

# מבני נתונים - תרגול 5

## רשימת דילוגים - Skip-List

גלעד אשרוב

12 באפריל 2013

### תקציר

בתרגול זה נלמד מבנה נתונים חדש המשלב מעין חיפוש בינארי עם רשימה מקושרת. כך, אנו מקבלים גם את הגמישות של רשימה מקושרת (בכל הקשור להכנסת איבר לאמצע הרשימה), בנוסף לחיפוש וגישות מהירות אל איברים בתוך הרשימה, כמו במערך.

## 1 מוטיבציה

בניגוד למבני נתונים שנלמד בהמשך הקורס, שבהם נתקל בפרטים קטנים שלפעמים קשים לזכירה - רשימת דילוגים הוא מבני נתונים פשוט מאוד, וקל לזכור כיצד הוא פועל (אם כי, ניתוחי זמן הריצה עלולים להיות מעט מסובכים). מבני הנתונים הזה הוא כנראה גם החדש ביותר שנלמד בקורס, ופורסם ע"י Pugh ב-1990. ראה [1]. נתבונן בפעולה -  $search(x)$  - מצא את האיבר  $x$  במבנה הנתונים. אם נשמור את הנתונים בתוך מערך - ונשמור תמיד שהמערך יהיה ממוין, חיפוש יעלה לנו  $O(\log n)$  (ניתן לבצע חיפוש בינארי). 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. זהו הרעיון שעומד מאחורי רשימת דילוגים. מימוש מעין *random access* 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. נתבונן בפעולה זו ברשימה מקושרת ממויינת, ונרצה להוסיף "מידע נוסף" שיעזור לנו לבצע את החיפוש. נעיר שהסיבה היחידה שבה אנו מדברים על רשימת דילוגים ולא על מערך היא מכיוון שאנו בכל זאת רוצים לדבר על *insert* ו-*delete*, ולכן נרצה בהחלט לתמוך בפעולות אלו. בכל אופן, בשלב ראשון, בזמן שאנו חושבים רק על הרעיון שמאחורי מבני הנתונים, נתמקד רק ב-*search*.

## 2 רעיון ראשון - שתי רשימות

כאשר אנחנו מבצעים חיפוש בינארי במערך, אנחנו נכנסים לאיברים באמצע המערך. איך ניתן לעשות זאת ברשימה מקושרת? הרעיון הראשון הוא להחזיק - שתי רשימות מקושרות. 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 - תרגיל(?)). 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:

$$T(\text{search}_2) = |L_2| + \frac{|L_1|}{|L_2|}$$

כלומר, במקרה הגרוע ביותר אנו עוברים על כל רשימת הנציגים. לאחר מכן, אנו עוברים על חלק מהרשימה המלאה. מכיוון שחילקנו 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? בכדי לגלות - נגזור את הביטוי. נסמן את  $|L_1|$  ב-  $n$  (מספר האיברים ברשימה הגדולה), ואת  $|L_2|$  ב-  $k$ .

$$T(\text{search}_2) = k + \frac{n}{k}$$

כאמור, אנחנו רוצים לדעת מהו מספר הנציגים האופטימלי, ולכן נגזור את הביטוי לפי  $k$ . נקבל:

$$T'(\text{search}_2) = 1 - \frac{n}{k^2}$$

נשווה ל-0, ונקבל כי:  $k^2 = n$ , כלומר,  $k = \sqrt{n}$ . במילים אחרות, כאשר רשימת הנציגים היא  $\sqrt{n}$ , עלות החיפוש היא:  $\sqrt{n} + n/\sqrt{n} = 2\sqrt{n}$ , וכאשר עוברים בין מנציג אחד לשני מדלגים על  $\sqrt{n} = n/\sqrt{n}$  איברים אמיתיים. למעשה, הצלחנו להוריד את זמן החיפוש מ-  $n = |L_1|$  ל-  $2\sqrt{n}$ . 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.

## 2.1 Ideal Skip List

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

$$T(\text{search}_3) = |L_3| + \frac{|L_2|}{|L_3|} + \frac{|L_1|}{|L_2|}$$

ביטוי זה מקבל מינימום כאשר  $|L_3| = \sqrt[3]{|L_1|}$ ,  $|L_2| = \sqrt[3]{|L_1|^2}$  ואז נקבל:  $T(\text{search}_3) = 3\sqrt[3]{n}$ . אם נמשיך ונבנה עוד ועוד רמות, נקבל בצורה כללית:

$$\begin{aligned} T(\text{search}_1) &= n \\ T(\text{search}_2) &= 2 \cdot \sqrt{n} \\ T(\text{search}_3) &= 3 \cdot \sqrt[3]{n} \\ T(\text{search}_4) &= 4 \cdot \sqrt[4]{n} \\ &\vdots \\ T(\text{search}_k) &= k \cdot \sqrt[k]{n} \\ &\vdots \\ T(\text{search}_n) &= n \cdot \sqrt[n]{n} = n \end{aligned}$$

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? טוב, בכדי להביא למינימום, נגזור את הביטוי (לפי  $k$ ) ונשווה ל-0. נקבל:

$$\begin{aligned} T(\text{search}_k) &= k \sqrt[k]{n} = k \cdot n^{1/k} \\ T'(\text{search}_k) &= n^{1/k} + k \cdot \ln n \cdot n^{1/k} \cdot \left(-\frac{1}{k^2}\right) \\ T'(\text{search}_k) &= n^{1/k} + k \cdot \ln n \cdot n^{1/k} \cdot \left(-\frac{1}{k^2}\right) \\ T'(\text{search}_k) &= n^{1/k} - \frac{\ln n \cdot n^{1/k}}{k} \end{aligned}$$

נשווה ל-0 ונקבל:

$$\begin{aligned} n^{1/k} &= \frac{\ln n \cdot n^{1/k}}{k} \\ k &= \ln n \end{aligned}$$

כלומר, הביטוי  $k \cdot \sqrt[k]{n}$  מקבל מינימום כאשר  $k = \log n$ . במקרה זה נקבל כי  $\log n \cdot \sqrt[\log n]{n} = T(\text{search}_{\log n}) = \log n \cdot 2$ , נקבל כי:  $\log n \cdot 2$ . 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. המצב "יתקלקל" כאשר נגיע ל-*insert* ו-*delete*. לסיכום, חיפוש איבר ברשימה ייתבצע באופן הבא:

### חיפוש ( $x$ ).

1. נחפש את  $x$  ברשימה העליונה, ונעצור כאשר עברנו את  $x$ . נלך צעד אחד שמאלה ונקבל את ה"קודם" של  $x$  ברשימה העליונה.
2. נלך ל"בן" של הקודם של  $x$  (כלומר, לאיבר שמתחתיו). הגענו לרשימה השנייה.
3. נחפש את  $x$  ברשימה השנייה כפי שחיפשנו ברשימה הראשונה, רק שנתחיל את החיפוש באיבר שאיתו הגענו לרשימה זו (בנו של הקודם של  $x$  ברשימה הראשונה).

4. נמשיך את ההליך עד שנגיע רמה התחתונה. ברמה זו, או שנמצא את  $x$ , או שנראה שאינו קיים ונוכל להחזיר את הקודם שלו.

קל לראות שברשימת דילוגים "אינדאלית", זמן הריצה של חיפוש הוא  $\Theta(\log n)$ .

### 3 רשימת דילוגים אקראית

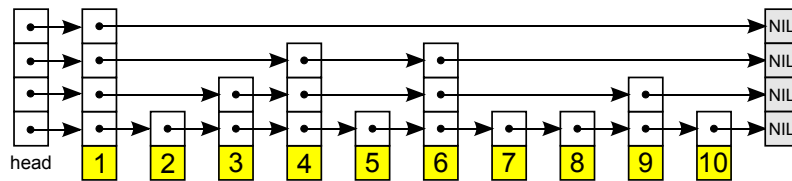
לפני שנתחיל בהכנסה, לשם נוחות - נוסיף איבר דמיוני שיהיה בתחילת כל רשימה - את האיבר  $-\infty$ . איבר זה ייקל עלינו בהמשך.

הכנסה היינו מבצעים בצורה הבאה: בהינתן איבר  $y$  שאותו נרצה להכניס, נחפש קודם את  $y$  במבנה הנתונים. אלגוריתם החיפוש שהצגנו למעלה, יביא לנו את הקודם של  $y$  ברמה התחתונה ביותר, ונוכל להכניס את  $y$  לרמה התחתונה.

אבל, כאשר נעשה זאת, נקבל שהפרנו את האיזון של מבני הנתונים. כעת, המרחק בין שני איברים הוא לא "אחיד", וייתכן שנצטרך להעלות את  $y$  רמות למעלה. אבל - כמה רמות נעלה אותו? את מי נעלה?

**רעיון:** נטיל מטבע. אם המטבע יוצא "עץ" - נעלה את  $y$  רמה. אם הוא יוצא "פלי" - סיימנו את ההכנסה. נניח יצא "עץ" והעלנו את  $y$  רמה, שוב ניתקל באותו הבעיה - האם להמשיך להעלות את  $y$ , או לעצור. שוב, נקבל את ההחלטה ע"י מטבע. למעשה, אנחנו מטילים את המטבע שוב ושוב, עד לקבלת "עץ" בפעם הראשונה (מזכיר לנו משהו?).

בצורה כזו, מבנה הנתונים שלנו לא יהיה מאוזן לחלוטין, אבל "בערך" המרחק בין כל שני אלמנטים יהיה שניים, כפי שרצינו. צורה אפשרית למבנה הנתונים יכולה להיות:



### 3.1 ניתוח סיבוכיות

קיבלנו מבני נתונים ש"נראה" שעובד בסדר. בסעיף זה, נבצע ניתוח ונראה כמה יעיל מבני נתונים זה.

#### 3.1.1 ניתוח מקום

נחשב את תוחלת הזיכרון שמבני הנתונים צורך. נסמן את האיברים ב  $x_1, \dots, x_n$ . יהי  $X_i$  משתנה מקרי המציין בכמה רמות נמצא האיבר  $x_i$  (ולמעשה, כמה "מקום" תופס לנו  $x_i$  במבנה הנתונים). כפי שראינו בעבר ("מספר ההטלות עד שמקבלים עץ"), רואים כי:

$$E[X_i] = 2$$

נסמן ב  $SIZE$  משתנה מקרי המציין את גודל מבני הנתונים. ברור כי:  $SIZE = \sum_{i=1}^n X_i$ . נקבל:

$$E[SIZE] = \sum_{i=1}^n X_i = \sum_{i=1}^n 2 = 2n$$

כלומר, אנחנו מצפים לקבל מבנה נתונים עם גודל לינארי, בדיוק כמו ב-*ideal skip list*. ניתן לחשב את הנ"ל בצורה אחרת. נרצה לחשב מהו תוחלת מספר האיברים ברמה  $i$ . לשם כך, שוב נשתמש בלינאריות התוחלת. נסמן ב-  $X_j$  משתנה מקרי, המקבל 1 אם האיבר  $j$  עלה עד לרמה  $i$ , ו-0 אחרת. אז,  $\Pr[X_j = 1] = 2^{-i}$ . לכן, נקבל כי ברמה  $i$  ישנם בתוחלת:  $\sum_{j=1}^n X_j = n \cdot 2^{-i}$  איברים. או במילים אחרות:

- (תוחלת) מספר האיברים ברמה 0 (הרמה הראשונה) היא:  $n$ .
- תוחלת מספר האיברים ברמה 1 היא:  $n/2$ .
- תוחלת מספר האיברים ברמה 2 היא:  $n/4$ .
- תוחלת מספר האיברים ברמה 3 היא:  $n/8$ .
- ...

נסכום על הכל, ונקבל  $2n$ , בדיוק כמה שהיה לנו ב *ideal skip list*.

### 3.1.2 ניתוח זמן חיפוש

**חיפוש worst case.** במקרה הגרוע ביותר, אף איבר לא יעלה לרשימה השנייה, ופשוט נישאר עם רשימה מקושרת. לכן, זמן החיפוש במקרה הגרוע ביותר יהיה  $n$ , וזה יקרה בהסתברות  $2^{-n}$  (הסתברות ממש זניחה).

### ניתוח תוחלת מספר הרמות.

**למה 1.** בהסתברות גבוהה, רשימת דילוגים עם  $n$  אלמנטים מכילה  $O(\log n)$  רמות.

**הוכחה:** ההסתברות שאיבר מסויים  $x_i$  יעלה מעל  $c \cdot \log n$  רמות היא:  $2^{-c \log n} = 1/n^c$ . כעת, לפי חסם האיחוד:

$\Pr$  [skip list has more than  $c \cdot \log n$  levels]

$$= \Pr \left[ \bigvee_{i=1}^n x_i \text{ has promoted more than } c \cdot \log n \text{ times} \right] \leq \sum_{i=1}^n \Pr [x_i \text{ has promoted more than } c \cdot \log n \text{ times}]$$

$$\leq n \cdot \frac{1}{n^c} = \frac{1}{n^{c-1}}$$



אם כן, בהסתברות גבוהה, נקבל שבמבני הנתונים שלנו יש לכל היותר  $c \log n$  רמות.

### 3.1.3 תוחלת זמן חיפוש

**תוחלת זמן החיפוש.**<sup>1</sup> ניתוח זה מעט מסובך יותר. נתחיל את הניתוח במבט על מסלול החיפוש "לאחור". כלומר, מסתכלים על המסלול מהצומת התחתונה ביותר לראש הרשימה. בכל שלב, אנחנו עולים רמה אחת למעלה, או שפונים שמאלה. נזכור שכאשר הכנסנו את הנתונים לרשימה, על כל איבר שאנחנו מתבוננים בו במהלך המסלול, העלינו את האיבר הנתון רמה בהסתברות  $1/2$  (מה שאומר שבמקרה שלנו - יש לנו במסלול עליה למעלה), או שעצרנו את העליה שלו בהסתברות  $1/2$  (מה שאומר שנלך שמאלה במסלול). אנחנו מחשבים את תוחלת זמן החיפוש כאילו בנינו

אנחנו יודעים שבהסתברות גבוהה, אנחנו עולים למעלה מקסימום  $O(\log n)$  רמות. השאלה היא כמה פעמים אנחנו זזים שמאלה? (אולי הנתונים לא מסודרים יפה, ולכן אנחנו הולכים אולי אפילו  $O(n)$  פעמים שמאלה?) נסמן ב-  $C(k)$  את תוחלת העלות של מסלול שעולה  $k$  רמות. כעת, נתבונן על המסלול ההפוך. בכל שלב, יש שתי אפשרויות:

1. המסלול עולה מעלה. זה קורה בהסתברות  $1/2$ . במקרה זה, אנחנו משלמים 1 עבור הביקור בצומת עצמה, ובנוסף, אנחנו משלמים  $C(k-1)$ .

<sup>1</sup>דילגנו על חלק זה בכיתה.

2. המסלול הולך שמאלה. שוב, מקרה זה קורה בהסתברות  $1/2$ . במקרה זה, אנחנו משלמים 1 עבור הביקור בצומת עצמה, ובנוסף, אנחנו נשלם  $C(k)$  (נשארנו ברמה  $k$ ).

נקבל אם כן:

$$C(k) = \frac{1}{2} \cdot (C(k-1) + 1) + \frac{1}{2} \cdot (C(k) + 1)$$

$$C(k) = \frac{1}{2}C(k) + \frac{1}{2} \cdot C(k-1) + 1$$

$$C(k) = C(k-1) + 2$$

כאשר פותחים את הנוסחא, מקבלים:

$$C(k) = 2k$$

כלומר, קיבלנו כי תוחלת עלות החיפוש היא פי שניים ממספר הרמות. מכיוון שתוחלת מספר הרמות הוא  $O(\log n)$ , תוחלת עלות החיפוש היא בסה"כ  $O(\log n)$ .

### 3.1.4 תוחלת הכנסה

בהכנסת איבר, אנחנו קודם מוצאים את הקודם, מכניסים את האיבר, ומתחילים לקדם אותו רמות. תוחלת מספר הרמות שייקדם הוא  $2$  (למה?), תוחלת חיפוש הוא  $O(\log n)$ , ולכן נקבל כי בסה"כ תוחלת ההכנסה היא  $O(\log n)$ .

## References

- [1] Pugh, William. Skip lists: a probabilistic alternative to balanced trees. In *Communications of the ACM* 33 (6): 668—676.