

## תרגול 4

# מחסנית ורשימת דילוגים

תאריך עדכון אחרון: 9 באפריל 2010.

**שים 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 פעולות ניתן לבצע על מבנה הנתונים, וכיצד מנהלים אותו. אנו נפריד בין שני סוגים של פעולות על מבנה הנתונים - "שאלות" (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-דינמי, כלומר - מגדירים כיצד יבוצעו פעולות השינוי, לעומת זאת, ניתן להתייחס על מבנה נתונים סטטי, שבו לא מגדירים פעולות שינוי, אלא מניחים שברגע שמבנה הנתונים כבר בנוי וכל הפעולות שיתבצעו עליו יהיה מהסוג - "שאלות". 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 "שאלות".

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<sup>1</sup>. בד"כ בשלב זה נרצה להעריך את סדר גודל כל פעולה שהגדרנו, ונרצה לשפר את מבנה הנתונים כך שיתאים בצורה אופטימלית לפעולות שאותן אנחנו ממשיים. את שני השלבים האלה נעשה ברוב הקורס. לבסוף, בכדי להשתמש בפועל במבנה הנתונים, נצטרך לממש את מבנה הנתונים בשפת תוכנה מסוימת, לדוגמא  $C++$ .

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

## 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.

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 [1].

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:

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

ביטוי זה מקבל מינימום כאשר  $|L_2| = \sqrt[3]{|L_1|^2}$ ,  $|L_3| = \sqrt[3]{|L_1|}$  ואז נקבל:  $T(\text{search}_3) = 3\sqrt[3]{n}$ . 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?

הביטוי  $k \cdot \sqrt[k]{n}$  מקבל מינימום כאשר  $k = \log n$ . במקרה זה נקבל כי  $\log n \cdot \sqrt[\log n]{n} = T(\text{search}_k)$ , מכיוון שמתקיים  $\sqrt[\log n]{n} = 2$ , נקבל כי:  $T(\text{search}_{\log n}) = \log n \cdot 2$ . 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 סטטיים...

- **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 מסוימות נרצה להגביל את גודל המחסנית. במקרה שהמחסנית מלאה, ומישהו מבצע פעולת *push* - נקבל הודעת שגיאה שנקראת *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:  $6 + [(1 + 2) + (3 + (4 + 5))]$  - מקוננים כהלכה. לעומת זאת -  $1 + 2 + [(3 + 4) + 5]$  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 (+, -, ·, ...) 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-  $\text{top}(S)$  קודם ל- c:

(קודם מבחינת סדר פעולות)

2.2.2: הדפס  $\text{pop}(S)$ .

2.3:  $\text{push}(c, S)$ .

3: כל עוד S לא ריקה - הדפס  $\text{pop}(S)$

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 (-). המחסנית ריקה - מכניסים את . למחסנית.

• רואים אופרנד - 4. מדפיסים אותו. מקבלים:

3,4

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

3,4, .

• רואים 5, מדפיסים אותו. נקבל:

3,4, ., 5

• רואים (.), בודקים את ראש המחסנית, . קודם ל - , ולכן רק מכניסים את . למחסנית.

• רואים 6, מדפיסים אותו, ומרוקנים את המחסנית. נקבל:

3,4, ., 5,6, ., -

## ביבליוגרפיה

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