מטלת מנחה (ממ"ן) 15

הקורס: 20433 - מבני נתונים

שאלה 1 (20 נקודות: 10 נקי לכל סעיף)

א. כתוב אלגוריתם המקבל מצביע לרשימה מקושרת חד-כיוונית והופך את כיוון המצביעים ברשימה, כך שהצומת האחרון ברשימה יהפוך להיות הראשון, הצומת הלפני אחרון יהפוך להיות השני וכו'.

Reverse (List)

Prev ← Nil — אין קודם לראשון

While List <> Nil do — כל עוד יש איברים ברשימה

Next ← next(List) — שמירת האיבר הבא

List(next) ← Prev — עידכון הצבעת האיבר הנוכחי להצביע על הקודם לו

Prev ← List — קידום המצביעים לאיבר הבא. הנוכחי הופך קודם.

List ← Next — הבא, הופך לנוכחי.

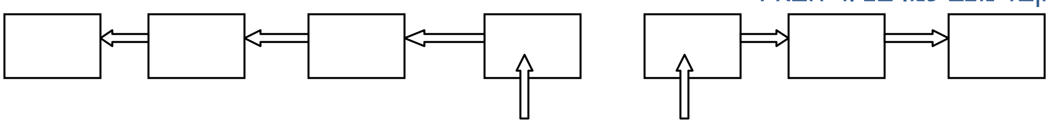
Return Prev — המצביע על הנוכחי מגיע עד nil

לכן ההצבעה על הקודם מצביע לאיבר שהפך לראשון

ב. נתונה רשימה מקושרת חד-כיוונית.
יש לגשת לאיברים ברשימה עפ"י סדרת בקשות המגיעה ב"זמן אמיתי". (כלומר, סדרת הבקשות איננה ידועה מראש!) למשל, אם סדרת הבקשות היא ...,72, 83, 17, אז צריך קודם כל לגשת לאיבר הנמצא במקום ה 72 ברשימה, אח"כ יש לגשת לאיבר הנמצא במקום ה 83, אח"כ לאיבר הנמצא במקום ה17- וכך הלאה.
נסמן את סדרת הבקשות ב- $L_1, L_2, \ldots, L_n$.
תאר אלגוריתם, שיבצע את סדרת הבקשות בזמן $O\left(L_1 + \sum_{i=2}^{n} \left|L_i - L_{i-1}\right|\right)$

הערה: מותר לשנות את הרשימה המקורית.

הרעיון הוא שכל התקדמות – לפנים או לאחור, תעשה ע"י הפיכת ההצבעות של הרשימה עליה עוברים כדי להגיע לאיבר מסויים. כך, כאשר נמצאים "על" איבר מסויים, אפשר להמשיך ממנו "קדימה" (לכיוון איברים בעלי אינדקס גדול יותר) וגם לאחור (משם הגענו אולי), ולכן הפכנו את ההצבעות כך שניתן להתקדם גם אל איברים בעלי אינדקס נמוך יותר.
לצורך כך נשתמש באלגוריתם דומה מאוד לאלגוריתם מסעיף א, בשינוי קל, כך שההתקדמות אינה עד סוף הרשימה אלא מספר מסויים של איברים, וכן, נחזיר שני מצביעים – מצביע לאיבר שהפך ראשון ברשימה ההפוכה, ומצביע לאיבר הבא – שהפך ראשון ברשימה שלא הפכנו. כך נקבל מצב כמו בציור הבא:



כעת, הביצוע של סידרת הבקשות יתבצע בזמן:

הבקשה הראשונה: $O(L_1)$, הבקשה הבאה: $O(|L_2 - L_1|)$ כיוון שנצטרך לעבור רק על האיברים שבין L_2 לבין המקום בו היינו כשהתחלנו לטפל בבקשה: L_1.
באותו אופן טיפול בבקשה השלישית לוקח: $O(|L_3 - L_2|)$.

ובאופן כללי: טיפול בכל הבקשות עד הבקשה ה- n ית לוקח זמן:
$O(L_1 + |L_2 - L_1| + |L_3 - L_2| + \cdots + |L_n - L_{n-1}|)$ ובסימון של $\sum$: $O\left(L_1 + \sum_{i=2}^{n} |L_i - L_{i-1}|\right)$

וזה סדר הגודל של האלגוריתם שהתבקשנו להציע.

הערה: בשאלה נכתב שיש <u>להציע</u> אלגוריתם, ולא לכתוב אלגוריתם, ולכן מספיק התיאור הנ"ל של אלגוריתם ואין צורך בפירוט רב מזה.

שאלה 2 (20 נקודות)

נתונים שני מספרים עשרוניים גדולים מאוד *x, y*.
כל אחד מהמספרים נמצא במחסנית, כך שהספרה המשמעותית ביותר נמצאת בתחתית המחסנית.
כתוב אלגוריתם המקבל שתי מחסניות S_1, S_2 שבהן מאוחסנים המספרים *x* ו- *y*, ומחזיר מחסנית S_3 שבה נמצא הסכום $x+y$. (גם ב- S_3, הספרה המשמעותית ביותר תהיה בתחתית המחסנית).
פרט ל- S_3, מותר לאלגוריתם להשתמש במחסנית עזר אחת נוספת.
מותר להשתמש אך ורק בפעולות הבסיסיות המוגדרות על מחסנית ובמידת הצורך גם בפעולה head(S) המחזירה את ערך האיבר הנמצא בראש המחסנית.

```
stack-init(s3)
stack-init(s)
carry ←0
while (not (stack-empty(s1))) and (not (stack-empty(s2) ))    //חיבור 2 ספרות
        x ←pop(s1)
        y ←pop(s2)
        push (s, x+y+carry mod 10)
        carry ←x+y+carry div 10
while (not (stack-empty(s1))                    // S1 נותרו ספרות רק במחסנית
        x ←pop(s1)
        push (s, x+carry mod 10)
        carry ← x+carry div 10
while  (not (stack-empty(s2) )                  // S2 נותרו ספרות רק במחסנית
        y ←pop(s2)
        push (s, y+carry mod 10)
        carry ← y+carry div 10
while (not (stack-empty(s))                     // העברת המספר למחסנית המטרה
        push (s3, pop(s) )
```

שאלה 3 (20 נקודות: 10 נק' לכל סעיף)
הכלה בין עצים בינריים מוגדרת באופן הבא:
עץ *t2* מוכל בעץ *t1* אם *t2* זהה ל- *t1*, או ש-*t2* מוכל בתת-עץ השמאלי או בתת עץ הימני של השורש של *t1*. (שני עצים בינריים הם זהים אם שניהם ריקים, או אם שניהם אינם ריקים ומתקיימים שני התנאים הבאים: 1. הערכים שבשני השורשים הם שווים. 2. תתי-העצים השמאליים והימניים של השורשים הם זהים בהתאמה).

א. תאר אלגוריתם הבודק האם עץ *t2* מוכל בעץ *t1*.
ב. יהי *t1* עץ בינרי בעל *n* צמתים, כך שבכל צומת יש ערך שונה. כמה עצים *t2* קיימים, כך ש-*t2* מוכל ב-*t1*?

<u>תשובה</u>
האלגוריתם משתמש באלגוריתם לבדיקת שוויון בין שני עצים בינריים. האלגוריתם לבדיקת שוויון מופיע מייד לאחר האלגוריתם לבדיקת הכלה.

```
Contain (t1, t2)
if t1=nil  and  t2 ≠ nil
   return false
else if equal (t1, t2)
        return true
     else if contain (t1(left), t2) or contain (t1(right), t2)
             return true
          else
               return false
```

האלגורית לבדיקת שוויון בין שני עצי, בודק אם שני העצים ריקים – אם שניהם ריקים – הרי שהם שווים!

אם שניהם אינם ריקים – אזי נבדקים השורשים, האם לשניהם בנים שמאליים, או לחלופין – האם לשניהם אין בן שמאלי, והאם לשניהם בן ימני, או לחלופין – האם לשניהם אין בן ימני.
וזאת, מפני שאם לאחד השורשים יש בן ימני (למשל), ולחברו אין בן ימני – הרי ששני העצים – בהכרח אינם שווים.
אם השורשים הם באותה "תצורה", הרי שהשוויון בין העצים יקבע, ע"י שוויון בין תת העץ השמאלי של האחד לתת העץ השמאלי של חברו וכן שוויון בין תת העץ הימני של האחד, לתת העץ הימני של חברו.

```
equal (t1, t2)
if t1 = t2 = nil
   return true
else if t1 ≠ nil  and t2 ≠ nil
        if ((t1(left) = t2(left) = nil) or
           (t1(left) ≠ nil  and  t2(left) ≠ nil) )  and
           (t1(right) = t2(right)  = nil) or
           (t1(right) ≠ nil  and t2(right) ≠ nil)
           return equal(t1(left), t2(left))  and  equal(t1(right), t2(right)
         else   return false
     else  return false
```

שאלה 4 (14 נקודות : 7 נק' לכל סעיף)

H היא טבלת גיבוב בגודל 11.

פונקציית הגיבוב היא: $h(k, i) = (h_1(k) + i\, h_2(k)) \mod 11$

כאשר: $h_1(k) = k \mod 11$

$h_2(k) = (k \mod 9) + 1$

לכל אחת משתי הטבלאות שלהלן, קבע האם קיימת סדרה חוקית של פעולות הכנסה שתוצאתה היא הטבלה הנתונה:

א.

	0	1	2	3	4	5	6	7	8	9	10
k	36	26	11	12	46	17	47	16	7	8	9
k mod 11	*3*	*4*	*0*	*1*	*2*	*6*	*3*	*5*	*7*	*8*	*9*

אפשר לראות כי אף מפתח אינו במקומו הראשוני, לכן לא קיימת סדרה חוקית של פעולות הכנסה.

ב.

0	1	2	3	4	5	6	7	8	9	10
13	50	16	45	22	30	8	7	19	47	52

7 → 7
19 → 8
8 →8 → 8+8+1=6
50 → 6 →6+5+1=1
וכך הלאה עד לפתרון הקיים

שאלה 5 (26 נקודות: 13 נק' לכל סעיף)

נתונה קבוצת איברים S. על הקבוצה S מוגדרות הפעולות הבאות:

$\text{INSERT}(S, x)$ - הכנסת האיבר x ל-S.

$\text{SEARCH}(S, k)$ - החזרת מצביע לאיבר ב-S בעל מפתח k;
אם אין ב-S איבר כזה - החזרת NIL.

הקבוצה S מוגבלת בגודלה, ולכן לפני כל ביצוע פעולה של הכנסת איבר ל-S צריך למחוק איבר מ-S כדי לפנות מקום לאיבר חדש.
מחיקת האיבר יכולה להתבצע באחת משתי שיטות:

1. FIFO (First In First Out) - מחיקת האיבר שהוכנס ל-S לפני הכי הרבה זמן.
2. LRU (Least Recently Used) - מחיקת האיבר ב-S שניגשנו אליו לפני הכי הרבה זמן (כל פעולה של החזרת מצביע לאיבר x, או הפעולה של הכנסת x ל-S הן פעולות שבהן התבצעה גישה לאיבר x).

מכיוון שיש חשיבות רבה למהירות הביצוע של הפעולות השונות, כל פעולה צריכה להתבצע בזמן $O(1)$ בממוצע.
הצע מבנה נתונים עבור הקבוצה S כאשר:

א. מחיקת איבר מתבצעת בשיטת FIFO.
ב. מחיקת איבר מתבצעת בשיטת LRU.

הערה: מבנה הנתונים יכול להיות מורכב מכמה מבני נתונים בסיסיים.

א. טבלת גיבוב + תור — כל איבר יוכנס גם לטבלת הגיבוב וגם לתור.
האיבר שיימחק יהיה האיבר שבראש התור.

ב. טבלת גיבוב + רשימה דו-מקושרת, שמנוהלת בשיטת ה- move-to-front
(כלומר, כל איבר שניגשים אליו מועבר לתחילת הרשימה).
לצורך עדכון הרשימה לאחר ביצוע הפעולה SEARCH, צריך לשמור מצביעים מכל איבר בטבלת הגיבוב לאיבר המתאים ברשימה.
כל איבר יוכנס גם לטבלת הגיבוב וגם לראש הרשימה.
האיבר שיימחק יהיה האיבר שבסוף הרשימה.

בשני המקרים, נשמור את **הנתונים בטבלת גיבוב** (התבקשנו שכל פעולה תתבצע בזמן $O(1)$ בממוצע.) וכן נשמור את **המפתח במבנה נתונים נוסף**.

א. מבנה הנתונים הנוסף, יהיה **תור** (מבנה הנתונים שתומך ב FIFO)
כל נתון שיוכנס לטבלת הגיבוב, המפתח שלו יוכנס גם לתור.
בכל פעם שנאלץ למחוק נתון מתוך הקבוצה S, ניגש לראש התור – נבצע dequeue לאיבר שבראש התור. האיבר יכיל את המפתח של הנתונים. נחפש את המפתח בטבלת הגיבוב-וכך נוכל לשלוף את האיבר המתאים גם מתוך טבלת הגיבוב.

Insert (S,x)	Search (S,K)
Key ←Dequeue(Q) HashTable(remove, Key) HashTable(insert, x, key(x)) Enqueue(Q, key(x))	Return HashTable(search, K)

ב. כאשר מחיקת איבר מתבצעת בשיטת LRU : נשמור בנוסף לטבלת הגיבוב, גם רשימה דו מקושרת של מפתחות של האיברים שבקבוצה.
הרשימה תנוהל כמו תור (הוספה לזנב, והוצאה מהראש) אבל נצטרך דרך לגשת גם לאיברים שבתוך הרשימה, ולכן לא ניתן להשתמש בתור.
לכל נתון, נשמור בתוך טבלת הגיבוב, בנוסף לנתונים - מצביע אל מיקומו של המפתח ברשימה הדו מקושרת, זאת כדי שבכל פעם שמתבצעSEARCH, נוכל להוציא את המפתח ממקומו ברשימה הדו מקושרת, ולהכניס אותו לסוף הרשימה (באיבר זה השתמשו אחרון, ולכן נפנה אותו עקב חוסר מקום – אחרון.), וכמובן שנעדכן את

ההצבעה שבתוך טבלת הגיבוב, למיקום החדש של האיבר. כל הוספה של נתון, תוסיף אותו לסוף הרשימה, כמו בתור.

Insert (S,x)	**Search (S,K)**
Key ← S.HeadList(data) S.HeadList ← S.HeadList(next) S.Headlist(prev)←Null Delete(S.HTable(Key)) new NewListElement NewListElement(data)←key(x) S.TailList(next) ←NewListElement NewListElement(next) ← NULL NewListElement(prev)←S.TailList S.TailList ← NewListElemnt Insert(S.Htable, x, key(x), NewListElement)	 (x,ListPtr) ←Search(HashTable, K) ListPtr(prev)(next)←ListPtr(next) ListPtr(next)(prev)←ListPtr(prev) S.TailList(next)←ListPtr ListPtr(prev)←S.TailList S.TailList←ListPtr S.TailList(next)←NULL Return (x)