אנו מחשבים סכום כל העלויות עבור סידרה כללית של פעולות. מכיוון שאנו מתבוננים על סידרה כללית, הניתוח יכסה גם את הסידרה הגרועה ביותר. בכדי לדבר על סידרה כלשהי ־ נתבונן בסידרה כללית של פעולות:

$$S = (\sigma_1, \sigma_2, \ldots, \sigma_n)$$

כל פעולה $\sigma_i \in \{1, 2, 3\}$, כאשר ־ 1 מציין את הפעולה `push`, 2 מציין ־ `pop`, ו ־ 3 מציין ־ `multi-pop`.
אם עלות הפעולה ה ־ i היא c_i, נרצה לחשב:

$$T(n) = \sum_{k=1}^{n} c_k$$

נתבונן בכל המקומות בסידרה כך ש ־ $\sigma_i = 3$ (`multi-pop`). נניח שכל הפעולות הנ"ל נמצאות במקומות $I = \{i_1, \ldots, i_t\}$, וכך ש ־ $i_1 < i_2 < \ldots < i_t$. (נעיר שאם I ריקה, אזי $T(n)$ הוא סכום של קבועים (אין פעולת `multi-pop`) ולכן הסכום כולו לינארי ב ־ n). בנוסף, נגדיר כי ־ $i_0 = 0$.

נסמן את סידרת הפעולות של S בין המקומות i_{j-1} ובין i_j ב ־ S_{i_j}. כלומר, הסידרה:

$$S_{i_j} = \{\sigma_{i_{j-1}+1}, \ldots, \sigma_{i_j}\}$$

כעת, נראה את הטענה הבאה:

טענה 3.1 לכל $1 \leq j \leq t$, עלות הפעולה *multi-pop* הנמצאת במקום i_j (כלומר, עלות הפעולה σ_{i_j}) הינה $i_j - i_{j-1} - 1$ לכל היותר. בנוסף, עלות הסידרה S_{i_j} הינה $2(i_j - i_{j-1} - 1)$.

הוכחה: יהי j נתון. לאחר הפעולה $\sigma_{i_{j-1}}$ המחסנית ריקה (שכן, פעולה זו הייתה פעולת `multi-pop`). כל שאר הפעולות אינן פעולות `multi-pop` (חתכנו את סדרת הפעולות בין זוג פעולות `multi-pop`). לכן, לכל פעולה בסדרה הנ"ל, הפעולה היא `push` או `pop`. במקרה הגרוע ביותר ־ כל הפעולות הן `push`, מה שאומר שכאשר מגיעים לפעולה σ_{i_j} (פעולת ה ־ `multi-pop`) ־ ישנם $i_j - i_{j-1} - 1$ איברים במחסנית. לפיכך, העלות של פעולת ה ־ `multi-pop` היא $i_j - i_{j-1} - 1$. לכן, העלות הכוללת לסידרה S_{i_j} היא לכל היותר ־ $2(i_j - i_{j-1} - 1)$.

■

כעת, לכל סידרה S ניתן להתייחס ל ־ $I = \{i_1, \ldots, i_t\}$. נקבל:

$$T(n) \leq \sum_{j=1}^{t} 2(i_j - i_{j-1} - 1) + \sum_{j=i_t+1}^{n} c_j = 2(i_t - i_0) - 2t + (n - i_t)$$

נזכור ש ־ i_t הינו איזשהו אינדקס בתחום $1 \leq i_t \leq n$, וכמו־כן ־ i_0 הינו ־ 0. לפיכך:

$$T(n) = 2(i_t - i_0) - 2t + (n - i_t) = 2i_t - 2t + n - i_t = n + i_t - 2t \leq 2n - 2t \leq 2n$$

כלומר, הצלחנו להראות שלכל סידרה של פעולות מאורך n, העלות הכוללת היא לכל היותר $2n$. הנ"ל מראה שעלות ממוצעת של כל פעולה היא $O(1)$.

3.3 שיטת החיובים ("שיטת הבנק")

בשיטה זו ־ ניתן לכל פעולה עלות שונה מזו שהיא עולה באמת. עלות זו נקראת "עלות לשיעורין". חלק מהפעולות יקבלו עלות גדולה יותר מהעלות האמיתית שלהן; חלק מהפעולות יקבלו עלות קטנה יותר מזו שהן עולות בפועל (בדרך כלל, הפעולה היקרה ביותר בפועל תקבל את העלות הקטנה ביותר לשיעורין). העיקרון שצריך תמיד לשמור עליו, הוא שסידרה של n פעולות (לכל n) לפי העלויות לשיעורין שלנו ־ תהיה גדולה יותר מהעלות האמיתית שלהן בפועל.

נשתמש ב"בנק" בשביל "לשמור" לנו עלויות. נניח שעל פעולת push אנחנו משלמים 2 יחידות במקום יחידה אחת; יחידה אחת תהיה העלות האמיתית של הפעולה (שהיינו צריכים לשלם ממילא), והיחידה השנייה פשוט תכנס ל"בנק" בשביל "התחשבנות עתידית". כעת, כאשר נבצע פעולת multi־pop ־ לא נשלם מ"הכיס" אלא נמשוך את העלויות שצברנו בבנק; מכיוון שעל כל איבר שיש כרגע במחסנית ־ כבר צברנו יחידה אחת בבנק, נקבל כי יש בבנק מספיק כסף בשביל לממן את הפעולה multi־pop, ולכן לא נשלם עבור פעולה זו מהכיס בכלל.

הסבר נוסף ־ כאשר אני מכניס איבר למחסנית (ומשלם עליו 1), אני משלם גם עבור ההוצאה שלו כבר בזמן ההכנסה (עוד יחידה אחת). לכן, בסה"כ הכנסה עולה 2 יחידות; כעת, הוצאה יחידה, או הוצאה ע"י multi־pop הן למעשה ־ בחינם (שילמנו עליהן קודם; בבנק יש מספיק כסף בשביל לשלם על ההוצאה שלהן).

יותר פורמלי. נסמן ב־ $\hat{c}_i$ את העלות לשיעורין של הפעולה ה ־ i. נסמן ב ־ c_i את העלות האמיתית של הפעולה ה־ i. העלות לשיעורין של הפעולה ה ־ i בשיטת הבנק הינה:

$$\hat{c}_i = c_i + deposit - withdraw$$

כלומר ־ עלות הפעולה היא העלות האמיתית + ההפקדה שאנו מבצעים לבנק, פחות העלות של המשיכה (במידת הצורך).

כאשר אנו שומרים על העיקרון שלעולם "לא ניכנס למינוס" (לעולם לא נמשוך מהבנק סכום כסף שלא הפקדנו אותו לפני כן), מתקיים שסך כל ההפקדות (סכום כל הערכים של $deposit$) גדול מכל הפעמים שמשכנו מהבנק (סכום כל הערכים שביצענו $withdraw$), ולכן:

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i = T(n)$$

כלומר, העלות האמיתית של n הפעולות היא קטנה מסכום העלות לשיעורין. כעת, ננתח כל אחת מפעולות המחסנית:

- עבור פעולת push - עלות הפעולה היא 1, בנוסף, מכניסים 1 לבנק. נקבל:

$$\hat{c}_i = 1 + 1 - 0 = 2$$

- עבור פעולת pop - עלות הפעולה היא 1. לא מפקידים לבנק, ולא מושכים ממנו[1]. נקבל:

$$\hat{c}_i = 1 + 0 - 0 = 1$$

- עבור פעולת multi-pop - נניח שמספר האיברים כרגע הוא k. עלות הפעולה אם כן היא k. כעת, נמשוך מהבנק k יחידות (בדוק שהבנת מדוע אנו בטוחים כי בבנק יש k יחידות). נקבל:

$$\hat{c}_i = k + 0 - k = 0$$

אם כן, העלות לשיעורין של כל פעולה קטן מ - 2. מכיוון שלעולם לא נכנסים למינוס, נקבל כי:

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i = T(n)$$

אצלינו, לכל i, $\hat{c}_i \leq 2$. כלומר:

$$T(n) \leq \sum_{i=1}^{n} \hat{c}_i \leq 2n$$

ולכן סכום התשלומים עבור n פעולות קטן מ - $2n$. כלומר, העלות הממוצעת לכל פעולה היא 2.

חשוב לציין שכאשר מבצעים ניתוח בשיטת הבנק - חייבים תמיד לשמור על האינווריאנטה שסכום הכסף בבנק הוא חיובי. אסור בשום אופן "להכנס למינוס", שכן כאשר אנו "נכנסים למינוס" - העלות האמיתית של הפעולות היא פחות מהעלות לשיעורין שאנו מבצעים, ואז לא נוכל לטעון שהעלות הכוללת של העלויות לשיעורין גדולה מהעלויות האמיתיות.

3.4 שיטת הפוטנציאל

שיטת הפוטנציאל דומה מאוד לשיטת הבנק, רק שאנו ממדלים אותה באופן יותר מתמטי, ולכן היא יותר פורמלית (ופחות אינטואיטיבית..).

בשיטת הבנק "התשלום מראש" היה שמור כיתרה לזכותם של עצמים ספציפיים במבנה הנתונים (כל איבר שמר לעצמו את העלות להוצאתו בעתיד). כאן, ה"תשלום מראש" מובע בעזרת "אנרגיה פוטנציאלית" שניתן לשחרר כדי לשלם עבור פעולות בעתיד.

כמו בשיטת הבנק - פעולת push תגדיל לנו את הפוטנציאל של מבנה הנתונים (ונשלם על הגדלה זו), ופעולת multi-pop תשולם בעזרת "פירוק הפוטנציאל הזה", או, ריקון הפוטנציאל (ואין צורך לשלם עליה - כי כבר שילמנו על הפוטנציאל עצמו).

[1] יכולנו להגדיר שגם במקרה זה אנו מושכים את העלות מהבנק, ולא משלמים מהכיס. במקרה זה נקבל כי העלות לשיעורין היא 0. נציין שעדיין לא נכנס למינוס, וכל המשך הניתוח הוא בדיוק אותו הדבר.

שיטת הפוטנציאל. יהי D_0 המצב ההתחלתי של מבנה הנתונים שעליו מבוצעות n פעולות. נסמן ב ־ c_i את העלות של הפעולה ה ־ i. D_i הינו המצב של המערכת לאחר הפעולה ה ־ i, על המצב D_{i-1}. כלומר, מתחילים ממצב D_0, מבצעים פעולה כלשהי (σ_1) ומגיעים למצב D_1. לאחר מכן מבצעים פעולה כלשהי (σ_2) ומגיעים למצב D_2, וכן הלאה.

פונקציית הפוטנציאל ϕ ממפה כל מצב של מבנה הנתונים D_i למספר ממשי שמציין את הפוטנציאל המיוחס למבנה הנתונים.

נגדיר את העלות לשיעורין של הפעולה ה ־ i כ:

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

כלומר, העלות לשיעורין של הפעולה ה־ i היא העלות האמיתית c_i + הפרש הפוטנציאלים של מבנה הנתונים. במילים אחרות, העלות האמיתית של הפעולה ה ־ i הינה:

$$c_i = \hat{c}_i + \phi(D_{i-1}) - \phi(D_i)$$

העלות של כל סידרה כלשהי של n פעולות:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} c_i = \sum_{i=1}^{n} [\hat{c}_i + \phi(D_{i-1}) - \phi(D_i)] \\ &= \sum_{i=1}^{n} \hat{c}_i + \sum_{i=1}^{n} [\phi(D_{i-1}) - \phi(D_i)] \\ &= \sum_{i=1}^{n} \hat{c}_i + \phi(D_0) - \phi(D_n) \end{aligned}$$

כאשר הצעד האחרון נכון מכיון שהטור הינו טלסקופי. אם תמיד יתקיים $\phi(D_0) \leq \phi(D_n)$, נקבל כי :

$$T(n) = \sum_{i=1}^{n} \hat{c}_i + \phi(D_0) - \phi(D_n) \leq \sum_{i=1}^{n} \hat{c}_i$$

כלומר,

$$T(n) = \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$$

גם פה, כמו בשיטת הבנק, אנו שומרים על כך שסכום עלות n פעולות קטן יותר מסכום ה"עלות לשיעורין". לכן, הניתוח הופך לפשוט יותר: כל שצריך הוא לחשב את $\hat{c}_i$ לכל אחת מהפעולות הקיימות על מבנה הנתונים.

ניתוח מחסנית בעזרת שיטת הפוטנציאל. נגדיר את פונקציית הפוטנציאל $\phi(D_i)$ כמספר האיברים שיש כרגע (D_i = מצב) במחסנית. נקבל:

$$\phi(D_0) = 0$$

ובנוסף:

$$\forall 1 \leq i \leq n, \phi(D_i) \geq 0$$

כלומר, פונקציית פוטנציאל זו טובה ועונה על הדרישה.

הפיתוח הנ"ל אומר לנו שבכדי להראות שהעלות הממוצעת של כל פעולה בכל סדרה הינו קבוע, מספיק להראות ש־ $\hat{c}$ קבוע. לכן, נחשב עבור כל אחד מסוגי הפעולות.

- עבור פעולת push ־ עלות הפעולה הוא 1, ונניח שהיו t איברים במחסנית. לאחר הפעולה, ישנם $t+1$ איברים במחסנית. נקבל:

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) = 1 + (t+1) - t = 2$$

- עבור פעולת pop ־ עלות הפעולה הוא 1, ונניח שהיו t איברים במחסנית. לאחר הפעולה, ישנם $t-1$ איברים במחסנית. נקבל:

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) = 1 + t - 1 - t = 0$$

- עבור פעולת multi־pop. נגיד שיש כרגע t איברים במחסנית. אזי, עלות הפעולה הוא ־ t, ומצד שני ־ הפרש הפוטנציאלים הוא $-t$ (לפני הפעולה היו t איברים במחסנית, ולאחריה ־ 0). נקבל אם כן:

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) = t + 0 - t = 0$$

נקבל:

$$T(n) = \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \leq \sum_{i=1}^{n} 2 = 2n$$

ולכן העלות הממוצעת של כל פעולה ־ 2.

3.5 מונה בינארי

נניח יש לנו מונה עם k ביטים, הסופר בבינארית. אנחנו למעשה תומכים רק בפעולה אחת ־ $increment$ ־ העלה ב ־ 1. עלות כל פעולה = מספר הביטים שהוחלפו. לדוגמא: (לצד כל ערך בינארי מצויין עלות פעולת ה $increment$)

$$\begin{array}{ccc} 00000 & \rightarrow & 0 \\ 00001 & \rightarrow & 1 \\ 00010 & \rightarrow & 2 \\ 00011 & \rightarrow & 1 \\ 00100 & \rightarrow & 3 \\ 00101 & \rightarrow & 1 \\ 00110 & \rightarrow & 2 \\ & \vdots & \end{array}$$

כמה עולה הפעולה $increment$? כמה עולה רצף של n פעולות? שוב, ניתוח שגוי יאמר כי בכל פעם, במקרה הגרוע ביותר אנו מבצעים $O(k)$ החלפות (שכן יש k ביטים במונה. במקרה הגרוע ביותר ־ נחליף את כל הביטים במונה). ולכן, במקרה הגרוע ביותר ־ $O(k \cdot n)$. ניתוח זה שגוי. נבצע ניתוח לשיעורין.

3.5.1 ניתוח לפי שיטת הצבירה

במקרה זה, מכיוון שישנה רק פעולה אחת ($increment$) הניתוח לפי שיטת הצבירה פשוט. כל רצף של פעולות נראה בדיוק אותו הדבר.

נתבונן על n פעולות. הביט הראשון יתחלף n פעמים (בכל פעם). הביט השני לעומת זאת, יתחלף בכל פעם זוגית, כלומר ב ־ $\frac{n}{2}$ פעמים. הביט השלישי יתחלף כל פעם רביעית, כלומר בסה"כ ב ־ $\frac{n}{4}$ פעמים. הביט ה

i ־ יתחלף $\frac{n}{2^{i-1}}$ פעמים. כלומר:

$$\begin{aligned} T(n) &= \text{number of bits that were changed} \\ &= \sum_{i=1}^{k} \Big[\text{number of times that the i}^{\text{th}} \text{ bit was changed}\Big] \\ &= \sum_{i=1}^{k} \frac{n}{2^{i-1}} = n \sum_{i=1}^{k} \frac{1}{2^{i-1}} \leq 2n \end{aligned}$$

כלומר, $T(n)/n \leq 2$.

3.5.2 ניתוח לפי שיטת הפוטנציאל

נגדיר פונקציאת פוטנציאל: $\phi(D_i)$ = מספר האחדות במונה[2]. מתקיים: $\phi(D_0) = 0$, ולכל $1 \leq i \leq n$ נקבל: $\phi(D_i) \geq 0$, ולכן פונקציית פוטנציאל זו עונה על הדרישה שלנו.
נראה כמה עולה פעולת $increment$. כאשר מבצעים פעולת $increment$, האפס הראשון מימין הופך ל ־ 1, וכל שאר האחדות מימין אליו ־ מתאפסים. נניח שישנם t אחדות רצופים מסוף המונה (מימין). במצב כזה, עלות פעולת $increment$ תהיה $t+1$ (שכן נשנה את t האחדות לאפסים, ואת האפס שלאחריהם ל ־ 1). מצד שני, נשים לב שכעת יש פחות $t-1$ אחדות בביטוי (שכן t אחדות הפכנו לאפסים, ואפס אחד הפכנו ל ־ 1). נקבל:

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) = t + 1 - (t-1) = 2$$

לכן, רצף של n פעולות יעלה פחות מ ־ $2n$. כלומר, הממוצע לכל פעולה ־ 2.

3.6 מערך דינאמי

המערך תומך בפעולת $insert$ ־ ומכניס את האיבר למקום הפנוי הבא. נניח שבמערך יש n מקומות, ונניח שהמערך מלא. כעת, כאשר נבקש להוסיף איבר נוסף, נבצע את הפעולות הבאות:

- נקצה מערך בגודל $2n$.
- נעתיק את n האיברים הישנים.
- נוסיף את האיבר החדש במקום $n+1$.

הנחה: זמן הקצאה: $O(1)$.
שוב, נעיר שניתוח של n פעולות יכול להיות שגוי ־ במקרה הגרוע ביותר, $insert$ עולה $O(n)$, ולכן n פעולות יעלה $O(n^2)$. שוב, הניתוח הנ"ל שגוי מכיוון שכדי באמת להגיע לפעולה שעולה $O(n)$, צריך לבצע קודם לכן n פעולות שעולות $O(1)$ כל אחת, ולכן ממוצע כל פעולה הוא קבוע.
ננתח בעזרת שיטת הפוטנציאל בלבד.
בשיטת הפוטנציאל, ננסה לתת מוטיבציה לבחירת פונקציאת הפוטנציאל. ככל שיש יותר איברים במערך ־ הסיכוי שנצטרך להעתיק אותו למערך חדש הוא גדל. "הפוטנציאל" המצטבר במערכת הוא יותר גדול, ולכן נרצה לשלם על כך מראש (כדי שפירוק הפוטנציאל יהיה "בחינם").
אם כן, נרצה לבדוק עד כמה אנו קרובים לנקודת פירוק הפוטנציאל. לשם כך נשתמש בשני משתנים ־ num ו ־ size. num ישמור את מספר האיברים שיש כרגע במערך. size ישמור את אורך המערך. נשים לב שכאשר $\mathsf{num} = \mathsf{size}$ ־ בהכנסה הבאה נצטרך לפרק את הפוטנציאל. כלומר, כאשר $\mathsf{num} = \mathsf{size} + 1$ נצטרך להגדיל את size פי שתיים. בנוסף, כאשר $\mathsf{size} \approx 2 * \mathsf{num}$ ־ נקבל כי הגדלנו את מבנה הנתונים פי שתיים רק זה מכבר, ונרצה שבנקודה זו ערך הפוטנציאל יהיה 0. ככל שיכניסו יותר איברים ־ ערך הפוטנציאל יעלה, עד אשר נגיע למצב $\mathsf{num} = \mathsf{size}$. כאן נרצה שערך הפוטנציאל יהיה מספיק בשביל לשלם עבור ההרחבה הבאה.

[2] במדעי המחשב, הפונקציה שסופרת את מספר האחדות בביטוי בינארי נקראת "משקל המינג".

פונקציית פוטנציאל שמתארת את הדרישות שלנו היא:

$$\phi(D_i) = 2 \cdot \mathsf{num} - \mathsf{size}$$

מכיוון שהמערך תמיד מלא בלפחות חצי מהאיברים ($\mathsf{size} \leq 2\mathsf{num}$) נקבל כי לכל i $\phi(D_i)$ חיובי. בנוסף, $\phi(D_0) = 0$. כלומר, פונקציה זו עונה על הדרישה.
כלומר, אם גודל המערך הוא 16, ויש לנו 9 איברים, ערך הפוטנציאל הוא 2. כאשר נגיע ל־ 16 איברים, ערך הפוטנציאל יהיה כבר 16, ופירוקו ישלם לנו עבור ההעתקה למערך החדש.
כעת נחשב את העלות לשיעורין. נחלק לשני מקרים:

- אם הפעולה ה־ i לא גורמת להרחבת מבנה הנתונים, נקבל כי $\mathsf{size}_i = \mathsf{size}_{i-1}$, $c_i = 1$ ו $\mathsf{num}_i = \mathsf{num}_{i-1} + 1$. לכן:

$$\begin{aligned}
\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\
&= 1 + 2 \cdot \mathsf{num}_i - \mathsf{size}_i - (2 \cdot \mathsf{num}_{i-1} - \mathsf{size}_{i-1}) \\
&= 1 + 2 \cdot (\mathsf{num}_{i-1} + 1) - \mathsf{size}_i - 2 \cdot \mathsf{num}_{i-1} - \mathsf{size}_{i-1} \\
&= 1 + 2 + 2 \cdot \mathsf{num}_{i-1} - 2 \cdot \mathsf{num}_{i-1} \\
&= 3
\end{aligned}$$

- אם הפעולה ה־ i כן גורמת להרחבת מבנה הנתונים, נקבל כי $\mathsf{size}_i = 2 \cdot \mathsf{size}_{i-1}$ (אנו מגדילים את גודל המערך פי שניים). כמו כן, $c_i = \mathsf{num}_i = \mathsf{num}_{i-1} + 1$ ־ העלות של הפעולה ה־ i היא העתקת num_{i-1} איברים, והכנסת איבר נוסף אחד. מספר האיברים כעת הוא $\mathsf{num}_i = \mathsf{num}_{i-1} + 1$. לבסוף, מכיוון שהמערך הועתק ־ נקבל כי המערך היה מלא. כלומר $\mathsf{num}_{i-1} = \mathsf{size}_{i-1}$. אם כן:

$$\begin{aligned}
\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\
&= \mathsf{num}_i + 2 \cdot \mathsf{num}_i - \mathsf{size}_i - (2 \cdot \mathsf{num}_{i-1} - \mathsf{size}_{i-1}) \\
&= 3 \cdot \mathsf{num}_i - \mathsf{size}_i - 2 \cdot \mathsf{num}_{i-1} + \mathsf{size}_{i-1} \\
&= 3 \cdot \mathsf{num}_i - 2(\mathsf{num}_i - 1) - 2 \cdot \mathsf{size}_{i-1} + \mathsf{size}_{i-1} \\
&= 3 \cdot \mathsf{num}_i - 2 \cdot \mathsf{num}_i + 2 - \mathsf{size}_{i-1} \\
&= \mathsf{num}_i - \mathsf{size}_{i-1} + 2 \\
&= 1 + 2 = 3
\end{aligned}$$

שכן, מספר האיברים כרגע במערך (num_i) גדול מאחד מגודל המערך לפני ההגדלה.

קיבלנו כי לכל סוג של פעולה $\hat{c} \leq 3$, ולכן, העלות הכוללת של כל סידרת פעולות היא:

$$T(n) = \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \leq \sum_{i=1}^{n} 3 = 3n$$

ולכן, העלות הממוצעת לכל פעולה היא $T(n)/n = 3$.

3.7 לקריאה נוספת

- קורמן, לייזרסון, ריבסט: "מבוא לאלגוריתמים", הוצאת MIT (ראה [2]). תורגם לעברית ע"י האוניברסיטה הפתוחה.

תרגול 4
מחסנית ורשימת דילוגים

תאריך עדכון אחרון: 12 ביוני 2010.

לשים לב! הסיכום הפעם לא מכיל את כל החומר שנלמד בשיעור; יש להשלים מהדברים שלמדנו בכיתה. כמו־כן, הסיכום אינו מכיל ציורים שכנראה הכרחיים לשם הבנת החומר.

4.1 מבוא ־ מהו מבנה נתונים?

מבנה נתונים הוא דגם המגדיר את היחסים בין הנתונים ואת הפעולות המבוצעות עליהם. אנו מראים אילו פועלות ניתן לבצע על מבנה הנתונים, וכיצד מנהלים אותו. אנו נפריד בין שני סוגים של פעולות על מבנה הנתונים ־ "שאילתות" (queries), כלומר החזרת מידע על מבנה הנתונים, לבין פעולות "שינוי" (modifying operations) שבהם משנים את מבנה הנתונים עצמו. לאחר בניית מבנה הנתונים, אפשר להתחייס למבנה הנתונים כ־דינאמי, כלומר ־ מגדירים כיצד יבוצעו פעולות השינוי, לעומת זאת, ניתן להתייחס על מבנה נתונים סטאטי, שבו לא מגדירים פעולות שינוי, אלא מניחים שברגע שמבנה הנתונים כבר בנוי וכל הפעולות שיתבצעו עליו יהיה מהסוג ־ "שאילתות".

לדוגמא, ניתן להגדיר את מבנה הנתונים S ועליו להגדיר את הפעולות הבאות:

- $Search(S,x)$ מחפש אחר האיבר x במבנה הנתונים S.
- $Min(S)$ מחזיר את האיבר ב־S בעל המפתח הקטן ביותר.
- $Insert(S,x)$ מכניס את האיבר x למבנה הנתונים S.
- $Delete(S,x)$ מוציא את x ממבנה הנתונים S.

נשים לב שהפעולות $Insert, Delete$ הינן פעולות "שינוי", בעוד שפעולת $Search$ ו ־ Min אלו פעולות מסוג "שאילתות".

על מנת לתאר את מבנה הנתונים, תחילה נתאר אותו באופן אבסטרקטי (מערכת הקשרים על הנתונים, ומהן הפעולות שיש לבצע על הנתונים). לאחר מכן, מגדירים את האלגוריתם לכל אחת מן הפעולות בפסאדו קוד [1]. בד"כ בשלב זה נרצה להעריך את סדר גודל כל פעולה שהגדרנו, ונרצה לשפר את מבנה הנתונים כך שיתאים בצורה אופטימלית לפעולות שאותן אנחנו ממשים. את שני השלבים האלה נעשה ברוב הקורס. לבסוף, בכדי להשתמש בפועל במבנה הנתונים, נצטרך לממש את מבנה הנתונים בשפת תוכנה מסוימת, לדוגמא $C++$.

4.2 רשימת דילוגים ־ Skip - List

בשלב זה אנו מניחים כי הסטודנטים מכירים את מבני הנתונים ־ רשימה מקושרת, מחסנית, תור, וכמו־כן ־ כיצד לממש מבני נתונים אלו, מה ההבדל בין מימוש בעזרת מערך ובין מימוש בעזרת רשימה מקושרת.

[1] כלומר, כותבים את הפתרון בצורה טכנית הנראית כמו קוד, אך לא מתייחסים לפרטים קטנים ומעצבנים של מימוש בשפת תוכנה מסויימת. בעוד שקוד רגיל צריך לדעת לקרוא **מחשב**, את הפסאדו קוד צריכים לקרוא **בני אדם**, ולכן צריך לכתוב תוכנית בצורה שיהיה ברור מה מנסים לכתוב.

בניגוד למבני נתונים שנלמד בהמשך הקורס, שבהם נתקל בפרטים קטנים שלפעמים קשים לזכירה - רשימת דילוגים הוא מבני נתונים פשוט וקל לזכירה. מבני הנתונים הזה הוא כנראה גם החדש ביותר שנלמד בקורס, ופורסם ע"י $Pugh$ ב - 1990. ראה [4].

נדבר על הפעולה - $search(x)$. אם נשמור את הנתונים בתוך מערך - ונשמור תמיד שהמערך יהיה ממוין, חיפוש יעלה לנו $O(\log n)$ (ניתן לבצע חיפוש לינארי). גם אם x לא נמצא בקבוצה, נוכל להחזיר את הקודם או העוקב במבנה הנתונים.

הבעיה במערך היא הכנסה והוצאה. כל עוד שומרים שהמערך ממוין, בכדי להכניס איבר בין שני איברים - יש צורך "להזיז" את שאר האיברים במערך, מה שיכול להעלות $O(n)$. כנ"ל לגבי הוצאה.

אם עוברים לרשימה מקושרת - בעיית ההכנסה וההוצאה נפתרת. אך כעת, גם כאשר הרשימה ממויינת - לא ניתן לבצע חיפוש איבר בזמן $O(\log n)$. זאת מכיוון שלא ניתן לבצע גישה ישירה לאיבר במקום ה - i (רשימה מקושרת לא תומכת ב $random\ access$). אם כן, נרצה לשפר את החיפוש ברשימה מקושרת; נרצה לבצע מעין חיפוש בינארי - ברשימה מקושרת.

נדבר כרגע רק על מבנה נתונים סטטי. כלומר - כל האיברים כבר נתונים וידועים מראש, והפקודה היחידה שעליה אנו מדברים היא $search(x)$ - פעולת שאילתא (נעיר שהסיבה היחידה שבה אנו מדברים על רשימת דילוגים ולא על מערך היא מכיוון שאנו בכל זאת רוצים לדבר על $insert$ ו- $delete$, ולכן ההתמקדות במבני נתונים סטטי נראה תמוה; על כל פנים, נראה את רעיון מבני הנתונים, ונציין לאחר מכן כיצד מבצעים את הפעולות הנ"ל. כלומר, כיצד הופכים את מבנה הנתונים מסטטי - לדינאמי).

4.2.1 רעיון ראשון - שתי רשימות

נתבונן ברשימה מקושרת, שבה כל איבר מצביע לאיבר שאחריו, ולאיבר שלפניו (רשימה מקושרת דו כיוונית). נוסיף רשימה מקושרת נוספת מעליו. רשימה זו תהיה רשימה של "נציגים" של הרשימה המקורית. לא נשמור את כל האיברים ברשימה זו, אלא רק נציגים מהרשימה שמתחתיה. כל נציג יצביע לאיבר שמתחתיו - לאיבר האמיתי. כך, כאשר נרצה לבצע חיפוש, נבצע חיפוש רגיל ברשימה מקושרת ברשימת הנציגים. כאשר נמצא את הקודם והעוקב של האיבר אותו אנו מחפשים, נרד לרשימה מתחתיה (שבה כל האיברים) - ושם נחפש את האיבר עצמו, בין שני הנציגים.

כמה נציגים כדאי לבחור? מה המרחק בין כל שני נציגים?

לשאלה השנייה נענה כרגע - מרחק אחיד (נעיר שזוהי תשובה אופטימלית בסדר גודל, אם כי, ניתן לשפרה בקבועים - ראה בתרגיל). באשר לשאלה הראשונה, נתבונן בעצם כמה עולה לנו חיפוש. תהי L_1 הרשימה המלאה, ותהי L_2 רשימת הנציגים. חיפוש איבר x במבנה הנתונים שיצרנו עולה:

$$T(search_2) = |L_2| + \frac{|L_1|}{|L_2|}$$

כלומר, במקרה הגרוע ביותר אנו עוברים על כל רשימת הנציגים. לאחר מכן, אנו עוברים על חלק מהרשימה המלאה. מכיוון שחליקנו את הנציגים באופן אחיד, המרחק בין כל שני נציגים הוא $|L_1|/|L_2|$ (אם נניח ברשימה המלאה יש 20 איברים, וברשימת הנציגים יש 4 איברים, בין כל שני נציגים יש בערך 5 איברים; בכל מעבר ברשימת הנציגים, אנו "מדלגים" על 5 איברים אמיתיים).

מתי ביטוי זה מקבל מינימום? ניתן לגזור, את הביטוי, ולגלות כי הביטוי מגיע למינימום כאשר $L_2 = \sqrt{|L_1|}$. בצורה זו, חיפוש עולה $2\sqrt{L_1}$.

למעשה, הצלחנו להוריד את זמן החיפוש מ - $|L_1| = n$ ל - $2\sqrt{L_1} = 2\sqrt{n}$. סיבוכיות המקום עלתה מ - n ל - $n + \sqrt{n} = O(n)$, כלומר - אין הבדל בסדר גודל בכל הקשור לסיבוכיות מקום. כבר יש לנו שיפור משמעותי.

4.2.2 ובצורה כללית..

נמשיך לבנות רשימת נציגים לנציגים, נקבל שלוש רמות. נניח שכל רשימה של נציגים מחולקת באופן אחיד. נקבל אם כן, שזמן החיפוש בשלוש רשימות הינו:

$$T(search_3) = |L_3| + \frac{|L_2|}{|L_3|} + \frac{|L_1|}{|L_2|}$$

ביטוי זה מקבל מינימום כאשר $|L_3| = \sqrt[3]{|L_1|}$, $|L_2| = \sqrt[3]{|L_1|^2}$ ואז נקבל: $T(search_3) = 3\sqrt[3]{n}$. בצורה כללית:

$$\begin{aligned}
T(search_1) &= n \\
T(search_2) &= 2 \cdot \sqrt{n} \\
T(search_3) &= 3 \cdot \sqrt[3]{n} \\
T(search_4) &= 4 \cdot \sqrt[4]{n} \\
&\vdots \\
T(search_k) &= k \cdot \sqrt[k]{n} \\
&\vdots \\
T(search_n) &= n \cdot \sqrt[n]{n} = n
\end{aligned}$$

נשים לב שהביטוי מורכב מ ־ $k \cdot \sqrt[k]{n}$ ־ כאשר החלק הראשון במכפלה (k) הוא מספר הרמות, והחלק השני ־ הוא מספר האיברים שמדלגים עליהם בכל רמה.

אם כן, מהו מספר הרמות האופטימלי?

הביטוי $k \cdot \sqrt[k]{n}$ מקבל מינימום כאשר $k = \log n$. במקרה זה נקבל כי $T(search_k) = \log n \cdot \sqrt[\log n]{n}$, מכיוון שמתקיים $\sqrt[\log n]{n} = 2$, נקבל כי: $T(search_{\log n}) = \log n \cdot 2$. כלומר, יש לנו במבנה הנתונים $\log n$ רמות (רשימות), כל אחת מכילה נציגים של הרמה מתחתיה, והרווח בין כל שני נציגים הוא 2. בצורה כזו, נקבל כי:

- ברמה התחתונה יהיה כל האיברים.
- ברמה שמעליה יהיו $1/2$ מהאיברים.
- ברמה שמעליה יהיו $1/4$ מהאיברים (חצי מהאיברים שברמה האחת לפני האחרונה).
- ברמה שמעליה יהיו $1/8$ מהאיברים,
- וכן הלאה.

נקבל אם כן, שסיבוכיות הזיכרון היא:

$$n \cdot \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots\right) = 2n = O(n)$$

חיפוש איבר עולה לנו $O(\log n)$ ־ בדיוק כמו בחיפוש בינארי.

נעיר כי מה שהצגנו פה הוא "Ideal Skip List", כלומר ־ רשימת דילוגים אידיאלית. לא הראינו כיצד מבצעים $insert$ ו ־ $delete$; נעיר שישנם שתי דרכים לבצע זאת:

- **בצורה הסתברותית:** כלומר, בכל פעם שמכניסים איבר לרשימה, "מטילים מטבע" בכדי להחליט האם להעלות אותו נציג לרמה למעלה, ובכל שלב מטילים מטבעות האם להמשיך ולהעלות אותו למעלה. בצורה כזו, נקבל כי המרחקים אינם בהכרח בעלי מרחק אחיד, וכמו־כן, מספר הרמות יכול להעלות (או להיות קטן יותר) מ ־ $O(\log n)$. על כל פנים, ניתן להוכיח **שבהסתברות גבוהה**, הכנסה, הוצאה וחיפוש עולים $O(\log n)$. כמו־כן, מספר הרמות יהיה $O(\log n)$ בהסתברות גבוהה. מכיוון שהסטודנטים בשלב זה לא למדו הסתברות ברמה מספיקה לשם ניתוח מבנה הנתונים, אנו נאלצים שלא להראות את הפעולות הנ"ל, והניתוח למקרה זה, ולכן מסתפקים רק במבנה נתונים סטאטי...
- **בצורת ניתוח לשיעורין:** ניתן להתחיל עם $skip\ list$ אידיאלי, ולאחר מכן ־ להכניס איברים ו"ללכלך" אותו, ולאבד מהאופטימליות שלו. כאשר "מלכלכים" יותר מדי (מתי?) נצטרך לבצע שוב "סדר" ולאזן אותו. בצורה כזו, ניתן להראות שהעלות לשיעורין של כל פעולה היא $O(\log n)$, אם כי, ישנה פעולה ־ פעולת ה"איזון מחדש" שעולה לפעמים יותר; על כל פנים, נבצע אותה פעם אחת בסדרה של הרבה פעולות שנבצע במבנה הנתונים, ולכן בעלות לשיעורין ־ היא "נבלעת".

4.3 מחסנית

מחסנית היא אוסף סדור של פריטים. ההכנסה וההוצאה נעשית אך ורק דרך ראש המחסנית. הפריט שנכנס לראש המחסנית הינו זה שנמצא בראש המחסנית, ומדיניות ההכנסה וההוצאה של הפריטים במחסנית היא Last In First Out - LIFO. ניתן לבצע גישה במחסנית רק לאיבר הנמצא בראשה. פעולות אפשריות במחסנית:

- דחיפה $push(S, x)$: הכנסת אלמנט למחסנית. ההכנסה תתבצע לראש המחסנית.
- שליפה $pop(S)$: הוצאת האלמנט שבראש המחסנית. אם המחסנית ריקה - פעולה זו גורמת להודעת שגיאה ($exception$): stack underflow (חמיקה).
- האם ריקה - $isEmpty(S)$: מחזירה האם המחסנית S הינה ריקה.
- ראש: $top(S)$ - מחזירה את האיבר שנמצא בראש המחסנית (ולא מוציאה אותו; זוהי שאילתא).

נעיר שעל פי ההגדרה גודל המחסנית אינו מוגבל, ולכן ניתן לבצע את הפעולה $push$ כאוות נפשנו. אך, ייתכן ולפעמים באימלפמנטציות מסויימות נרצה להגביל את גודל המחסנית. במקרה שהמחסנית מלאה, ומישהו מבצע פעולת $push$ - נקבל הודעת שגיאה שנקרא $stack\ overflow$. כאמור, לעיתים ניתן לקבל גם הודעת $stack\ underflow$, בכדי להימנע מכך - לפני כל פעולת pop נבצע את פעולת $isEmpty$, ונבצע את פעולת ה - pop אך ורק אם $isEmpty$ החזיר `false`.

4.3.1 דוגמא לשימוש במחסנית - בדיקת חוקיות של סוגריים

בהינתן ביטוי הכולל מספר של זוגות סוגריים מקוננים, כיצד נוודא שהסוגריים מקוננים כהלכה?

לדוגמא: $[(1+2)+(3+(4+5))]+6$ - מקוננים כהלכה. לעומת זאת - $(5+[4+3)]+2+1$ אינם מקוננים כהלכה.

הערה: אם היה קיים רק סוג סוגריים אחד בביטוי, אז ניתן היה לפתור זאת בקלות ללא מחסנית ע"י שימוש במונה (כיצד?).

נשים לב שכדי שביטוי יהיה חוקי צריך להתקיים:

- קיום מספר שווה של סוגריים ימניים ושמאליים.
- קיום סוגר שמאלי תואם לכל סוגר ימני.
- לא קיימת רישא של הביטוי שבו מספר הסוגריים הימניים גדול ממספר הסוגריים השמאליים.

האלגוריתם: נאתחל מחסנית להיות מחסנית ריקה.

```
נקרא את הביטוי משמאל לימין:
    ברגע שנתקל בסוגר פותח (סוגר שמאלי) - נדחוף אותו למחסנית.
    ברגע שנתקל בסוגר חותם (סוגר ימני) - נבדוק:
        אם המחסנית ריקה - אזי נחזיר "הביטוי לא תקין".
        אם המחסנית לא ריקה - נבדוק האם האיבר הפותח של הסוגריים
            שבראש המחסנית (הסוגר השמאלי) מתאים לאיבר שאנו מחזיקים
            כרגע ביד (הסוגר הימני).
            אם לא - נחזיר "הביטוי לא תקין". אחרת - נמשיך בסריקה.
כאשר סיימנו לקרוא את הביטוי - נבדוק את המחסנית.
אם היא לא ריקה - נחזיר - "הביטוי לא תקין"
אחרת - נחזיר "הביטוי תקין".
```

4.3.2 דוגמא לשימוש במחסנית ־ המרת ביטוי מייצוג $infix$ לייצוג $postfix$

ביטוי בצורה $infix$ הינו ביטוי (לדוגמא) מהצורה:

$$2 \cdot (2+3) + 3 \cdot 4 - 5 \cdot 6 \cdot (7-4-2).$$

ביטוי המתאים בצורת $postfix$ נראה כך:

$$2, 3, +, 2, \cdot, 3, 4, \cdot, +7, 4, -, 2, -, 6, \cdot, 5, \cdot, -$$

שם של משתנה / מספר נקרא בשם **אופרנד**. לפעולות עצמן $(+, -, \cdot, \ldots)$ אנו קוראים בשם **אופרטור**.

ביטוי בצורת $infix$ יותר אינטואיטיבי לבני אדם, וכך אנו משתמשים ביום יום. ביטוי בצורת $postfix$ הינו קל יותר למחשב בשביל לבצע אבלואציה ־ כלומר, בשביל לחשב את הביטוי (תראו בהרצאה שאכן קל לחשב ביטוי כאשר הוא נתון בצורת $postfix$). נראה כעת איך ניתן להמיר ביטוי בצורת $infix$ לביטוי ב ־ $postfix$.

האלגוריתם צריך להתייחס לקדימויות של אופרטורים, קדימויות של סוגריים, לסדר של האופרנדים, וכו'.

האלגוריתם:

```
1: אתחל מחסנית ריקה S (מחסנית האופרטורים).
2: לכל סמל c בביטוי (משמאל לימין):
  2.1: אם אופרנד (מספר / משתנה) ־ הדפס c.
  2.2: אחרת: (c אופרטור ־ סימן)
    2.2.1: כל עוד S לא ריקה ו ־ top(S) קודם ל ־ c:
           (קודם מבחינת סדר פעולות)
    2.2.2: הדפס pop(S).
  2.3: push(c ,S).
3: כל עוד S לא ריקה ־ הדפס pop(S)
```

במימוש זה התעלמנו בסוגריים. נעיר שבכדי לדעת להעביר גם סוגריים ־ צריך לבצע רקורסיה.

דוגמא. נראה דוגמא עבור $3 \cdot 4 - 5 \cdot 6$:

- המחסנית ריקה, מתבוננים באיבר הראשון ־ 3 ומדפיסים אותו. הביטוי המתקבל:

$$3$$

- רואים אופרטור ־ $(\cdot)$. המחסנית ריקה ־ מכניסים את $\cdot$ למחסנית.
- רואים אופרנד ־ 4. מדפיסים אותו. מקבלים:

$$3, 4$$

- רואים אופרטור $(-)$. בודקים את ראש המחסנית. מגיון שבראש המחסנית יש אופרטור בעל ייחס קדימות ($\cdot$ קודם ל ־ $-$), מוציאים את $\cdot$ מהמחסנית, ומדפיסים אותו. כעת המחסנית ריקה, ומכניסים לתוכה את $-$. נקבל:

$$3, 4, \cdot$$

- רואים 5, מדפיסים אותו. נקבל:

$$3, 4, \cdot, 5$$

- רואים $(\cdot)$, בודקים את ראש המחסנית, $\cdot$ קודם ל־ $-$, ולכן רק מכניסים את $\cdot$ למחסנית.
- רואים 6, מדפיסים אותו, ומרוקנים את המחסנית. נקבל:

$$3, 4, \cdot, 5, 6, \cdot, -$$

תרגול 8
Splay Trees

8.1 סקירה כללית

במסגרת ההרצאה למדתם עצים מאוזנים, ובפרט עצי AVL. בשיעור זה נלמד עצי $splay\ trees$. עצי $splay\ trees$ הם עצי חיפוש בינאריים, והומצאו בשנת 1985 ע"י $Sleator$ ו־ $Tarjan$. ראה [5].

עץ חיפוש זה "מאזן את עצמו". בנוסף, הוא בעל תכונה מעניינת שבה האיברים האחרונים שאותם חיפשנו יימצאו בראש העץ (קרובים לשורש), ולכן חיפושם יהיה מהיר. הרעיון הוא שאיבר שחיפשנו לא מזמן ־ כנראה שנחפש אותו שוב בעתיד הקרוב, ולכן נרצה להחזיר אותו ב"מהירות".

הפעולה הבסיסית בעצי $splay\ trees$, היא הפעולה $splay$. הפעולה $splay(x,T)$ מחזירה עץ חיפוש בינארי עם אותם האיברים כמו ב ־ T כך ש־ x בשורש. אם x לא ב ־ T, בשורש יהיה y כך ש:

- או ש־ y הוא הגדול ביותר ב־ T שקטן מ ־ x (כלומר, מבין כל האיברים ב ־ T, y הוא הקודם ל ־ x).
- או ש ־ y הוא הקטן ביותר ב ־ T שגדול מ ־ x (כלומר, מבין כל האיברים ב ־ T, y הוא העוקב ל ־ x).

נשים לב ש ־ $splay$ אינה פעולת הכנסה או פעולת הוצאה; זוהי פעולה שרק משנה את צורת העץ. מימוש כל שאר הפעולות על העץ ייעשה בעזרת הפעולה $splay$:

- $member(x,T)$ ־ האם x חלק מהעץ T. בכדי לממש פעולה זו, נבצע $splay(x,T)$, אם $x \in T$, נקבל כי ־ x בשורש העץ, ולכן נחזיר "כן". אחרת ־ נחזיר ־ "לא".
- $delete(x,T)$ ־ מחק את האיבר x מ־ T. נבצע $splay(x,T)$ ונקבל חזרה T'. למעשה. נקבל עץ שבו ־ x בשורש, וקיים לנו תת עץ שמאלי T_1, ותת עץ ימני T_2. נרצה להוציא את x ואיכשהו "לאחד" את שני העצים T_1, T_2. כיצד נבצע את האיחוד כך שהעץ המתקבל יהיה עץ חיפוש בינארי?

  ניקח את T_1. פה ־ כל האיברים קטנים מ ־ x. נבצע $splay(T_1, +\infty)$. נקבל חזרה T_1', כך שהאיבר שנמצא בשורש הוא האיבר הגדול ביותר בעץ T_1. לפיכך, בהכרח, אין לו בן ימני, אלא רק בן שמאלי. כבן ימני ־ "נשתול" את העץ T_2. העץ שקיבלנו הוא חוקי מכיוון שכל האיברים ב ־ T_1 קטנים מ ־ x, וכל האיברים ב ־ T_2 גדולים מ ־ x. לפיכך, כל האיברים ב ־ T_2 גדולים מהאיבר הגדול ביותר ב ־ T_1 ־ שהוא זה שיימצא בשורש.
- $insert(x,T)$ ־ נכניס את x לתוך T כמו כל הכנסה בעץ חיפוש בינארי. לאחר מכן מבצעים פעולת $splay$ בכדי להעלות אותו לשורש.

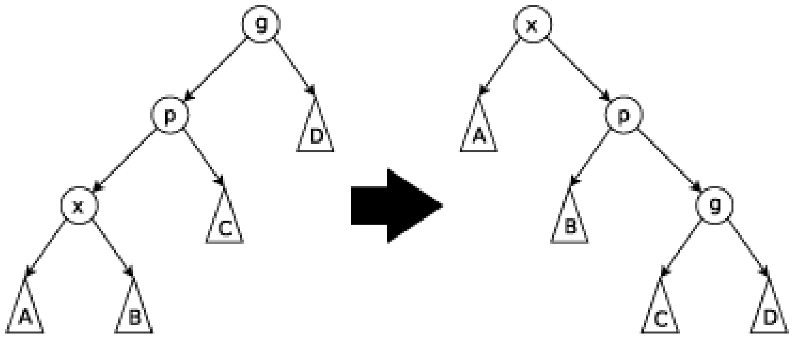
אם כן, הפעולה היחידה שנותרה לנו היא הפעולה $splay(T,x)$.

8.2 מימוש הפעולה $splay(T,x)$

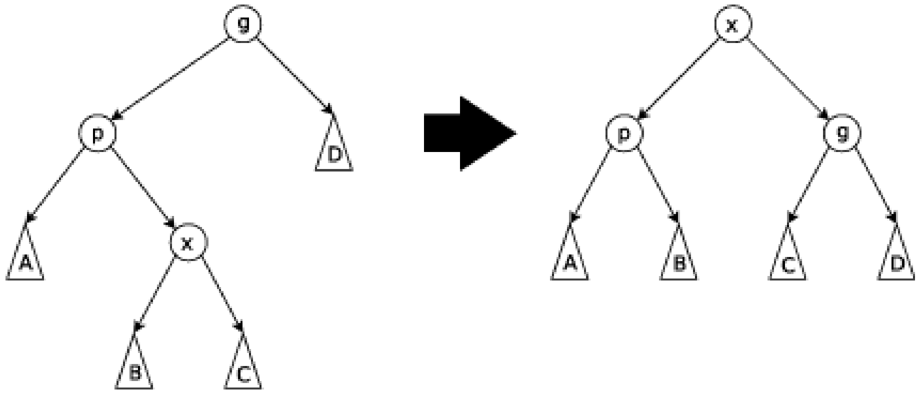
בפעולה $splay$ ־ מצאנו איבר x (או קודם או עוקב אליו), וכעת רוצים להעלות אותו לראש העץ, כלומר, להופכו לשורש. לשם כך, נעזר בשלוש פעולות $Zig - Zig$, $Zig - Zag$ ו ־ Zig. אנו מסתכלים על המסלול בין x לבין

השורש, ומנתחים את צורת המסלול. אם יש לנו "ימינה ימינה" (או באופן שקול - "שמאלה שמאלה") - נבצע פעולת $zig - zig$. אם יש לנו "שמאלה-ימינה" (או לחילופין - "ימינה-שמאלה") - נבצע פעולת $zig - zag$. כל פעולה שכזו מקפיצה את x שתי רמות לכיוון השורש, ושומרת על העץ כעץ חיפוש בינארי. מכיוון שלא מובטח ש - x נמצא ברמה זוגית, נבצע לבסוף פעולת zig שמעבירה אותנו רמה אחת בלבד. אם כן:

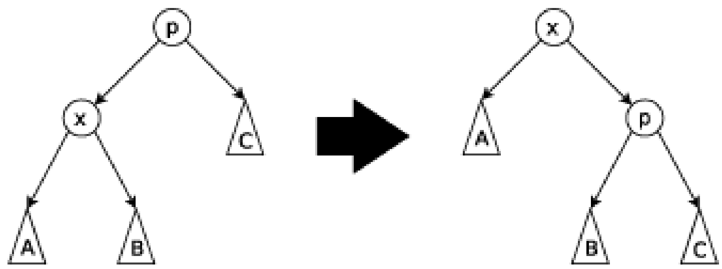
- פעולת $Zig - Zig$: בצורה זו נקפיץ את x שתי רמות למעלה.



- פעולת $Zig - Zag$: שוב, נקפיץ את x שתי רמות למעלה.



- פעולת Zig:



כעת, נרצה להראות שאם ישנם n איברים בעץ, אזי m פעולות לוקחות לכל היותר - $O(m \log n)$, כלומר - $O(\log n)$ בממוצע לכל פעולה. זהו למעשה - ניתוח לשיעורין. כלומר, נראה שבניתוח לשיעורין - פעולת $splay$ לוקחת $O(\log n)$.

8.3 ניתוח פעולת $splay$

ננתח בעזרת שיטת הפוטנציאל. עבור עץ T, נסמן ב־ $\phi(T)$ את פונקציאת הפוטנציאל. נראה שפונקציאת הפוטנציאל תהיה מינימלית כאשר העץ פחות או יותר מאוזן. בנוסף, פונקציית הפוטנציאל תהיה גדולה כאשר העץ לא מאוזן.

לכל קודקוד w בעץ נסמן ב־ $T(w)$ את תת העץ שהשורש שלו ב־ w. נגדיר:

$$\mu(w) \stackrel{\text{def}}{=} \lfloor \log T(w) \rfloor$$
$$\phi(T) \stackrel{\text{def}}{=} \sum_{w \in T} \mu(w)$$

בדקו שהנכם מבינים מדוע פונקציית הפוטנציאל זו חוקית. (מהי הדרישה?).

המחיר לשיעורין של הפעולה $splay$ היא:

$$\hat{c}(splay(T,x)) = c(splay(T,x)) + \phi(T') - \phi(T)$$

כאשר אנו מסמנים ב־ T' את העץ שהתקבל לאחר הפעולה $splay$.

הטענה הבאה אומרת שהמחיר של פעולת $splay$ הוא סכום העלויות לשיעורין של כל פעולת זיג־זיג, זיג־זג ו־זיג שמבצעים במהלך הפעולה $splay$. פורמלית:

טענה 8.1 נניח ש־ $splay(T,x)$ היא סידרה של פעולות $(zig, zig-zag, zig-zig)$ ־ $P_1, \ldots, P_\ell$. אזי העלות לשיעורין של $splay$:

$$\hat{c}(splay(T,x)) = \sum_{i=1}^{\ell} \hat{c}(P_i)$$

הוכחה: נסמן $T = T_0$. T_i המצב של העץ לאחר הפעולה P_i. נקבל:

$$\begin{aligned}
\sum_{i=1}^{\ell} \hat{c}(P_i) &= \sum_{i=1}^{\ell} \left(c\left(p_i\right) + \phi\left(T_i\right) - \phi\left(T_{i-1}\right)\right) \\
&= \sum_{i=1}^{\ell} c(p_i) + \phi(T_\ell) - \phi(T_0) \\
&= c(splay(T,x)) + \phi(T') - \phi(T) = \hat{c}(splay(T,x))
\end{aligned}$$

■

לפי הטענה, מספיק להסתכל על כמה עולה על פעולה $zig-zig$, $zig-zag$ ו־ zig, ולהתבונן במספר הפעולות שישנן לנו.

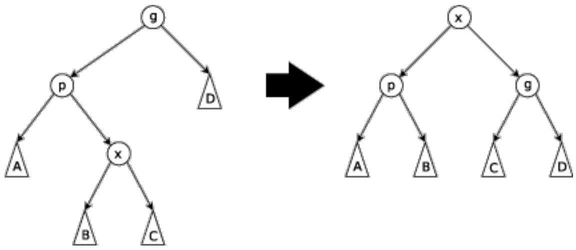
נסתכל על הפעולות $zig-zig$, $zig-zag$ ו־ zig. נסמן ב־ $\mu(x)$ את הפוטנציאל של הקודקוד x לפני הפעולה וב־ $\mu'(x)$ את הפוטנציאל של x אחרי הפעולה. בצורה דומה נסמן את הנ"ל עבור שאר הקודקודים p, g.

8.3.1 ניתוח פעולת zig-zag

טענה 8.2 בכל תת פעולה P_i מסוג $zig-zag$ מתקיים:

$$\hat{c}(zig-zag) \leq 3(\mu'(x) - \mu(x))$$

הוכחה: ניזכר בפעולה:



נזכור כי - $c(zig-zag)=1$. על פי האיור, ניתו לראות כי הפרש הפוטנצילאים הוא:

$$\Delta(zig-zag)=\mu'(x)+\mu'(p)+\mu'(g)-\mu(x)-\mu(p)-\mu(g)$$

(כי כל שאר העצים - A,B,C,D לא השתנו). נשים לב:

1. $\mu'(x)=\mu(g)$. כלומר - הפוטנציאל של x לאחר הפעולה הוא כמו הפוטנציאל של g לפני הפעולה. זאת מכיוון שלפני הפעולה g היה הקודקוד שכל צאצאיו הם כל הקודקודים הנידונים; לאחר הפעולה - x הוא הקודקוד הנ"ל.

2. $\mu(x)\le\mu(p)$: נובע ממבנה העץ. p הוא האבא של - x.

3. $\mu'(x)\ge\mu'(g)$ וגם - $\mu'(x)\ge\mu'(p)$. זאת מכיוון ש - x הוא האבא של g ושל - p, ולכן בהכרח בעל יותר צאצאים מכל אחד מהם בנפרד.

4. $|T'(x)|\ge 2|T'(p)|$ או $|T'(x)|\ge 2|T'(g)|$.

בכדי לראות זאת, ניתן להתבונן על המינימלי מבין $|T'(g)|$ ו - $|T'(p)|$. נניח בלי הגבלת הכלליות ש - $|T'(g)|$ הוא המינימלי מבינהם. מתקיים: $|T'(p)|\ge|T'(g)|$, ולכן:

$$|T'(x)|=|T'(p)|+|T'(g)|+1\ge|T'(g)|+|T'(g)|+1\ge 2|T'(g)|.$$

המקרה השני מתקבל כאשר $T'(p)$ הוא המינימום.

ממקרה זה, כאשר נפעיל $\log$ ונעבור ל - μים, נקבל: $\mu'(x)\ge\mu'(p)+1$ או $\mu'(x)\ge\mu'(g)+1$.

נקבל אם כן:

$$\begin{aligned}
\Delta(zig-zag) &= \mu'(x)+\mu'(p)+\mu'(g)-\mu(x)-\mu(p)-\mu(g)\\
&\overset{(1)}{=} \mu'(p)+\mu'(g)-\mu(x)-\mu(p)\\
&\overset{(2)}{\le} \mu'(p)+\mu'(g)-\mu(x)-\mu(x)\\
&\overset{(3)+(4)}{\le} \mu'(x)+\mu'(x)-1-\mu(x)-\mu(x)\\
&= 2\mu'(x)-1-2\mu(x)=2(\mu'(x)-\mu(x))-1
\end{aligned}$$

הסוגריים מעל השווינים מציינים לפי איזו נקדוה השוויון נכון (ראה למעלה). לכן, העלות לשיעורין של הפעולה - $zig-zag$:

$$\begin{aligned}
\hat{c}(zig-zag) &= c(zig-zag)+\Delta(zig-zag)\le 1+2(\mu'(x)-\mu(x))-1\\
&= 2(\mu'(x)-\mu(x))\le 3(\mu'(x)-\mu(x))
\end{aligned}$$

כאשר האי שיוויון האחרון נכון מכיוון ש - $\mu'(x)\ge\mu(x)$.

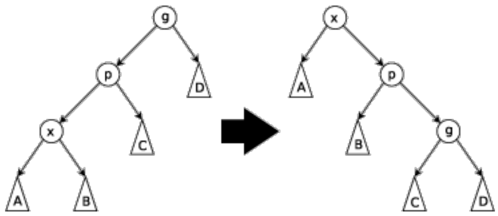
■

8.3.2 ניתוח פעולת zig-zig

טענה 8.3 בכל תת פעולה מסוג $zig-zig$ מתקיים:

$$\hat{c}(zig-zig) \leq 3(\mu'(x)-\mu(x))$$

הוכחה: ניזכר בפעולה:



מתקיים:

$$\Delta(zig-zig) = \mu'(x)+\mu'(p)+\mu'(g)-\mu(x)-\mu(p)-\mu(g)$$

שוב, מתקיים

1. $\mu'(x)=\mu(g)$. x מכיל את כל האיברים ש - g החזיק לפני הפעולה.
2. $\mu'(x)\geq\mu'(p)$ וגם - $\mu(p)>\mu(x)$. לאחר הפעולה - x מכיל יותר איברים מ - p. לפני הפעולה - ההיפך הוא הנכון.
3. נשים לב כי - $|T(x)|+|T'(g)|\leq|T'(x)|$. זאת מכיוון שמתחת ל - x ב- T ישנם העצים A, B. מתחת g ב- T' נמצאים העצים C,D. מתחת x ב- T' נמצאים כל העצים הללו, פלוס שני קודקודים (g,p). אזי מתקיים:

$$\begin{aligned}
|T'(x)| &\geq |T(x)|+|T'(g)| \\
|T'(x)|^2 &\geq (|T(x)|+|T'(g)|)^2 = |T(x)|^2+|T'(g)|^2+2\cdot|T(x)|\cdot|T'(g)| \geq 2\cdot|T(x)|\cdot|T'(g)| \\
\log|T'(x)|^2 &\geq \log\left(2\cdot|T(x)|\cdot|T'(g)|\right) \\
2\log|T'(x)| &\geq \log 2+\log|T(x)|+\log|T'(g)| \\
2\mu'(x) &\geq 1+\mu(x)+\mu'(g) \\
2\mu'(x)-1-\mu(x) &\geq \mu'(g)
\end{aligned}$$

אם כן, נשים הכל יחד ונקבל:

$$\begin{aligned}
\Delta(zig-zig) &= \mu'(x)+\mu'(p)+\mu'(g)-\mu(x)-\mu(p)-\mu(g) \\
&\overset{(1)}{=} \mu'(p)+\mu'(g)-\mu(x)-\mu(p) \\
&\overset{(2)}{\leq} \mu'(x)+\mu'(g)-\mu(x)-\mu(x) \\
&\overset{(3)}{\leq} \mu'(x)+(2\mu'(x)-1-\mu(x))-2\cdot\mu(x) \\
&= 3\left(\mu'(x)-\mu(x)\right)-1
\end{aligned}$$

ולכן:

$$\hat{c}(zig-zig) = c(zig-zig)+\Delta(zig-zig) \leq 1+3\left(\mu'(x)-\mu(x)\right)-1 = 3\left(\mu'(x)-\mu(x)\right)$$

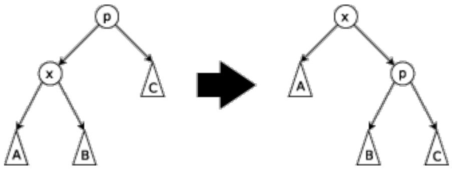
ולכן הטענה מתקיימת.

■

8.3.3 ניתוח פעולת zig

טענה 8.4 $\hat{c}(zig) \leq 3(\mu'(x) - \mu(x)) + 1$

הוכחה: ניזכר בפעולה:



העלות של הפעולה zig היא: $c(zig) = 1$. בנוסף:

$$\Delta(zig) = \mu'(x) + \mu'(p) - \mu(x) - \mu(p) \leq 2\mu'(x) - 2\mu(x) \leq 3(\mu'(x) - \mu(x))$$

כאשר הצעד האחרון נכון מכיוון ש־ $\mu'(p) \leq \mu'(x)$, וכמו־כן: $\mu(x) \leq \mu(p)$. נקבל אם כן:

$$\hat{c}(zig) \leq c(zig) + \Delta(zig) \leq 1 + 3(\mu'(x) - \mu(x))$$

■

8.3.4 סיום הניתוח

אנו מוכנים למשפט העיקרי של חלק זה:

משפט 8.5 $\hat{c}(splay(T,x)) = O(\log n)$.

הוכחה: כזכור, הראינו כי־ $\hat{c}(splay(T,x)) = \sum_{i=1}^{\ell} \hat{c}(P_i)$. ראינו כי לכל $i < \ell$, הפעולה היא $zig - zig$ או $zig - zag$, ולכן:

$$\hat{c}(P_i) \leq 3(\mu'(x) - \mu(x))$$

בנוסף, עבור $i = \ell$, מתקיים: $\hat{c}(P_i) \leq 3(\mu'(x) - \mu(x)) + 1$

נשנה מעט את הנוטציות. נסמן ב־ $\mu_i(x)$ את הפוטנציאל של האיבר x לאחר הפעולה ־ ה־i, עבור $i \in \{1, \ldots, \ell\}$. אם כן, הראינו כי לכל $i < \ell$

$$\hat{c}(P_i) \leq 3(\mu_i(x) - \mu_{i-1}(x))$$

וכמו כן, עבור $i = \ell$ נקבל: $\hat{c}(P_\ell) \leq 3(\mu_\ell(x) - \mu_{\ell-1}(x)) + 1$. נקבל אם כן כי בסה"כ:

$$\hat{c}(splay(T,x)) \leq \sum_{i=1}^{\ell} \hat{c}(P_i) \leq 3(\mu_\ell(x) - \mu_0(x)) + 1 \leq 3\mu_\ell(x) + 1 = 3\lfloor \log |T'| \rfloor + 1 = O(\log n)$$

■

תרגול 10
עצים אדומים שחורים

עצים אדומים שחורים הומצאו ע"י $Bayer$ [1], ב-1972. עץ אדום שחור הוא עץ חיפוש בינארי, שבו כל פעולה במקרה הגרוע עולה $O(\log n)$, כאשר n הוא מספר הנתונים בעץ. הפעולות בהן הוא תומך הן - $insert$, $delete$ ו - $search$.

עצים אדומים שחורים הם כנראה העצים שבהם משתמשים הכי הרבה ביום יום. בספרייה הסטנדרטית של $C++$ (STL) משתמשים ב - rb_tree (כלומר, בעץ אדום שחור) בכדי לממש את מבני הנתונים $std::map$ ו - $std::multi\_map$.

10.1 סקירה כללית

בעץ אדום שחור, כל צומת נצבעת בצבע אדום או שחור (כלומר שומרים סיבית נוספת לכל קודקוד המכילה את צבע הקודקוד).

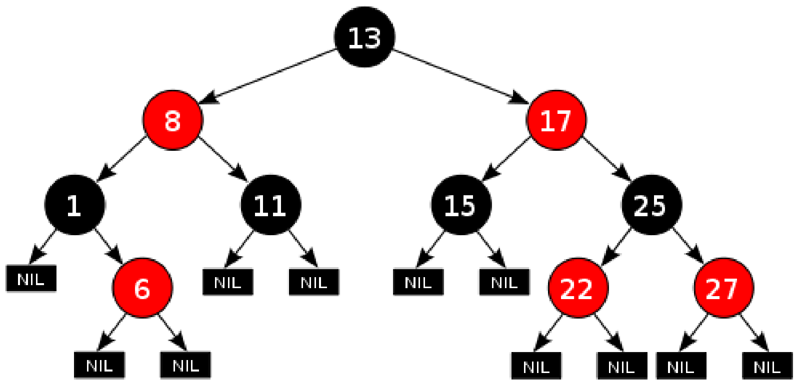
צמתים פנימיים ועלים חיצוניים. לכל צומת בעץ החיפוש נוסיף בנים מדומים, כך שלכל צומת בעץ המקורי יהיו בדיוק 2 בנים. לצמתים המקוריים נקרא "צמתים פנימיים", ולצמתים המדומים נקרא "עלים חיצוניים". נשים לב שכל העלים בעץ שנוצר הם למעשה - עלים חיצוניים.

בעץ אדום שחור - כל הצמתים יהיו צמתים פנימיים, וכל העלים - יהיו ערכי NULL (אבל יהיו צבועים בצבע שחור, ולכן אנו זקוקים להם - כפי שנראה בהמשך). אנו זקוקים לעלים החיצוניים רק לניתוח; במימוש - ניתן להתעלם מהם.

עצים אדומים שחורים. עץ חיפוש בינארי ייקרא "עץ אדום שחור" אם הוא מקיים את התכונות הבאות:

1. כל צומת צבוע ב"אדום" או "שחור".
2. כל עלה (NULL) צבוע ב"שחור".
3. אם צומת הוא אדום, אז שני בניו שחורים.
4. כל המסלולים מכל קודקוד לכל העלים שהם צאצאיו, מכילים את אותו המספר של צמתים שחורים.

דוגמא לעץ אדום שחור:



הגדרה 10.1 (גובה שחור) לכל צומת x נגדיר "גובה השחור" ($black\ hight$) של x, כמספר הקודקודים השחורים בכל מסלול מצומת x ועד לעלה (לא כולל את הצומת x עצמו). נסמן מספר זה ב־ $bh(x)$.

לפי תכונה (4), כל המסלולים מקודקוד x מסויים לכל אחד מהעלים שהם צאצאיו מכילים את אותו מספר הקודקודים שחורים, ולכן הגדרה זו מוגדרת היטב.

דוגמא. בדוגמא הנתונה לנו, נרשום את כל הקודקודים x כך ש:

- $bh(x)=0$: קודקודים אלו הם כל העלים (NIL).
- $bh(x)=1$: הקודקודים $1,6,11,15,22,25,27$ (נשים לב שהגדרת $bh(x)$ אומרת כל הקודקודים השחורים עד לעלה ־ מלבד הקודקוד x עצמו), ולכן גם הצמתים ־ 1, 11, 15, 25 הם בעלי $bh(x)=1$ אף על פי שהם שחורים בעצמם.
- $bh(x)=2$: $8,17,13$.

10.2 ניתוח גובה עץ אדום שחור

הכללים על עצים אדומים שחורים מספיקים בשביל לנתח את גובה העץ, זאת עוד לפני שראינו כיצד מממשים ושומרים על תכונות וכללים אלו.

טענה 10.2 לכל קודקוד x בעץ אדום שחור, תת העץ המושרש ב־ x מכיל לפחות $2^{bh(x)}-1$ צמתים פנימיים.

הוכחה: ההוכחה היא באינדוקציה על גובהו של x. אם x הוא עלה, אזי $bh(x)=0$, ותת העץ שלו אינו מכיל צמתים פנימיים. הטענה אומרת שתת העץ מכיל $2^0-1=0$ צמתים פנימיים, ולכן הטענה מתקיימת.
צעד האינדוקציה: נניח ש־ x הוא בעל גובה חיובי. נתבונן בתת עץ ששורשו ב־ x. מכיוון ש־ x אינו עלה, יש לו שני בנים, נניח y,z. אזי:

- אם y אדום, אזי $bh(y)=bh(x)$.
- אם y שחור, אזי $bh(y)=bh(x)-1$ (שכן, y שחור, ואינו נספר ב־ $bh(y)$, אבל כן נספר ב־ $bh(x)$).

הנ"ל נכון גם ל־ z. לסיכום, נוכל לומר כי $bh(y)=bh(z)\geq bh(x)-1$.
על כל פנים, גם y וגם z, שניהם בעלי רמה נמוכה יותר מהרמה של ־ x, ולכן ניתן להפעיל עליהם את הנחת האינדוקציה. לפי ההנחה, $|T(y)|\geq 2^{bh(y)}-1$. לפי המסקנה שראינו, נקבל כי $|T(x)|\geq 2^{bh(x)-1}-1$, ובאופן דומה $|T(z)|\geq 2^{bh(x)-1}-1$. נקבל אם כן:

$$|T(x)|=|T(y)|+|T(x)|+1\geq 2\cdot\left(2^{bh(x)-1}-1\right)+1=2^{bh(x)}-2+1=2^{bh(x)}-1$$

וזה בדיוק מה שהיינו צריכים להוכיח. ■

הטענה הבאה היא ליבו של ניתוח עץ אדום שחור, ומראה שעצים אדומים שחורים הם בעלי רמה נמוכה ויחסית מאוזנים, מה שיראה שביצוע הפעולות הוא $O(\log n)$ במקרה הגרוע ביותר.

טענה 10.3 גובהו של עץ אדום שחור המכיל n צמתים פנימיים הוא לכל היותר: $2\log(n+1)$.

הוכחה: ההוכחה פשוטה ומבוססת על הטענה הקודמת. נתבונן בעץ אדום שחור כלשהו. נתבונן בשורש, x, ונניח שבעץ h רמות. לפי כלל (3) (לכל קודקוד אדום - שני בניו שחורים), נקבל שבכל מסלול פשוט מהשורש לעלה יש לכל הפחות $h/2$ צמתים שחורים, ולכן $bh(x) \geq h/2$. לפי הטענה הקודמת, $|T(x)| \geq 2^{bh(x)} - 1$. נשים לב ש - $|T(x)| = n$, ולכן:

$$\begin{aligned} n &\geq 2^{bh(x)} - 1 \\ n+1 &\geq 2^{bh(x)} \geq 2^{h/2} \\ \frac{h}{2} &\leq \log(n+1) \\ h &\leq 2\log(n+1) \end{aligned}$$

■

10.3 רוטציות

הרוטציה היחידה שנבצע בעץ אדום שחור היא כמו הפעולה zig בעצי $splay$. תהיה לנו רוטציה ימנית, ורוטציה שמאלית.

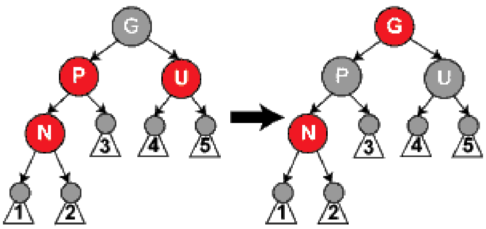
10.4 הכנסה

ההכנסה של איבר x לעץ אדום שחור T תעשה באופן הבא:

- חפש את x ב - T, והכנס את x כמו בעץ חיפוש בינארי רגיל.
- נצבע את הקודקוד באדום.
- נתקן את העץ להיות שוב עץ אדום שחור.

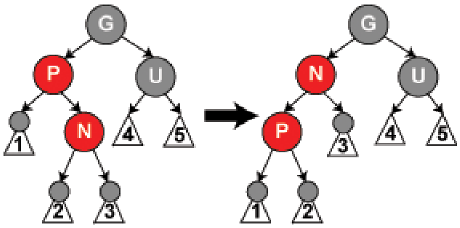
התיקון היא הפעולה המסובכת. נחלק למספר מקרים:

- אם אין לקודקוד הנכנס אבא - כלומר הקודקוד נכנס בתור שורש העץ: במקרה זה אין הפרה של איזשהו כלל.
- אם אביו של הקודקוד (הנכנס) הוא בצבע שחור - אין שום בעיה, פשוט "דחפנו" קודקוד אדום בין שני קודקודים שחורים. שום תכונה לא ניזוקה (בדוק!).
- נותרנו עם המקרה בו אביו של הקודקוד הנכנס הוא אדום. במקרה זה נפריד בין שלושה מקרים:
  - אם דודו של הקודקוד הנכנס הוא גם אדום, סבו של הקודוקוד הוא שחור (אחרת תכונה 3 מופרת).



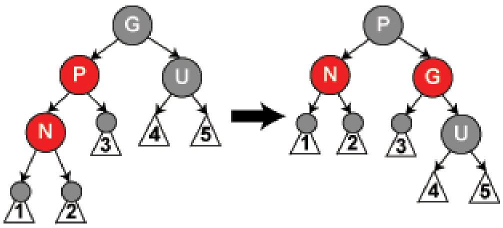
במקרה זה נוכל פשוט להחליף את הצבעים של הרמה של האבא והדוד, עם הצבע של הסבא. בצורה זו, נשמור על תכונה 4. אך, כעת ייתכן שהקודקוד G הפר את תכונה 3 (במקרה שאביו אדום). לכן, נמשיך הלאה לכיוון השורש כאילו G הוא הקודקוד אותו הכנסנו.

– במקרה שדוד של הקודוד החדש הוא שחור, לא נוכל להחליף בין הצבעים של האבא והדוד ובין הצבע של הסבא; במקרה שהקודקוד החדש הוא בן ימני של האבא, נבצע רוטציה שמאלית.



בצורה זו העלנו את N (הקודקוד החדש) רמה אחת למעלה. נשים לב שתכונה 4 אינה מופרת, אך תכונה 3 עדיין מופרת. כעת, נתייחס לקודקוד P כאל הקודקוד החדש, ובעצם נקבל את המקרה הבא:

– במקרה זה הקודוק החדש הוא בן שמאלי של האבא, והדוד הוא בצבע שחור, נבצע רוטציה של הקודוק P ימינה.



נשים לב שהחלפנו את הצבע של P ו - G. כל הדרישות מתקיימות.

דוגמא להכנסה - עמוד 245 בספר (מבוא לאלגוריתמים, האוניברסיטה הפתוחה, כרך א', מהדורה א').

תרגול 12
Universal Hashing and Perfect Hash

12.1 הערה כללית לפני שמתחילים

ישנן שתי שיטות לדבר על פונקציית $hash$: הראשונה שפונקציית הערבול היא קבועה, אבל המפתחות הן אקראיות. הבעיה בשיטה זו היא ההנחה שבעצם ישנה התפלגות מסויימת על המפתחות. בד"כ מדברים על התפלגות אחידה. הנחה זו בעייתית שכן, ייתכן מאוד שההתפלגות אינה התפלגות אחידה, ובכלל, להניח התפלגות כלשהי על המפתחות היא בעייתית.

השיטה השנייה היא לדבר על מפתחות קבועים, אך פונקציית הערבול נבחר באקראי. אנו הולכים לדבר בעיקר על השיטה הזו.

12.2 הקדמה

נתבונן בבעיה הבאה:

נתונה קבוצה של מספרים $U = \{0, \ldots, u-1\}$, ותת קבוצה $S = \{x_1, \ldots, x_n\} \subseteq U$. נרצה לענות על השאילתות הבאות:

- $query(x)$ ־ האם $x \in S$. אם יש מידע ־ החזר (כלומר, x הוא רק מפתח).
- $insert(x)$ ־ הכנס את x ל־ S.
- $delete(x)$ ־ מחק את x מ־ S.

פתרון. פתרון אפשרי לבעיה הוא:

- מערך בגודל U. מעין $bit-vector$. הבעיה ־ אם $|S| << |U|$, יש לנו המון בזבוז של מקום...
- פונקציית $hash$.

בבואינו לעסוק כעת בפונקציית $hash$, נתעלם משתי הפעולות האחרונות (הכנסה והוצאה), ונדבר רק על הפעולה $query$. למעשה, נדבר על מבנה נתונים **סטטי**.

כיצד נבחר פונקציית $hash$ טובה? נניח $|S| = n$. אנו רוצים לבחור פונקציה שלוקחת אלמנטים מ ־ U ומעבירה אותם ל ־ $\mathbb{Z}_m$ (למערך בגודל m. עוד לא הגבלנו את גודל m, אך נרצה להגיע למשהו שהוא לינארי ב ־ n (כדי שסיבוכיות המקום תהיה לינארית)). פונקציה טובה כזו היא פונקציה אקראית לחלוטין. כלומר, לכל איבר ב ־ U, נבחר איבר באקראי מהקבוצה $\{0, \ldots, m-1\}$. עבור שני איברים x, y, קבועים, ההסתברות להתנגשות היא:

$$\Pr_h[h(x) = h(y)] = \frac{1}{m}$$

כאשר ההסתברות נלקחת על הבחירה של h (ניתן להסתכל על זה בצורה כזו ־ h נבחר באקראי, בהתפלגות אחידה, מבין כל הפונקציות הממפות את U ל ־ $\mathbb{Z}_m$). הסבר לחישוב ההסתברות: מתבוננים במספר התא ש

x ־ הגיע אליו, ושואלים מהי ההסתברות ש ־ y יגיע לאותו התא. מכיוון שהתא אליו y מגיע נבחר באקראי מבין כל התאים, נקבל הסתברות של $1/m$.

בכלל, עבור k איברים מתוך S, נניח $s_1, \ldots, s_k$, ההסתברות שכולם ייתנגשו לאותו התא היא:

$$\Pr_h[h(x_1) = \ldots = h(x_k)] = \left(\frac{1}{m}\right)^{k-1}$$

כאשר ההסתברות נלקחת על הבחירה של h. הסבר לחישוב ההסברות: x_1 קובע תא שאליו כולם צריכים ללכת. ההסתברות שכל אחד מ ־ $x_2, \ldots, x_k$ נפלו בתא ש ־ x_1 נפל אליו היא $1/m$. מכיוון שכל המאורעות בלתי תלויים, נקבל $(1/m)^{k-1}$.

הבעיה. כזכור, בחרנו פונקצייה אקראית בשביל פונקציית הערבול ($hash$). כדי לייצג את הפונקציה, לכל ערך ב־ U נצטרך לשמור לאיזה תא הוא ממופה, או בעצם ־ ערך ב ־ $\mathbb{Z}_m$. בכדי לייצג ערך ב ־ $\mathbb{Z}_m$ דרושים לנו $\log m$ ביטים (למה?), ולכן, כדי לייצג את כל הפונקצייה אנו נדרשים ל ־ $|U| \log m$ ביטים, שזה המון!

12.2.1 דרישות

למעשה, נרצה מספר תכונות מפונקציית ה ־ $hash$:

- **אתחול:** בהינתן קבוצה S נרצה לבנות בצורה מהירה.
- **זמן:** נרצה לענות על שאילתות בצורה מהירה.
- **מקום:** פונקציית ה ־ $hash$ צריכה להיות בעלת תיאור קומפקטי. כלומר, ייצוג הפונקצייה צריך להיעשות בעזרת מספר מועט של ביטים.

12.3 פונקציית גיבוב מושלמת ־ Perfect Hash

המטרה בסופו של דבר היא להגיע לפונקצייה גיבוב מושלמת. כלומר:

הגדרה 12.1 בהינתן קבוצה $S \subseteq U$, נאמר שהפונקציה ־ $h : U \to \mathbb{Z}_m$ היא פונקציית גיבוב מושלמת אם לכל $x \neq y \in S$, מתקיים $h(x) \neq h(y)$ (כלומר, אין התנגשויות בכלל).

12.4 פונקציית hash אוניברסלית

כאמור, נדבר על משפחה של פונקציות, ונבחר פונקציה אחת מתוך המשפחה באקראי. בסעיף הקודם, דיברנו על כלל הפונקציות $U \to \mathbb{Z}_m$, ואמרנו ש ־ h נבחרת באקראי מתוכם. עבור משפחה זו, ראינו שקיימות שתי תכונות:

- לכל $x, y \in U$, $x \neq y$, מתקיים:

$$\Pr_h[h(x) = h(y)] = \frac{1}{m}$$

- ובאופן כללי, לכל $x_1, \ldots, x_k \in U$, $x_1 \neq \ldots \neq x_k$, מתקיים:

$$\Pr_h[h(x_1) = \ldots = h(x_k)] = \left(\frac{1}{m}\right)^{k-1}$$

אמרנו שמשפחה זו בעייתית מכיוון שאין לפונקציית הגיבוב ייצוג קומפקטי. נתבונן כעת במשפחה חדשה של פונקציות, המתקיימת רק את התכונה הראשונה. לאחר מכן, נראה שתכונה זו מספיקה לנו, וכמו־כן, שקיימת משפחת פונקציות כאלה בעלי ייצוג קומפקטי. פורמלית, נגדיר:

הגדרה 12.2 משפחת פונקציות $hash$ $\mathcal{H} = \{h_i \mid h_i : U \to \mathbb{Z}_m\}$ תקרא משפחה אוניברסלית אם לכל $x, y \in U$, $x \neq y$ מתקיים:

$$\Pr_{h \in \mathcal{H}} [h(x) = h(y)] \leq \frac{1}{m}$$

כאשר ההסתברות נלקחת על פני בחירת h מתוך $\mathcal{H}$.

למעשה, המשפחה הקודמת שהגדרנו (כלל הפונקציות מ ־ U ל ־ $\mathbb{Z}_m$) הייתה עמידה בפני התנגשויות לכל תת קבוצה של איברים. אצלינו ־ אנחנו עמידים בפני התנגשות רק לזוג.

השאלה הראשונה היא האם קיימת משפחה של פונקציה כזאת. השאלה היא השנייה ־ האם ייצוגה קומפקטי, והשאלה השלישית היא.... למה היא באמת טובה לנו??

ראשית, קיימת משפחה כזאת של פונקציות. נתבונן במשפחה הבאה:

$$\mathcal{H}_{p,m} = \{h_{a,b} \mid 1 \leq a \leq p-1, 0 \leq b \leq b-1\}$$

וכאשר:

$$h_{a,b} = ((ax + b) \bmod p) \bmod m$$

כמה פונקציות במשפחה? עבור p מסויים, יש כאן p^2 פונקציות. הטענה אומרת שהמשפחה הנ"ל היא משפחה אוניברסלית. כלומר, אם a ו ־ b נבחרים באקראי כפי שצויין, התכונה שביקשנו ־ מתקיימת. ההוכחה היא בספר, מבוא לאלגוריתמים, קורמן, **מהדורה שנייה** (איני בטוח שיש תרגום לעברית של המהדורה הזו..).

נשים לב שייצוג הפונקציה הוא קומפקטי ־ בכדי לייצג את הפונקצייה דרושים לנו רק a ו־ b, שלשניהם אנו צריכים $\log p$ ביטים.

12.5 בניית פונקציית Perfect Hash עבור $S \subseteq U$

הבנייה (האלגוריתם) ייעשה בצורה הבאה:

- **קלט:** הקבוצה $S \subseteq U$ (נניח, נתונה לנו רשימה מקושרת של כל האיברים).
- **פלט רצוי:** מערך מגודל m, ופונקציה $h : U \to \mathbb{Z}_m$, כל שלכל זוג $x \neq y \in S$, $h(x) \neq h(y)$.
- **האלגוריתם:**
  1. נקח משפחה אוניברסילית בצורה שרירותית (התלויה ב־ m).
  2. נבחר $h \in \mathcal{H}$ בצורה אקראית.
  3. ניצור "קבוצות" (רשימות) של $S_i = \{x \in S \mid h(x) = i\}$. (כלומר, לכל אינדקס, בודקים כמה איברים מופו לאותו האינדקס).
  4. אם קיים i כך ש ־ $|S_i| > 1$, חזור ל (2). (אם קיימת איזושהי התנגשות ־ בחר פונקציה מחדש).

ניתוח מספר ההתנגשויות. אנו מעוניינים לחשב מהי ההסתברות שכאשר בחרנו פונקציית $hash$ מתוך $\mathcal{H}$, נקבל איזושהי התנגשות. נקבל: (הסבר על החישוב מובא לאחר החישוב)

$$\Pr\left[\exists\ collision\right] = \Pr\left[\exists x, y \in S, x \neq y, h(x) = h(y)\right] \leq \sum_{x,y \in S, x \neq y} \Pr\left[h(x) = h(y)\right] \leq \binom{n}{2} \cdot \frac{1}{m} \leq \frac{n^2}{2m}$$

מספר הערות על חישוב זה:

- בחישוב זה נעזרנו ב־ $union\ bound$. הכלל אומר כי: $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$. קל להוכיח אי שיוויון זה בעזרת ציור. ניתן להכליל את האי שיוויון לכל מספר אירועים (סופי) באינדוקציה.

  אצלינו, שאלנו מהי ההסתברות שקיים איזשהו זוג ב ־ S עבורו יש התנגשות. השאלה "האם קיים זוג שמתנגש" שקולה לשאלה "האם הזוג x_1 ו ־ x_2 מתנגשים **או** האם הזוג x_1 ו ־ x_3 מתנגשים **או**...".

בשאלה זו ישנם הרבה "או" - מה שמוביל אותנו ל - $union\ bound$. מכיוון שכל שההסתברות ש - x_1 ו - x_2 יתנגשו, היא כמו ההסתברות ש x_2- ו - x_3 ייתנגשו, וכן הלאה, נקבל את אי השיוויון השני. (בעצם, סכום ההסתברות שכל זוג כלשהו מתנגש).

- באשר לאי השיוויון השלישי - אנו בודקים את כל הזוגות x, y האפשריים שיש, ללא חזרות. ישנן $\binom{n}{2}$ זוגות כאלו. עליהן, אנו מחשבים מה ההסתברות שיש התנגשות בין כל זוג וזוג. ההסתברות להתנגשות של כל זוג היא קטנה מ - $1/m$, וזה נובע מתכונת האוניברסליות של $\mathcal{H}$.

אם ניקח $m = n^2$, נקבל כי:

$$\Pr\left[\exists\ collision\right] \leq \frac{n^2}{2m} = \frac{n^2}{2n^2} = \frac{1}{2}$$

כלומר בהסתברות קטנה מחצי - תהיה בעיה.

ניתוח זמן ריצה. בכל סיבוב מבצעים עבודה התלויה באורך של $|S| = n$. השאלה היא כמה סיבובים ישנם. מכיוון שאנחנו עובדים באלגוריתם הסתברותי, זמן הריצה של האלגוריתם יכול להמשיך עד אינסוף (אף פעם לא נמצא פונקציה שאין בה התנגשות...), אבל באופן כללי, אנו **מצפים** שמספר הסיבובים יהיה קטן. למעשה, אינטואיטיבת, מכיוון שההסתברות שקיימת התנגשות לפונקציה אקראית קטנה מ - $1/2$, אנו מצפים שנצטרך לבדוק 2 פונקציות עד שנקבל פונקציה טובה.

תוחלת. באופן פורמלי יותר אנו נאלצים להציג את מושג ה"תוחלת".

הגדרה 12.3 (תוחלת - $expected\ value$) יהי X משתנה מקרי, המקבל ערכים $x_1, x_2, \ldots$. אזי, תוחלת המשתנה X היא:

$$E(X) = \sum_i \Pr[X = x_i] \cdot x_i$$

תוחלת היא למעשה - הערך אותו אנו מצפים לקבל. לדוגמא, נניח אנו מטילים מטבע. בהסתברות $1/2$ הוא "עץ", ובהסתברות $1/2$ הוא "פאלי". אנו משחקים במשחק הבא: אנו מטילים את המטבע שוב ושוב, עד אשר מקבלים "עץ". אם קיבלנו "עץ" - נפסיק את המשחק. אם קיבלנו "פאלי" - נמשיך לעוד סיבוב. כמה הטלות אנו מצפים שנבצע עד שנקבל "עץ"? אינטואיטיבית, התשובה היא 2. נחשב:

- ההסתברות שנקבל עץ אחרי הטלה אחת היא $1/2$.
- ההסתברות שנקבל עץ אחרי 2 הטלות היא $1/4$ (פאלי בהטלה הראשונה, ועץ בשנייה).
- ההסתברות שנקבל עץ אחרי 3 הטלות היא $1/8$ (פאלי בשתי ההטלות הראשונות, ועץ בשלישית).
- ההסתברות שנקבל עץ אחרי i הטלות, היא $1/2^i$.

ולכן, תוחלת מספר הסיבובים של המשחק:

$$\begin{aligned}
E(\#\ of\ rounds) &= \sum_{i=1}^{\infty} \frac{1}{2^i} \cdot i = \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + 4 \cdot \frac{1}{16} + 5 \cdot \frac{1}{32} + \ldots \\
&= (\frac{1}{2} + \frac{1}{4} + \ldots) + (\frac{1}{4} + \frac{1}{8} + \ldots) + (\frac{1}{8} + \frac{1}{16} + \ldots) \\
&= 1 + \frac{1}{2} + \frac{1}{4} + \ldots = 2
\end{aligned}$$

נחזור לאלגוריתם. כאמור, ההסתברות שנצטרך לחזור לשלב (2) היא קטנה מ - $1/2$. לכן, נקבל:

$$E(number\ of\ rounds) = \sum_{i=1}^{\infty} i \cdot \Pr[number\ of\ rounds = i] \le \sum_{i=1}^{\infty} i \cdot \frac{1}{2^i} = 2$$

כלומר, אנו **מצפים** שלאחר שני סיבובים, האלגוריתם יעצור.
לסיכום, קיבלנו פונקציית $hash$ מושלמת. הבנייה (צפוייה) ב - $O(n)$, הפונקציה יעילה, קלה לחישוב, וקומפקטית. הבעיה היחידה - אנו צריכים $O(n^2)$ מקום!

12.6 FKS - Fredman Komlos Szemeredi[1]

(מבוסס על המאמר - [3]). המטרה היא לבחור m שיהיה בגודל לינארי ב - n. נתבונן במימוש הבא:

1. נבחר משפחה אוניברסלית $\mathcal{H}$ בצורה שרירותית (התלויה ב - m, כאשר m יהיה לינארי ב - n).
2. נבחר פונקצייה $h \in \mathcal{H}$ בצורה אקראית.
3. נבדוק כמה התנגשויות יש: ניצור "קבוצות" כך ש - $S_i = \{x \in S \mid h(x) = i\}$.
4. אם מספר ההתנגשויות הכולל $\le n$, חזור ל - (2).
5. לכל i, אם $|S_i| \le 1$, לא עושים כלום. אחרת:

   (א) נבחר $\mathcal{H}_{p,|S_i|^2}$ ונבחר מתוכה פונקציה אקראית h_i. נבדוק אם h_i פונקציה מושלמת.
   (ב) אם h_i פונקציה מושלמת, נקח אותה.
   (ג) אחרת - נחזור ל - (5א).

נתבונן על שלב (4) ונשאל כמה פעמים אנו חוזרים לשלב (2). לשם כך, נחשב: תוחלת מספר ההתנגשויות:

$$E(\#\ collisions) = \sum_{x,y \in S, x \ne y} \Pr[h(x) = h(y)] \le \binom{n}{2} \cdot \frac{1}{m} \le \frac{n^2}{2m}$$

כעת, אם נבחר $m = n$, ונשתמש באי שיוויון מרקוב ($\Pr[X \ge a] \le E(x)/a$), נקבל:

$$\Pr\left[\#\ collisions \ge n\right] \le \frac{E(\#\ collisions)}{n} \le \frac{n^2}{2mn} = \frac{1}{2}$$

כפי שכבר ראינו, הנ"ל מראה שאנו מצפים שנחזור פעמיים לשלב (2) עד שנמצא פונקציה טובה.
כעת, נרצה לבדוק כמה פעמים חוזרים על שלב (5). נזכור שאנו מגיעים לשלב (5) רק כאשר מספר ההתנגשויות הכולל קטן מ - n. זמן סיבוב 5א הוא $O(|S_i|^2)$ (בתוחלת). אם נחשב סה"כ, כל הסיבוכים הנ"ל:

$$\sum_{i=1}^{m=n} |S_i|^2 \le 4 \sum_{i=1}^{n} \binom{|S_i|}{2} = 4 \cdot (\#collisions) \le 4 \cdot n$$

כאשר השיווין השלישי נכון מכיוון שמספר ההתנגשויות ב - S_i הוא $\binom{|S_i|}{2}$.
קיבלנו אם כן, מבנה נתונים שזמן הבנייה שלו היא בתוחלת $O(n)$, גישה - $O(1)$. גודל מבני הנתונים:

$$O(\sum_{i=1}^{n} |S_i|^2) = O(\#\ of\ collisions) = O(n)$$

---
[1] לידע כללי בלבד....

ביבליוגרפיה

[1] R. Bayer. Symmetric binary B-Trees: Data structure and maintenance algorithms, In *Acta Informatica* 1:290–306, 1972.

[2] T. Cormen, C. Leiserson, R. Rivest and C. Stein: Introduction to Algorithms (2nd ed.). *MIT Press* and *McGrew-Hill*, 2001.

[3] M. Fredman, J. Komlos, E. Szemere'di Storing a Sparse Table with $O(1)$ Worst Case Access Time. In *Jorunal of the ACM*, July 1984, 31(3), 538–544.

[4] W. Pugh. Skip lists: a probabilistic alternative to balanced trees, In *Communications of the ACM*, June 1990, 33(6) 668–676.

[5] D. Sleator and R. E. Tarjan. Self-Adjusting Binary Search Trees, In *Journal of the ACM* , 1985, 32 (3) 652-–686. 668–676.