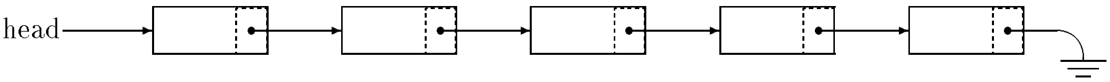
תרגיל כיתה III
רשימות מקושרות

1 רשימה מקושרת חד-כיוונית

1.1 הגדרה

<u>רשימה מקושרת</u> – סדרת רשומות, כאשר בכל אחת יש שדה שמצביע לרשומה הבאה.



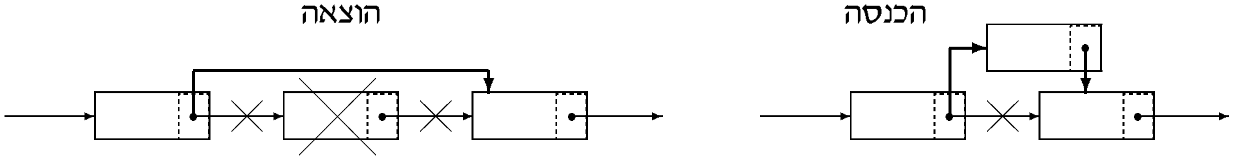
הגדרה ב-C:

```
struct element {
  int info;
     ⋮
  struct element *next;
}
```

1.2 כמה כללי תכונות

1. ברשימה מקושרת קיימת אפשרות להכניס ולהוציא רשומות ב-$O(1)$ זמן בהינתן מצביע. אך יש להיזהר: ברשימה מקושרת חד-כיוונית לא ניתן לגשת מרשומה כלשהי לזו הקודמת לה! לכן כדי לבצע את פעולות ההכנסה/הוצאה בזמן קבוע הפונקציה המתאימה חייבת לקבל כפרמטר מצביע לרשומה <u>שלפני</u> מקום ההכנסה/הוצאה. אז הפעולות מתבצעות כמתואר בתרשים (החיצים העבים מציינים את הקשרים החדשים שנוצרים עקב ביצוע הפעולה):

הכנסה הוצאה



2. נהוג להגדיר <u>כותרת</u> (header) לרשימה מקושרת.
כותרת זוהי רשומה ראשונה ברשימה, אשר לא מהווה איבר הרשימה. תפקידה להצביע בשדה ה-next שלה על האיבר הראשון של הרשימה. שדות האינפורמציה של הכותרת אינם רלוונטיים, וניתן לשמור בהם מידע כללי לגבי הרשימה. בעזרת הכותרת נוכל להמנע מהתייחסות מיוחדת למקרים בהם פעולה כלשהי מופעלת על האיבר הראשון.

1.3 סיבוכיות המבנה

1. כפי שאמרנו, סיבוכיות הזמן של הפעולות insert/delete בהינתן מצביע למקום המתאים הינה $O(1)$.

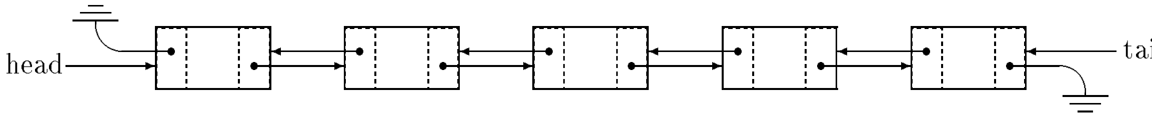
2. סיבוכיות הזמן של חיפוש ברשימה היא $O(n)$ (כאשר n הוא מספר האיברים במבנה).

3. סיבוכיות המקום גם היא $O(n)$.

2 ווריאציות של רשימות מקושרות

2.1 רשימה מקושרת דו-כיוונית

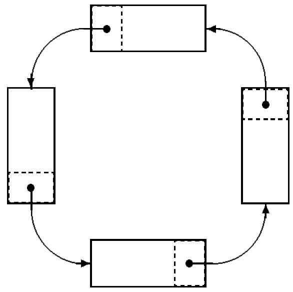
רשימה מקושרת דו-כיוונית מוגדרת בדומה לחד-כיוונית, פרט לכך שכל רשומה בה מכילה שני שדות המצביעים: קדימה (next) ואחורה (prev).



משתמשים ברשימה כזו במקרים, בהם יש צורך לגשת מרשומה כלשהי במבנה גם קדימה וגם אחורה.

2.2 רשימה מעגלית

רשימה מעגלית מוגדרת כרשימה חד-כיוונית, בה האיבר האחרון מצביע על הראשון.



השימוש ברשימה מעגלית נעשה כאשר מבצעים סיורים ציקליים במבנה. גם ברשימה רגילה ניתן לסייר ציקלית, אך זה כרוך בטיפול מיוחד עבור מקרה של השגת קצה הרשימה.

3 תרגילים

תרגיל 1

הצע מבנה נתונים המאפשר את ביצוע הפעולות הבאות על קבוצת מספרים שלמים S. כל המספרים בקבוצה הם מהתחום $[0, n-1]$, כאשר n ידוע מראש. מתחילים ממבנה ריק. אלו הפעולות הנדרשות:

1. בדיקה האם $k \in S$.
2. הוספת איבר לקבוצה S.
3. הוצאת המספר k מהקבוצה S.
4. בדיקה האם S ריקה.
5. בדיקה האם S מלאה (מכילה את כל האיברים מהתחום).
6. מציאת האיבר הוותיק ביותר ב-S.

הסיבוכיות הנדרשת לכל הפעולות הנ"ל היא $O(1)$. סיבוכיות האיתחול היא $O(n)$. אין להשתמש בה-קצאת זכרון דינמית.

פתרון: נשתמש במערך בגודל n, בו בתא ה-i-י נסמן האם $i \in S$. בנוסף נשמור רשימה מקושרת דו-כוונית של המספרים השייכים ל-S. סדר האיברים ברשימה יקבע עפ"י זמן הצטרפותם ל-S, כך האיבר בראש הרשימה הינו הותיק ביותר. מכיוון שאין להשתמש בהקצאת זכרון דינמית, במקום מצביעים נשתמש בשדות במערך שיכילו אינדקסים של תאים אחרים במערך.

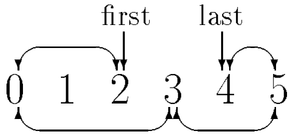
תאור מבנה נתינים :

- מערך $N[n]$ שכל תא בו מכיל את השדות הבאים:
  - $flag = 1 - flag$ אם"ם $i \in S$ (מאותחל ל-0).
  - $prev$ – "מצביע" לאיבר הקודם ברשימה (מאותחל ל--1).
  - $next$ – "מצביע" לאיבר הבא ברשימה (מאותחל ל--1).
- המשתנים הבאים:
  - $counter$ – מונה למספר האיברים ב-S (מאותחל ל-0).
  - $first$ – "מצביע" לאיבר הראשון ברשימה (מאותחל ל--1).
  - $next$ – "מצביע" לאיבר האחרון ברשימה (מאותחל ל--1).

דוגמא :

	0	1	2	3	4	5
flag	1	0	1	1	1	1
prev	2	-1	-1	0	5	3
next	3	-1	0	5	-1	4

counter = 5
first = 2
last = 4



מימוש הפעולות :

1. **בדיקה האם** $k \in S$ — **בדיקת** $N[k].flag$.
2. **הוספת** k **ל-**S.

```
if (N[k].flag == 1) return;      // אם האיבר כבר נמצא בקבוצה, סיים
N[k].flag = 1;                   // סמן שייכות לקבוצה
counter++;                       // עדכן את המונה
N[k].prev = last;                // הכנסה לשרימה מקושרת
N[k].next = -1;
if (counter == 1) first = k;
 else N[last].next = k;
last = k;
```

3. **הוצאת** k **מ-**S.

```
if (N[k].flag == 0) return;              // אם האיבר אינו בקבוצה, סיים
N[k].flag = 0;                           // סמן אי שייכות לקבוצה
counter--;                               // עדכן את המונה
if (first == k) first = N[k].next;       // הוצאה משרימה מקושרת
 else N[N[k].prev].next = N[k].next;
if (last == k) last = N[k].prev;
 else N[N[k].next].prev = N[k].prev;
```

III.3

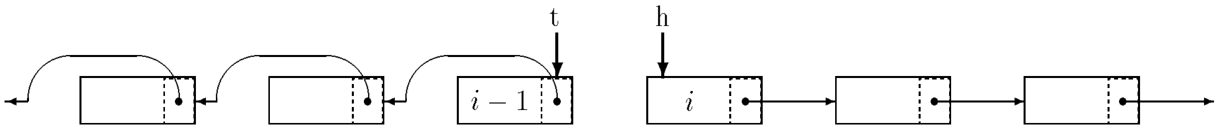
4. בדיקה האם S ריקה — בדיקה האם counter שווה ל-0.
5. בדיקה האם S מלאה — בדיקה האם counter שווה ל-n.
6. האיבר הותיק ביותר ב-S הוא first.

תרגיל 2

נתונה רשימה מקושרת חד-כוונית. יש לממש את הפעולה הבאה: לגשת עם מצביע לרשומה ה-j-ית של הרשימה. הסיבוכיות הנדרשת היא $\mathrm{O}(|i-j|)$ כאשר i הוא האינדקס של הרשומה ממנה מגיעים. מותר להשתמש ב-$\mathrm{O}(1)$ זכרון נוסף.

ניתן להניח: האיבר ממנו מתחילים לראשונה הוא האיבר הראשון ברשימה (כלומר, בהתחלה $i=0$).

פתרון: כאשר $i<j$ אין כל בעיה. פשוט נתקדם ברשימה $|i-j|$ צעדים. הבעיה תיווצר כאשר $i>j$, כי אז אי אפשר לעבור על המצביעים מ-i ל-j, אלא צריך לקפוץ לראש הרשימה ולעבור עד למקום ה-j-י. במקרה הזה הסיבוכיות תהיה $\mathrm{O}(j)$, ולא מה שהתבקשנו. גם להפוך את הרשימה לדו-כיוונית (כדי לאפשר התקדמות חופשית בשני הכיוונים) איננו יכולים, כי זה ידרוש $\mathrm{O}(n)$ מקום נוסף.
הפתרון שנציע ישנה את מבנה הרשימה בצורה הבאה: כאשר אנחנו נמצאים באיבר ה-i-י, נשמור שני מצביעים "חיצוניים" לאיבר ה-i-י ולאיבר ה-$(i-1)$. שדות המצביעים של כל הרשומות הנמצאות ברשימה אחרי האיבר ה-i-י יצביעו לאיבר הבא כרגיל, ואלו הרשומות שהן <u>לפני</u> האיבר ה-i-י ברשימה יצביעו על האיבר <u>הקודם</u> (כלומר, יהיה לנו חור בין האיבר בו נמצאים וזה שלפניו):



כדי לשמור על המצב הזה בכל רגע ורגע, נעביר בכל צעד את המצביעים על קצות החור, כך שהחור תמיד יהיה לפני האיבר הנוכחי. לשם כך נשתמש במשתנה עזר x.

```
x = t;                      // צעד אחד קדימה:
t = h;
h = h -> next;
t -> next = x;

x = h;                      // צעד אחד אחורה:
h = t;
t = t -> next;
h -> next = x;

h = head_of_list;           // איתחול:
t = NULL;
```

$head\_of\_list$ הינו מצביע לראש הרשימה.