<u>עבודה 2 מבני נתונים חלק ג'</u>

דן ירדן 316611854

אלמוג למאי 316413368

1. חיפוש lookup

```java
    @Override
    public FloorsArrayLink lookup(double key) {
        FloorsArrayLink result =(search(key));
        if (result.getKey()==key) return result;

        else return null;
    }
    //side assistant function
    private FloorsArrayLink search(double key) {
        FloorsArrayLink[] currentArr=firstLink.frontArr;
        FloorsArrayLink currentLink=firstLink;
        int i =arrMaxSize;
        while (i>=0) {

            while(currentArr[i].getKey()<=key) {
                if (currentArr[i].getKey()==key) return currentArr[i];
                currentLink=currentArr[i];
                currentArr=currentLink.frontArr;
            }
            i=i-1;
        }
        return currentLink;
    }
}
```

א. תיאור האלגוריתם:

נקרא לפונקציה בשם **search** שתמצא את החוליה המבוקשת, או את השכנה השמאלית שלה במידה והחולייה המבוקשת לא נמצאת.

אם היא החוליה כן נמצאת, lookup תחזיר אותה, אם לא היא תחזיר null.

האלגוריתם של search:

1. נתבונן במערך הקדמי של החולייה הראשונה **firstLink** (מינוס אינסוף) במקום השווה בערכו לגודל המערך הגדול ביותר ברשימה, נסמן את המערך ב- **currentArr**, ונגדיר אינדקס התחלתי : $\mathbf{i} \rightarrow$ **arrMaxSize**
2. **כל עוד i >= 0** , נבצע את הפעולות הבאות:
3. **כל עוד** המפתח של החולייה במקום ה-i במערך של החוליה הנוכחית (**currentLink**) קטן או שווה למפתח המבוקש (**key**), נתקדם לחוליה זו, כלומר :

$$currentLink \leftarrow currentArr[i]$$

$$currentArr \leftarrow currentLink.frontArr$$

(משמע, נחליף את החוליה הנוכחית בה)

4. **אם** מצאנו מפתח שווה לו, נחזיר את **currentLink**.
5. נחזיר את **currentLink** (יהיה השכן השמאלי של המפתח המבוקש במידה ולא נמצא במערך)

ב. תיאור זמן ריצה:

נשים לב כי במקרה הגרוע ביותר הלולאה בסעיף 2 תתבצע i פעמים, i=logn עבור i מקסימלי מכיוון שהמערך המקסימלי על פי הגדרה בסעיף ג' יהיה המספר המקסימלי x+1 שמקיים $n \bmod 2^x = 0$. x כזה הוא בהכרח logn.

נשים לב כי עד להגעה לחוליה המבוקשת מתבצעות פעולות ירידה באינדקס i, ועבור כל ירידה כזו מתבצעות פעולות התקדמות לחוליות הבאות (ימינה, לכיוון החוליה הקיצונית ימנית).

ברור כי מבחינת הקוד, מלבד לולאות ה- while, כל הפעולות מתבצעות בזמן ריצה קבוע.

לולאת הwhile הראשונה תתבצע לכל היותר log n פעמים עבור קלט בגודל n כי בכל איטרציה שלה i יורד ב-1,

נטען כעת כי לולאת ה- while הפנימית תתבצע מספר קבוע של פעמים.

טענת עזר: עבור כל איבר i ברשימה המקיימת את התנאים הנ"ל, אם גובה המערך של i הוא h, אז אחרי i, (לאו דווקא מיד אחריו) יגיע איבר עם גובה מערך גדול משל i, לפני שיגיע איבר עם גודל מערך זהה לשל i.

הוכחה: יהי i איבר שרירותי ברשימה עם גודל מערך h+1. אז i מתחלק ב 2^h ללא שארית לפי הגדרת הרשימה.

לכן ניתן לרשום אותו כך: $i = q \cdot 2^h \quad q \in \mathbb{N}$

נעיר ש-i לא מתחלק ב- 2^{h+1} כי אילו היה, היה לו מערך מגודל h+2.

בנוסף q אי-זוגי משום שאילו היה זוגי היה מתחלק ב-2 ואז i היה מתחלק ב 2^{h+1}.

נשים לב שהאיבר **הבא הכי קטן** שמתחלק ב- 2^h הינו $q \cdot 2^h + 2^h$, כלומר: $(1+q) \cdot 2^h$ אבל נתון ש-q אי-זוגי ולכן q+1 זוגי, מכאן שניתן לכתוב את האיבר הזה כ- $\left(\frac{1+q}{2}\right) \cdot 2^{h+1}\ ,\ \frac{1+q}{2} \in \mathbb{N}$, כלומר איבר זה מתחלק ב- 2^{h+1} ולכן גודל המערך שלו יהיה h+2. מכאן שלפני שיופיע איבר עם מערך מגודל h, יופיע איבר עם מערך בגודל h+1.

כעת נראה שמספר הכניסות ללולאת ה-while הפנימית הוא קבוע:

נראה שבעצם לא ניכנס ללולאה פעמיים ברצף.

נניח שנכנסנו ללולאה פעם אחת, אז על מנת שניכנס אליה שוב צריך שיתקיים ש $currentArr[i].key \le key$ וגם שגודל המערך של החולייה הנוכחית קטן או שווה לגודל המערך של החולייה הבאה (שמקבלת מצביע מהמערך הנוכחי במקום ה-i).

על פי טענת העזר נסיק שהמערך הבא שנוכל להצביע עליו יהיה גדול מהנוכחי, כי לא ייתכן שיהיו שני מערכים באותו גובה בלי מערך שלישי שגבוה משניהם וממוקם ביניהם.

ברור כי כניסה נוספת ללולאה לא אפשרית, משום שהתחלנו את החיפוש מגובה המערך המקסימלי בחולייה השמאלית הראשונה, ואילו היה מערך שגדול מהמערך הנוכחי עם מפתח קטן מהמפתח שאותו אנו מחפשים, אז היינו מגיעים למערך הגדול לפני שהיינו מגיעים למערך הנוכחי על ידי הצבעה ישירה אליו מחוליה קודמת.

לכן זמן הריצה של הלולאה הפנימית הוא קבוע, וסך זמן הריצה של הפונקציה search ואיתה גם lookup יסתכם ב-(O(log n.

2. הכנסה insert

החלק הראשון של insert:

```
@Override
public void insert(double key, int arrSize) {
    if (size==maxSize)
        return;
    //we don't want to insert values once the list is at its max capacity.
        this.size=size+1;
        if(arrSize>this.arrMaxSize) {
            arrMaxSize=arrSize;
        }//updating the new array maximal size in case we inserted an array with a bigger size array.

    FloorsArrayLink prevLink =search(key);
    //using the function "search" to determine where the link should be inserted (it gives us its future left neighbor-LN).

    FloorsArrayLink newLink = new FloorsArrayLink(key, arrSize); //creating the new link.
    (prevLink.RN).LN=newLink;
    newLink.RN=prevLink.RN;
    newLink.LN=prevLink;
    prevLink.RN=newLink;
```

בקוד של פונקציית ההכנסה מתבצעות פעולות בזמן קבוע, נתמקד בכל הפעולות שעלותן היא מעבר לזמן ריצה קבוע והן:

1) עידכון המצביעים
2) יצירת החוליה החדשה
3) קריאה לפונקציית search.

יצירת החוליה החדשה מתבצעת ב-O(log n) כי היא תלויה בגודל המערך של החוליה כאשר כפי שצוין לפני כן, הגודל המקסימלי של מערך של חוליה ברשימה הוא log n (עבור n שמייצג גודל מקסימלי של הרשימה).
הוכח בסעיף קודם כי **זמן הריצה של search הוא O(log n)** עבור רשימה בגודל n.

להלן הקוד של עדכון המצביעים, העידכון מתבצע באופן סימטרי עבור מצביעים אחוריים (שמאליים) ועבור מצביעים קדמיים (ימניים), לכן נתמקד כאן רק בעידכון האחורי אך נזכור בסוף לכפול את הזמן שקיבלנו ב- 2 ולהוסיף את זמן הריצה של יצירת החוליה ואת זמן הריצה של פונקציית החיפוש.

```
int j=1;
while(prevLink!=null & j<=arrSize) {
    if(prevLink.getArrSize()>=j) {//we only want to update arrays that have spots "higher" then what we updated before,
                                  //the index j holds the highest point reached so far.
        int min=Math.min(arrSize, prevLink.getArrSize());
        while(j<=min) {
            prevLink.setNext(j,newLink);
            newLink.setPrev(j,prevLink);
            j++;
        }
    }
    prevLink=prevLink.getPrev(prevLink.getArrSize());
}
```

זמן הריצה של קטע הקוד הנ"ל הינו O(log n) משום שמספר הפעולות שיתבצעו יהיה כגודל המערך של חוליה המוכנסת. תאורתית, נסתכל על קישור החוליה החדשה לחוליות שנמצאות מאחוריה ברשימה כהתקדמות שמאלה ולמעלה ברשימה, כל פעם שנגיע לחוליה חדשה נעבור למצביע האחורי שלה בתא הגבוה ביותר במערך שלה. בדרך זו, כל "התקדמות שמאלה" ברשימה בהכרח תביא אותנו למערך "גבוה יותר" כי על פי טענת העזר בשאלה 1, התא העליון במערך של החוליה בהכרח יצביע לחוליה עם מערך גבוה משל החוליה הנוכחית. תנאי זה מבטיח שלא נתקדם "פעמיים באותו גובה" ולכן מספר הפעמים

שלולאה זו תפעל ניתן לחסימה מלמעלה על ידי $O(\log n)$. ניתן להבטיח שלא נבצע צעדים מעבר לגובה של מערך החוליה המוכנסת משום שבכל הצבעה למערך גבוה מ-j , j יתקדם עד לגובה אותו מערך או המערך הנוכחי על ידי הלולאה הפנימית, וכאשר יגיע לגובה של המערך של הלולאה המוכנסת, שתי הלולאות תפסקנה לעבוד. מכאן שזמן הריצה של קטע קוד זה הוא $O(\log n)$ (כגובה המערך המקסימלי שיכול להיות לחוליה מוכנסת).

לכן בלי להתחשב בתרומה זניחה של פעולות עם זמן ריצה קבוע נקבל בסה"כ כי זמן הריצה של פונקציית ההכנסה הוא:
$O(\log n) + O(\log n) + O(\log n) + O(\log n)$
כלומר $O(\log n)$.

3. הסרה remove

```java
@Override
public void remove(FloorsArrayLink toRemove) {
    this.size=size-1;
    if(this.size==0) {
        arrMaxSize=0;
    }
    else {
        int k =toRemove.getArrSize();
        if (k==arrMaxSize) {//updating the new maximal array.
            k=k-1;
            while(toRemove.backArr[k]==this.firstLink & toRemove.frontArr[k]==this.lastLink) {
                k=k-1;//we want to locate the next cell that doesn't point to the last or first link.
            }
            arrMaxSize=Math.min(toRemove.backArr[k].getArrSize(), toRemove.frontArr[k].getArrSize());
        }//cells are only pointing to links with higher or equal arrays, so if one is bigger than the other at this point, than it is the first or the last.
    }
    FloorsArrayLink prev =toRemove.LN; //updating the neighbors field
    FloorsArrayLink next =toRemove.RN;
    prev.RN = next;
    next.LN =prev;
    for(int i=1;i<=toRemove.getArrSize();i++) {
        FloorsArrayLink left = toRemove.getPrev(i);
        FloorsArrayLink right = toRemove.getNext(i);
        left.setNext(i,right);
        right.setPrev(i,left);
    }
}
```

א. תיאור האלגוריתם:

1. עדכון גודל הרשימה לאחר הסרת האיבר this.size← size-1. **(שורה1)**
2. **אם** גודל הרשימה לאחר שורה 1 הוא 0, סימן שהרשימה התרוקנה ולכן גודל המערך המקסימלי הוא 0, כלומר: arrMaxSize=0
3. **אחרת**
4. **אם** המערך שמסירים הוא המערך מקסימלי, "נטייל" בלולאה מלמעלה למטה במערכים של החוליה המוסרת **כל עוד** שני המצביעים מצביעים למערכים הקיצוניים. נבדוק מה התא הבא שמצביע למערך שאינו אחד מהמערכים הקיצוניים (**lastLink, firstLink**) על פי יחסי המצביעים ברשימה, התא המבוקש יצביע למערך השני בגודלו.
5. נשמור גודל מערך של תא זה בשדה המבוקש:
arrayMaxSize← min(toRemove.backArray[k].ArraySize, toRemove.frontArr[k].ArraySize)
(*ניקח את המינימלי מבין הערכים כי נרצה להימנע מהשמה של גודל המערכים הקיצוניים בערך זה)
6. נעדכן את שדות השכנים RN, LN של החוליות שמצדי החוליה שהוסרה שיצביעו זו לזו בהתאמה.
7. בלולאת **for** נעדכן מצביעים של כל החוליות שהצביעו לחוליה המוסרת בצורה הבאה:
עבור חוליה משמאל לחוליה המוסרת שהתא ה-i שלה הצביע לחוליה המוסרת, נשנה את ההצבעה של תא זה להצביע כעת לחוליה שהתא ה-i במערך הימני של החוליה המוסרת מצביע אליו. נעשה זאת באופן סימטרי גם לחוליות מימין לחוליה המוסרת ובכך נשמור על סדר המצביעים ברשימה.

ב. זמן ריצה:

נבדוק זמן ריצה במקרה הגרוע ביותר, כשהחוליה שמסירים היא בעלת המערכים הגדולים ביותר.

<u>סעיפים 1-3</u>: זמן ריצה קבוע $O(1)$ כי כל הפעולות השורות אלו מתבצעות בזמן ריצה קבוע בלי תלות בקלט.

<u>סעיפים 4-5</u>: זמן ריצה: $O(\log n)$
במקרה הגרוע ביותר החוליה אותה מוחקים היא החוליה בעלת אורך המערכים הגדול ביותר, כלומר החוליה בעלת מערכים בגובה log n. הזמן הגרוע ביותר שיכול לקחת ללולאת ה-while הינו כגודל המערך של החוליה המוסרת, על ידי מעבר על כל תאיה.

<u>סעיף 5</u>: זמן ריצה קבוע $O(1)$ כי כל הפעולות השורות אלו מתבצעות בזמן ריצה קבוע בלי תלות בקלט.

<u>סעיפים 6-7</u>:
זמן הריצה של שורות אלו הינו $O(\log n)$.
זמן זה הוא בהתאם לאורכם של מערכי המצביעים של החוליה שנמחקה כי עבור כל תא במערכיה מתבצעות פעולות בזמן ריצה קבוע כדי לעדכן את המצביעים, דהיינו $O(1)$. במקרה הגרוע ביותר הייתה החוליה המוסרת בעלת מערכים בגודל המקסימלי האפשרי.
הגודל המקסימלי של מערך מצביעים (שאינו מערך של חוליות אינסוף)הוא log n, משום שגודל מערך של חוליה במקום ה-n הוא x+1, עבור x שמקיים: $n \, mod(2^x) = 0$, ברור כי x מקסימלי כזה יהיה logn כי עבורו נקבל n mod n. ולפיכך זמן הריצה של הפונקציה פרופורציוני לאורך זה.
בסך הכל, זמן הריצה של הפונקציה remove הוא: $O(\log n) + O(\log n) + O(1)$ כלומר : $O(\log n)$.

4. גודל המבנה

ננתח את גודל המבנה של הרשימה כאשר היא מכילה n מפתחות. נשים לב כי עבור כל חוליה במערך יידרש גודל קבוע של מקום O(1) , כפול n חוליות, זה O(n).

לכל חוליה קיימים שני מערכים, אשר גודלם תלוי במיקום החוליה ברשימה. בכל תא במערכים ישנו מצביע שנשמר גם הוא ב- O(1) ולכן גודל של מערכי החוליה יכפיל את כמות הזיכרון שלוקח לשמור את אותה חוליה פי 2 כפול גודל המערך של אותה חוליה.

ניתן לחשוב על הבעיה גם כספירת מספר התאים החל מהשורה השניה, הכפלת התוצאה ב-2, והוספת n לתוצאה, שמבטא את כמות החוליות בתחתית.

אז עבור השורה השניה מלמטה ברור כי ישנם $\frac{n}{2}$ זוגות מערכים שהם לפחות בגובה זה, כי כל חוליה באינדקס זוגי תהיה בעלת מערך שיגיע לגובה זה.

עבור השורה השלישית ישנם $\frac{n}{4}$ מערכים כי רק חוליות באינדקסים שמתחלקים ב-4 ללא שארית יגיעו לגובה זה.

ניתן לראות את דפוס חוזר והוא: עבור השורה ה-i קיימים $\frac{n}{2^i}$ זוגות של תאי מערכים.

נרצה לסכום את מספר התאים בכל הרמות על מנת לדעת מהו גודל הזיכרון שהמבנה ידרוש עבור ערכים מאוד גדולים של n.

כאשר n שואף לאינסוף גם גודל המערך המקסימלי log n שואף לאינסוף, ועל כן ישנו צורך לבדוק מהו גודל הזיכרון הדרוש עבור מצב בוא גודל המערכים המקסימליים של החוליות שואף לאינסוף.

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots+\frac{1}{2^i}=\frac{\frac{1}{2}}{1-\frac{1}{2}}=1$$

נקבל את הטור ההנדסי הבא:

נכפול סכום זה ב- 2 עבור שני מצביעים בכל גובה, עד כה נקבל 2. כעת נכפול ב-n כפי שהוסבר לעיל (2n עד כה) , נוסיף כעת את ה-n שמבטא את כמות החוליות בבסיס הרשימה. סה"כ נקבל 3n שזה שווה ל O(n).