

תרגיל 3 – רשימות מקושרות

1. נתונה רשימה מקושרת דו כיוונית A.

- א. (20 נק') כתבו אלגוריתם היוצר רשימה מקושרת חדשה B אשר מכילה אך ורק איברים מ-A, אשר שווים להפרש האיבר העוקב והאיבר הקודם להם ב-A. (שימו לב, אם ההפרש לא מוגדר כי אין לאיבר קודם או עוקב, האיבר לא נכלל ברשימה B).
- ב. (10 נק') מהו סדר הגודל של מספר הפעולות המתבצעות? נמקו.
- ג. (15 נק') כיצד הייתה משתנה התשובה לסעיפים א-ב במידה והרשימות הקלט והפלט היו רשימות חד-כיווניות?
- ד. (15 נק') כיצד הייתה משתנה התשובה לסעיף א במידה ולא היינו רוצים ליצור רשימה חדשה, אלא לשנות את רשימת הקלט?

פתרון

- א. כדי לבדוק שהאיבר הנוכחי שווה להפרש האיבר הקודם והאיבר העוקב לו נעבור על הרשימה בעזרת מצביע curr החל מהאיבר השני.
3 מצביעים: prev, curr, seq.
כאשר יש פחות משלושה איברים ברשימה תוחזר רשימה ריקה. אחרת, כל פעם נבצע בדיקה אם:
 $curr \rightarrow data = (curr \rightarrow prev \rightarrow data) - (curr \rightarrow next \rightarrow data)$
אם כן, ניצור איבר חדש שיכיל את ה-data של curr, ובכל מקרה נקדם את המצביע בצעד אחד. האלגוריתם יסיים את ריצתו כאשר next=NULL.

```
Node_type* subList(Node_type* a)
{
    Node_type *curr = a;
    Node_type *headB = NULL, *currB = NULL, *newB;

    // בדיקה אם יש לפחות 3 איברים ברשימה, וקידום המצביע לאיבר השני
    if (curr != NULL)
    {
        curr = curr->next;
        if (curr == NULL || curr->next == NULL)
            return headB;
    }
    else
        return headB;

    // מעבר על הרשימה
    while (curr->next != NULL)
    {
        if (curr->data == (curr->prev->data) - (curr->next->data)) // התנאי המבוקש מתקיים
        {
            newB = MakeNode(curr->data);
```

```

        InsertBetween(newB, &headB, &currB, NULL);
    }
    curr = curr->next;
}

return headB;
}

```

הוכחת נכונות:

מאחר והרשימה דו-כיוונית, ואנו מתחילים מהאיבר השני, המשתנים $(curr \rightarrow next \rightarrow data)$, $(curr \rightarrow prev \rightarrow data)$ ו- $curr \rightarrow data$ מוגדרים לכל אורך הרשימה. האלגוריתם ייצור איבר חדש ויכניסו ל-B רק כאשר:

$$curr \rightarrow data == prev \rightarrow data - seq \rightarrow data$$

ולכן האלגוריתם עובד כראוי.

ב. סדר גודל הפעולות המתבצעות הוא כסדר מספר האיברים ברשימה A. האלגוריתם עובר על כל האיברים ב-A ועל כל איבר מבצע מספר קבוע של פעולות (העלות של MakeNode ו-InsertBetween היא $\Theta(1)$ ולכן אם מספר האיברים ב-A הוא $\Theta(n)$ אז סדר גודל הפעולות הוא גם כן $\Theta(n)$.

ג. במידה והרשימה הייתה חד-כיוונית היינו צריכים להחזיק מצביע נוסף לאיבר הקודם, prev, לאתחל אותו לאיבר הראשון ברשימה ולערוך את השינויים הבאים:

- לבצע את הבדיקה הבאה:

$$curr \rightarrow data == (prev \rightarrow data) - (curr \rightarrow next \rightarrow data)$$

- בכל קידום של curr בסוף לולאת ה-while לקדם גם את המצביע הנוסף:

$$prev = prev \rightarrow next$$

זמן הריצה לא היה משתנה.

ד. במידה והיינו רוצים לשנות את הרשימה הנוכחית כך שיישארו רק האיברים השווים להפרש האיבר הקודם והאיבר העוקב, היינו צריכים לבצע באלגוריתם את השינויים הבאים:

- a. נצטרך שני משתנים חדשים:
 - i. prevVal – במקום לגשת לאיבר הקודם נשמור רק את הערך שלו, ובהתאם התנאי ישתנה באופן הבא: $curr \rightarrow data = prevVal - curr \rightarrow next \rightarrow data$
 - ii. currA – מצביע לאיבר הנוכחי ברשימה המעודכנת של A בה יש רק איברים שמקיימים את התנאי.
 - b. תחילה היינו שומרים את ערכו של האיבר הראשון ברשימה בתוך prevVal ומוחקים את האיבר מהרשימה. אחר כך היינו מחפשים את האיבר הראשון שמקיים את התנאי. כל עוד האיבר curr לא מקיים את התנאי מוחקים אותו מראש הרשימה. כאשר מוצאים את האיבר הראשון שמקיים את התנאי מגדירים ש- currA יצביע עליו.
 - c. מעבר על שאר הרשימה יתבצע באופן הבא:
 - i. במידה והתנאי מתקיים מגדירים ש- currA יצביע עליו.
 - ii. אחרת, יש למחוק את האיבר curr מהרשימה. נעשה זאת ע"י שימוש בפרונקציה Delete(currA, curr) (מוחקת את הפרמטר השני מהרשימה, והפרמטר הראשון נמצא ברשימה לפני הפרמטר השני)
- בכל מקרה מקדמים את הערכים prevVal ואחר כך curr.

d. כאשר נגיע לסוף הרשימה נגדיר ש- $currA \rightarrow next = NULL$ כדי שלא יצביע לאיבר האחרון ברשימה שבוודאות לא מקיים את התנאי כי אין לו איבר עוקב. זמן הריצה לא היה משתנה, כי עדיין עוברים על כל האיברים ברשימה.

א. נתונה רשימה מקושרת חד כיוונית A.

א. (20 נק') הציעו אלגוריתם בסיבוכיות זמן $O(n)$ ובעל דרישת זיכרון קבועה (כלומר, שאינה תלויה ב- n), ההופכת את סדר הרשימה (האיבר הראשון יהפוך להיות האיבר האחרון, האיבר השני יהפוך להיות האיבר לפני אחרון, וכו').

הערה – בתרגיל כזה עליכם להראות שדרישות הסיבוכיות מתקיימות.

פתרון:

השיטה היא להפוך את כיוון החצים כך ש- $next$ יהיה $prev$, כלומר – יצביע לאיבר ה"קודם" ולא לאיבר הבא, וכך בעצם להפוך את הרשימה. כל רשימה מסתיימת ב- $NULL$, אז נדאג ש- $next$ של האיבר הראשון יצביע בעצמו ל- $NULL$ ו- $head$ (המצביע לראש הרשימה) יצביע לאיבר האחרון.

ניצור שלושה מצביעי עזר: $prev$, $curr$, $next$
כל עוד לא נגיע לסוף הרשימה, נמשיך בהשמת המצביעים בסוף, נחזיר את $prev$ כ- $head$.
המימוש עצמו יראה כך:

```
Node_type* ReverseList2(Node_type* list)
{
    Node_type* prev,*curr,*next;
    prev=NULL,next=NULL;
    curr=list;
    while(curr!=NULL)
    {
        next = curr->next; // שלא נאבד אותה.
        curr->next = prev; // היפוך כיוון ההצבעה בתוך הרשימה.
        prev = curr; // קידום המצביע לרשימה ההפוכה שיצרנו אל האיבר שהוספנו כעת בראש הרשימה.
        curr = next; // הכנה לאיטרציה הבאה – קידום המצביע אל האיבר הבא ברשימה המקורית.
    }
    return prev;
}
```

סיבוכיות:

מבחינת הזמן: עוברים על הרשימה בדיוק פעם אחת, ולכן זה אלגוריתם בסיבוכיות ליניארית, כנדרש.
מבחינת הזיכרון: אין יצירה של מבנה נתונים נוסף אלא שימוש בשלושה מצביעים בלבד = דרישת זיכרון קבועה.

ב. (20 נק') בהנחה שהנתונים בשדות המידע ברשימה הם מספרים שלמים ובהינתן מספר שלם Z כקלט נוסף, יש לכתוב אלגוריתם שמחזיר TRUE אם קיים ברשימה זוג איברים שסכומם, Z ו- FALSE אחרת. יש להראות נכונות האלגוריתם ולנתח את סיבוכיות הזמן שלו.