מבני נתונים - תרגול 4
מבני נתונים לינאריים

גלעד אשרוב

3 באפריל 2013

תקציר

בתרגול זה נלמד על מבני נתונים לינאריים. נתרגל מערך, מחסנית, תור ורשימה מקושרת.

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 - "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 למבנה 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" (לפחות על פי הגדרה, אלו פעולות שלא משנות את מבנה הנתונים).

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.

[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.

2 מערך לעומת רשימה מקושרת

מערך. מערך הוא אוסף של פריטים שניתן לגשת אליהם בצורה ישירה באמצעות אינדקס (כלומר, גישה לאיבר הממוקם באינדקס ה-i נעשית ב-$O(1)$). גישה ישירה לכל איבר נקראת "$random\ access$". זוהי ההגדרה האבסטרקטית של מבנה הנתונים.

המימוש בפועל נעשה בעזרת רצף של תאים בזיכרון. כלומר, מגדירים בלוק שלם בזיכרון (רציף) שגודלו כגודל מספר האיברים שאותם רוצים לשמור, ושומרים מצביע לתחילת המערך. מכיוון שהנתונים שמורים כבלוק בזיכרון, מקומו בזיכרון של האיבר במקום ה־i ניתן לחשב בעזרת i, המצביע לאיבר הראשון במערך וכמות הזיכרון הנצרכת לכל איבר ($sizeof$).

נעמוד על שתי תכונות עיקריות:

1. כאמור, גישה לאיבר ה-i במערך נעשית בזמן $O(1)$.
2. חוסר גמישות (1): קשה למחוק איברים במערך, תוך שמירת המקומות המנוצלים סמוכים זה לזה. ברגע שמוחקים איבר באינדקס i, נצטרך לבצע $shift$ ולהזיז את כל האיברים לאחר האינדקס i, מקום אחד אחורה. במקרה הגרוע, פעולה זו לוקחת $O(n)$, כאשר n הוא גודל המערך.
3. חוסר גמישות (2): קשה להוסיף איברים חדשים. אם כל המערך מנוצל, אי אפשר פשוט להוסיף איברים חדשים, שכן כל המערך הוקצה כבלוק שלם בזיכרון, והמקומות בזכרון המופיעים לאחר המערך כבר הוקצו למטרה אחרת. לפיכך, יש צורך להקצות מערך חדש, ולהעתיק את כל הנתונים אליו. עלות זו עלולה להגיע ל-$O(n)$.[2]

עלויות:

- **חיפוש איבר.** במערך ממויין - חיפוש איבר יעלה $O(\log n)$ בעזרת חיפוש בינארי. במערך לא ממויין ־ אין ברירה אלא לסרוק את כל המערך, ועלות חיפוש היא $O(n)$.
- **הוספת איבר.** במערך ממויין, הכנסת איבר תוך כדי שמירה שיישאר ממויין תעלה $O(n)$. במערך לא ממויין, אם נתון מצביע למקום אותו רוצים להכניס את האיבר - העלות היא $O(1)$.
- **הסרת איבר.** במערך ממויין $O(n)$. במערך לא ממויין (בהינתן האיבר) $O(1)$.

רשימה מקושרת. רשימה מקושרת הינו אוסף של איברים המפוזרים בזיכרון המחשב, כאשר בכל איבר מאוחסן המידע אותו רוצים לאחסן, ובנוסף - מצביע לאיבר הבא ברשימה.

כלומר, כל צומת מורכב משני אלמנטים: הנתון עצמו, ומצביע לאיבר הבא ברשימה. נשמור מצביע מיוחד לראש הרשימה. בנוסף, האיבר האחרון ברשימה יצביע ל: `NULL`.

בצורה זו, נקבל מעט יותר גמישות מאשר במערך: בהינתן מצביע לאיבר אותו רוצים ניתן להכניס ולהוציא ב$O(1)$. לעומת זאת, גישה לאיבר ה-i עולה כעת $O(i)$ - שכן יש לעבור את כל המסלול מתחילת הרשימה ועד אליו.

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:

[2] בהמשך הקורס נראה מבנה נתונים הנקרא "מערך דינאמי", שבו ניתן לקבל את אותן התכונות כמו במערך (כלומר, $random\ access$), ובנוסף, ננהל את ההכנסות בצורה חכמה כך שהעלות של הכנסה תעלה $O(1)$ לשיעורין (כלומר, לפחות בשלב זה של הקורס, "בממוצע").

- 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 $stack\ overflow$. לעיתים ניתן לקבל גם הודעת $stack\ underflow$, הקורת במקרה שמבקשים להוציא מהמחסנית איבר כאשר המחסנית ריקה. בכדי להימנע מכך - לפני כל פעולת pop נבצע את פעולת $isEmpty$, ונבצע את פעולת ה-pop אך ורק אם $isEmpty$ החזיר `false`.

מימוש. נממש חמסנית בעזרת מערך או רשימה. ניתן לממש כך שעלות על אחת מהפעולות תהיה $O(1)$ (איך?).

3.1 דוגמא לשימוש במחסנית - המרת ביטוי מייצוג $infix$ לייצוג $postfix$

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

$$4+3$$

כלומר, האופרטור $(+,-,*,/)$ מופיע בין שני האופרנדים $(3,4)$. דוגמא נוספת היא למשל:

$$\text{infix} : (4+6)/5 \qquad\qquad \text{postfix} : 4\ 6\ +\ 5\ /$$

דוגמא נוספת:

$$\text{infix} : 4+6/5 \qquad\qquad \text{postfix} : 4\ 6\ 5\ /\ +$$

דוגמא נוספת:

$$\text{infix} : 4+6/5+2/3 \qquad\qquad \text{postfix} : 4\ 6\ 5\ /\ +\ 2\ 3\ /\ +$$

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)
      2.2.1.1: הדפס pop(S).
    2.2.2: 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. מכיון 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:

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

4 תרגילים

תרגיל 1. סדרה מעורבבת של 10 פעולות $push$ ו-10 פעולות pop בוצעה, כך שפעולות ה-$push$ הכניסו למחסנית את הספרות מ-0 ל-9 לפי סדר. מה מבין הסדרות הבאות אינה אפשרית? (הסדרות משמאל מייצגות את הערכים שהוצאו בפעולות pop לפי סדר הוצאתם)

א. $4,3,2,1,0,9,8,7,6,5$.

ב. $4,6,8,7,5,3,2,9,1,0$.

ג. $2,5,6,7,4,8,9,3,1,0$.

ד. $4,3,2,1,0,5,6,7,8,9$.

פתרון: נסמן ב-$*$ פעולות pop. נקבל:

1. $0,1,2,3,4,*,*,*,*,*,5,6,7,8,9,*,*,*,*,*$.

2. $0,1,2,3,4,*,5,6,*,7,8,*,*,*,*,*,9,*,*,*$.

3. $0,1,2,*,3,4,5,*,6,*,7,*,*,8,*,9,*,*,*,*$.

4. $0,1,2,3,4,*,*,*,*,*,5,*,6,*,7,*,8,*,9,*$.

תרגיל 2. נתונה רשימה מקושרת חד כיוונית. הציעו דרך לטייל (קדימה ואחורה) על הרשימה החד כיוונית (כלומר, ישנן שתי פעולות - "טיול קדימה" ובצורה כזו מדפיסים את כל האיברים ברשימה לפי סדר, מהאיבר הראשון לאיבר האחרון), ו"טיול אחורה" שבו מתחילים מהאיבר האחרון ומדפיסים את הרשימה בסדר הפוך).
הפתרון אמור להיות ללא שימוש ברשימה דו־כיוונית.

פתרון: בכל איבר, נשמור ככתובת "האיבר הבא" את תוצאת הXOR של כתובת האיבר הקודם וכתובת האיבר העוקב (כלומר, שומרים $PREV \oplus NEXT$).
כעת, בטיול "קדימה", יש לנו את כתובת האיבר הקודם - $PREV$. לכן, כאשר נחשב:
$PREV \oplus (PREV \oplus NEXT) = NEXT$, נקבל למעשה את הכתובת של האיבר הבא.
כאשר נבצע "טיול לאחור", יש לנו את כתובת האיבר הבא - $NEXT$, וכאשר נחשב
$NEXT \oplus (PREV \oplus NEXT) = PREV$, נקבל את כתובת האיבר הקודם.

תרגיל 3. ממשו תור באמצעות שתי מחסניות.

פתרון: נחזיק שתי מחסניות. נקרא להן inbox ו-outbox.
כאשר נדחוף לתור - נדחוף תמיד ל-inbox.
כאשר נשלוף מהתור, נבצע את הפעולה הבאה:

```
אם outbox ריקה:
    כל עוד inbox לא ריקה
        שלוף איבר ממחסנית inbox והשם במשתנה x
        דחוף את x ל outbox
שלוף מהמחסנית outbox והחזר את התוצאה.
```

תרגיל 4. ממשו מבנה נתונים השומר מטריצה של ביטים, ותומך בפעולות הבאות:

1. `init`: יצירת מופע ריק של מבנה הנתונים (עלות נדרשת: $O(n^2)$).
2. `Flip(i,j)`: הפיכת הערך בתא - (i,j). (עלות נדרשת - $O(1)$).
3. `HasAllOnesRow` - מחזיר האם יש שורה שכולה אחדות.
4. `HasAllZerosRow` - מחזיר האם יש שורה שכולה אפסים.

פתרון: מבנה נתונים שלנו יכיל את השדות הבאות:

1. מערך דו מימידי.
2. מערך בגודל n של מספרים. המקום ה-i במערך יציין את מספר האחדות בשורה i. נקרא למערך `numberOfOnesInRow`.
3. משתנה המציין את מספר השורות שכולן אחדים. נקרא למשתנה זה `numberOfAllOneRows`.
4. משתנה המציין את מספר השורות שכולן אפסים. נקרא למשתנה זה `numberOfAllZeroRows`.

בהתחלה, נאתחל את המערך הדו־מימדי לכולו אפסים, נאתחל את כל התאים במערך numberOfOnesInRow לאפסים. את המשתנה numberOfAllOneRows נאתחל ל-0, ואת המשתנה numberOfAllZeroRows נאתחל ל-n (מספר השורות).

כעת, כאשר נקרא ל- Flip(i, j), ניגש למקום ה־(i, j) במערך הדו מימידי ונחליף את תוכנו (נבצע $\oplus 1$). בנוסף נבצע: אם עברנו מ-0 ל-1, ייתכן שקיבלנו עכשיו שורת אחדות. ניגש ל[i]numberOfOnesInRow ונגדיל אותו ב־1. אם הערך הנ"ל גדל מ-$n-1$ ל- n, קיבלנו שורת אחדות ולכן נקדם את ערך המשתנה numberOfAllOneRows באחד. אם numberOfOnesInRow[i] גדל מ-0 ל-1, הייתה לנו שורת אפסים שביטלנו אותה. לכן, נקטין את numberOfAllZeroRows באחד.

בצורה דומה, נעשה עבור מעבר מ-1 ל-0.

כאשר נישאל HasAllOnesRow, נחזיר פשוט האם numberOfAllOneRows גדול מ-0.

כאשר נישאל HasAllZerosRow, נחזיר פשוט האם numberOfAllZerosRows גדול מ-0.

תרגיל 5. 1. רשימה מעגלית היא רשימה מקושרת שבה האיבר האחרון מצביע לאיבר הראשון. כלומר:

$$x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightsquigarrow x_1 \rightarrow x_2 \ldots$$

הציגו אלגוריתם המקבל כקלט רשימה, ומחזיר האם היא מעגלית או רשימה רגילה (רשימה שבה האיבר האחרון מצביע ל - $NULL$). הציגו פסאדו קוד. האלגוריתם צריך לעבוד בזמן $O(n)$ ולהשתמש בסיבוכיות זיכרון של $O(1)$.

2. רשימה חלקית מעגלית היא רשימה שבה האיבר האחרון מצביע לאיבר כלשהו ברשימה. כלומר, הרשימה נראית בצורה הבאה:

$$x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n \rightsquigarrow x_i \rightarrow x_{i+1} \ldots$$

כאשר $1 \leq i \leq n$ (ולכן, רשימה מעגלית שהוגדרה בסעיף (1) היא מקרה פרטי של רשימה חלקית מעגלית).

הציגו אלגוריתם המקבל כקלט רשימה, ומחזיר האם היא חלקית מעגלית או רשימה רגילה. הציגו פסאדו־קוד. האלגוריתם צריך לעבוד בזמן $O(n)$ וסיבוכיות זכרון של $O(1)$. יש **להוכיח** שזמן הריצה של האלגוריתם הוא $O(n)$.

פתרון: הסעיף הראשון קל - נחזיק פוינטר לאיבר הראשון ברשימה, ופוינטר נוסף שירוץ על הרשימה. בכל שלב נבדוק אם הם נפגשים. אם הרשימה מעגלית ־ הם ייפגשו. במקרה זה האלגוריתם יעצור ויחזיר "הרשימה מעגלית". אם היא אינה מעגלית ־ הפוינטר השני יגיע לאיבר האחרון ברשימה ־ שמצביע ל־NULL. האלגוריתם יעצור ויחזיר "רשימה רגילה".

הסעיף השני ־ לא ניתן לעמוד עם הפוינטר הראשון במקום. גם הוא צריך לזוז. נעיר שקל מאוד למצוא אלגוריתם שעובד בזמן $O(n)$ ובסיבוכיות מקום $O(n)$, (כיצד?). כמו־כן, ברור כי החסם התחתון לאלגוריתם מבחינת זמן ריצה הינו $O(n)$ (האלגוריתם חייב לרוץ על כל האיברים). אנו נראה אלגוריתם שמשתמש ב ־ $O(1)$ מקום, ולכן ־ הוא האופטימלי ביותר לבעיה.

נתחיל את הפתרון ברעיון אסטרקטי. נניח שצב וארנב מתחרים בתחרות ריצה. אם המסלול מעגלי, באחד מהסיבובים ־ שניהם ייפגשו שוב, בנקודה כלשהי על פני המסלול. לעומת זאת, אם המסלול אינו מעגלי, הארנב יגיע קודם לקו הסיום, ולעולם שניהם לא יעמדו באותו המקום.

מימוש הרעיון אצלינו ייעשה באופן הבא. נחזיק שני מצביעים, שזזים על הרשימה "בקצבים שונים": הראשון זז איבר אחרי איבר ("הצב"), בעוד השני זז בקפיצות של שני איברים ("הארנב"). הטענה היא שאם הרשימה היא חלקית מעגלית ־ שני המצביעים ייפגשו. לעומת־זאת, אם הרשימה אינה חלקית מעגלית (ולכן ־ היא סופית, בפרט, האיבר האחרון מצביע ל־NULL), המצביעים לא ייפגשו, ולמעשה אחד מהם יגיע קודם לאיבר האחרון, ויידע שהגיע לסוף הרשימה (יידע להגיד "הרשימה אינה חלקית מעגלית").

בצורה מפורשת, האלגוריתם יעבוד בצורה הבאה:

- **קלט:** רשימה מקושרת.

- **פלט:** האם הרשימה חלקית מעגלית.
- **האלגוריתם:**

שמור מצביע p1 לראש הרשימה. שמור מצביע נוסף p2 לראש הרשימה.
קדם את המצביע p1 באיבר אחד. קדם את המצביע p2 בשני איברים.
כל עוד אחד מהמצביעים לא הגיע ל – NULL, בצע:
    אם המצביעים מצביעים לאותו איבר (המצביעים שווים ממש):
        החזר 1 ("הרשימה חלקית מעגלית").
    קדם את המצביע p1 באיבר אחד.
    קדם את המצביע p2 בשני איברים.
החזר 0 ("הרשימה אינה חלקית מעגלית").