Data Structure Exercise 2 - Part C

הסבר על השדות בקוד:

FloorArrayLink :

arrForward – רשימת המצביעים קדימה של החוליה הנוכחית (next).
arrBackwards – רשימת המצביעים אחורה של החוליה הנוכחית (prev).
key – המפתח של החוליה.
arrSize – גודל המערכים של החוליה.

FloorArrayList :

maxSize – כמות מקסימלית של אברי הרשימה (חסם עליון על כמות אברי הרשימה).
size – גודל האיברים במערך.
first – החוליה הראשונה של המערך, ערך המפתח Negative_Infinity.
last – החוליה האחרונה של המערך, ערך המפתח Positive_Infinity.
min – ערך המפתח הקטן ביותר ברשימה.
max – ערך המפתח הגדול ביותר ברשימה.
currentMaxArray – גודל המערך של החוליה הגדול ביותר ברשימה.

1. א. תיאור האלגוריתם LookUp :

האלגוריתם מקבל key מסוג double וצריך להחזיר את החוליה שה – key שלה שווה ממש ל key שהאלגוריתם מקבל (אם קיים key כזה), ואם לא קיים להחזיר null. האלגוריתם קורא תחילה לפונקציה search.

תיאור הפונקציה search:

האלגוריתם יתחיל את תהליך החיפוש מהחולייה שהמפתח שלה הוא Negative_Infinity ונגדיר משתנה מסוג FloorArrayLink שייקרא current שיקבל תחילה כמצביע את החוליה הזו. מכאן נגדיר את תחילת החיפוש להיות החל מהתא ה CurrentMaxArray של מערכי ה next של החוליה current. את הערך המספרי של גודל המערך הגדול ביותר ברשימה נשמור במשתנה חדש שנקרא לו index. נבצע לולאת while שתפעל כל עוד index>0 . בתחילת הלולאה נשאל (**if**) האם ה key של התא במקום ה index של מערכי ה next של החוליה current **קטן או שווה** מ key. אם כן, נשים לב שה- key אותו אנו מחפשים אמור להיות מימין ל next של התא ה index ולכן נעדכן את current להיות ה next במקום ה - index. אחרת (**else**), נוריד את index ב – 1 ונחזור על התהליך מחדש (במידה ותנאי הלולאה ממשיך להתקיים). בסוף התהליך כאשר נצא מהלולאה במשתנה current יהיה מצביע לחוליה אחת מהרשימה.

המשך תיאור האלגוריתם look-up:

נבדוק את ה- key של החוליה שחזרה מהפונקציה search, אם הוא יהיה שווה ל key שאותו חיפשנו נחזיר את current אחרת נחזיר null .

* נשים לב שאם קיימת חוליה בעלת מפתח key נקבל בוודאות את החוליה הזו שהרי בכל פעם אנו משווים בין גדלי המפתחות וברגע שנגיע לגודל המפתח שהוא כגודלו של המפתח שלנו המשתנה current לא יתעדכן יותר משום שמימין לחוליה הנ״ל נמצאים מפתחות גדולים ממש מזה שאנו מחפשים וה index ירד עד יציאה מהלולאה.

* במידה ולא קיימת חוליה עם מפתח key שאנו מחפשים, נקבל ב current בסוף התהליך את החוליה בעלת מפתח הגדול ביותר שקטן מ key אותו אנו מחפשים. ההשוואה האחרונה באלגוריתם תגרום להחזרת null ולא את current .

* **דגש:** בקוד כתבנו את כל החלק של החיפוש של האלגוריתם בפונקציה שנקראת search ואת החלק באלגוריתם שמחזיר את החוליה אם המפתח של החוליה תואם את מה שחפשנו או null אם המפתח של החוליה אינו תואם את מה שחיפשנו בפונקציה שנקראת lookUp. זאת משום שהיינו צריכים עבור הפונקציה insert את הפונקציה search ללא החזרת null .

ב. ניתוח זמן ריצה במקרה הגרוע ביותר:

תחילה נשים לב שבכל מקרה באלגוריתם המתואר לעיל נצטרך לבצע לפחות maxArraySize פעולות, שהרי ה index של לולאת ה while מגיע בסופו של דבר ל – 0 וכמתואר לעיל מתחיל מ maxArraySize-. לכן, המקרה הגרוע ביותר יהיה המקרה בו נבצע כמה שיותר פעולות של עדכון ה current. נזכיר שכאשר אנו מעדכנים את ה current אין אנו מעדכנים את המשתנה index. נסמן את החוליה עם גודל המערך הכי גדול. נשים לב שכאשר אנו מחפשים איבר כלשהו אם ה-key של האיבר שאנו מכניסים גדול מהkey של החוליה שסימנו, נשים לב שבמקרה הגרוע כעת יש לנו לכל היותר חצי מערך לחפש בו (נניח בשלילה שהיה יותר מחצי מערך לחפש בו אז החוליה שסימנו לא היתה הגדולה ביותר - סתירה). כעת עדכנו את ה current לחוליה שסימנו, ונשאר לנו לכל היותר חצי מערך לחפש בו. כעת לפי אותו אלגוריתם בדיוק נחפש בחצי המערך הזה, ולכן במקרה הגרוע שוב נצטרך לעדכן את הcurrent ולחפש בחצי מערך. לכן נשים לב שבכל עדכון של current במקרה הגרוע אנו חוצים את המערך. ובאופן כללי כאשר גודל המערך הוא n, נחלק אותו לחצאים $\log n$ פעמים ולכן זה מספר העדכונים של הcurrent. ולכן זמן הריצה יהיה $\theta(maxArraySize + \log(n))$ ומכיוון שגודל המערך המקסימלי במערך לפי הנתון הוא log(n) נקבל שזמן הריצה:

$$\theta(\log n)$$

2. ניתוח זמן ריצה הדוק ביותר של הכנסה:

ראשית, נקרא לפונקציה *search* כדי למצוא את המיקום שבו עלינו להכניס את האיבר. הפעולה הזו עלתה לנו לפי הניתוח לעיל $\log n$. כעת ברשותנו המיקום שבו עלינו להכניס את האיבר. כאשר נקבל מפתח וגודל מערך, נבדוק אם זה משפיע על השדות של ה*min* ו *max* במבנה ונעדכן בהתאם, פעולה זו תקח סדר גודל של קבוע. לאחר שמצאנו את המיקום נותר לעדכן את המצביעים הרלוונטיים המושפעים מהכנסת החוליה החדשה. ניצור שני משתנים מסוג *FloorArrayLink* שיצביעו תחילה לחוליה העוקבת *next* של החוליה ולחוליה הקודמת *prev* של החוליה ונקרא להן *next* ו *prev* בהתאמה. לאחר מכן נעבור על האינדקסים של המערכים של החוליה החדשה בלולאת *while* שתבצע עדכון ל מערך ה*next* ולמערך ה*prev* באופן כזה שכל עוד *index* במערך קטן או שווה לגודל המערך של החוליה העוקבת (או הקודמת בהתאמה) המצביע של החוליה במקום ה *index* יצביע לחוליה העוקבת (או הקודמת בהתאמה) ונגדיל כל פעם את *index* ב – 1. במידה ובשלב כלשהו *index* יהיה גדול מגודל המערך של החוליה העוקבת או החוליה הקודמת, נקדם את המצביע של החוליות *next* ו*prev* להיות *next.getNext(index-1)* או *prev.getPrev(index -1)* וכך נמשיך בתהליך עד ש *index* יהיה גדול ממש מגודל המערך של החוליה אותה אנו מכניסים. נצטרך לבצע עדכון לכל אחת מהחוליות כמספר האיטרציות של לולאת ה *while* משמע, *4maxArraySize* פעמים (עדכון מצביעים לחוליה במערך *next* במערך *prev* ולחוליות במערכים שכל אחד מהנל מצביע אליהן). לכן בסך הכל יתבצעו :

$$maxArraySize \leq 4maxArraySize + c \leq 5maxArraySize$$

ולכן סך הכל זמן הריצה יהיה :

$$\Theta(maxArraySize + \log n)$$

ומכיוון שגודל המערך המקסימלי במערך לפי הנתון הוא log(n) נקבל שזמן הריצה :

$$\theta(\log n)$$

3. א. תיאור האלגוריתם של הוצאה:

האלגוריתם מקבל חולייה נסמן אותה toRemove הנמצאת ברשימה וצריך להסיר אותה מהרשימה תוך עדכון מחדש של כל המצביעים הרלוונטיים המושפעים מהמחיקה. תחילה נבדוק האם המחיקה משפיעה על השדות של הערך המקסימלי והערך המינימלי שברשימה (זה קורה אם ה key אותו אנו מוחקים מהווה מינימום או מקסימום ברשימה, את העדכון נעשה על ידי לקיחת ה next במקום ה (1) אם ה key הוא מינימום ובהתאמה את ה prev במקום ה (1) אם ה key הוא מקסימום. כדי לשמר את השדה המציין את גודל המערך הגדול ביותר ברשימה נאתחל משתנה חדש מסוג FloorArrayLink נקרא לו maxArrayLink ונבצע בו השמה של null. כעת, נרוץ בלולאת for על תאי המערכים של החולייה אותה אנו רוצים למחוק ונבצע עדכון של המצביעים באופן הבא :

תחילה נסמן ב – i את גודל האינדקס של לולאת ה for של כל איטרציה(i רץ מגודל המערך של החוליה ועד 1).ניצור משתנה בוליאני isUpdated שיסייע בהמשך לדעת האם עדכנו את השדה maxArraySize במידה וגודלו של המערך של החוליה אותה אנו מוחקים הוא maxArraySize. כעת בכל איטרציה של i, נעדכן את החוליות שנמצאות במערך של הnext במקום ה i להצביע על prev במקום ה i של החוליה אותה אנו מוחקים. באופן דומה, נעדכן את החוליות שנמצאות במערך של prev במקום ה i להצביע על next במקום ה – i של החוליה אותה אנו מוחקים. בנוסף לעדכון המצביעים נבדוק בתוך לולאת ה for באמצעות **(if)** האם החוליה אותה אנו מוחקים מכילה את המערך הגדול ביותר ברשימה. במידה ולא פשוט נמשיך לאינדקס הבא של i ונדלג על ה if במידה וכן נכנס לתוך ה if ושם יהיו שני if נוספים שיעזרו לאתר את החוליה הבאה שמכילה כעת את המערך הגדול ברשימה אחרי המחיקה. ברגע שגודלו של המערך יעודכן המשתנה הבוליאני יהיה true ולא נבצע יותר כניסה ל if האלה.

בסוף התהליך כשנצא מלולאת ה for נוריד את size ב- 1.

ב. ניתוח האלגוריתם: בהנחה והמבנה מממש את התנאי המוצג בחלק ג' ננתח את זמן ההוצאה הגרוע ביותר. זמן ההוצאה הגרוע ביותר יהיה כאשר נצטרך להוציא את החוליה הנמצאת במקום ה 2^i הגדול ביותר כאשר i הוא מספר טבעי. במקרה זה לחוליה יהיו i+1 תאים בכל מערך (לפי התנאי) והאלגוריתם יידרש גם לעבור על כל התאים וגם לעדכן את הכמות הגדולה ביותר של מצביעים – 2i+2 מצביעים. ניתוח זמן הריצה :

בתחילת האלגוריתם מבוצעות מספר פעולות שנועדו לשימור השדות min/max של המבנה, פעולות אלו לוקחות O(1) פעולות בסך הכל. לולאת ה for עוברת על כל תאי המערכים של החוליה ולכן מבצעות (i+1) פעולות (איטרציות). בתוך לולאת ה for מתבצע עדכון לכל מצביעי ה next ולכל מצביעי ה prev סה"כ קיימים i+1 תאים בכל מערך ולכן מתעדכנות בסך הכל (2*(i+1)) חוליות וזה גם מספר הפעולות

המתבצעות בחלק זה. בנוסף בכל כניסה ללולאה נבדקים התנאים של התנאי ומכיוון שיש שם שני תאים מתבצעות עוד 2i פעולות.

סה"כ נקבל לפי החישוב במקרה הגרוע ביותר שמתבצעות : 7i+7+c פעולות, ומכיוון ש –

$$i+1 \le 7i+7+c \le 8i+8$$

נקבל שזמן הריצה של האלגוריתם יהיה :

$$\Theta(i)$$

כאשר i+1 מהווה את כמות תאי המערך של החוליה אותה אנו מוחקים כמתואר לעיל,
ולכן נקבל שזמן הריצה הוא :

$$\Theta(maxArraySize)$$

ומכיוון שגודל המערך המקסימלי במערך לפי הנתון הוא log(n) נקבל שזמן הריצה :

$$\theta(\log n)$$

4. ננתח את גודל המקום שמבנה הנתונים דורש כאשר הוא מקיים את התנאי המופיע בחלק ג' ונחסום אותו מלמעלה. נגדיר את מספר המפתחות ברשימה להיות n.
מספר המפתחות הוא n לכן דרושות n חוליות ברשימה כדי לשמור את הערכים (key) שלהן.
ננתח כעת את המקום שמוסיפים המערכים שהחוליות מחזיקות. לפי התנאי בחלק ג' לחוליה במקום i יהיו x+1 תאים במערכים שלה (2 מערכים לכל חוליה) כאשר X הוא המספר הטבעי הגדול ביותר המקיים : $imod2^x = 0$. נשים לב מכך כי :

$$1 \le i \le n \longrightarrow 0 \le x \le \log(n)$$

תחילה, נחסום את כמות החוליות היכולות להיות ברשימה. מהנוסחה לעיל המקשרת בין מיקום החוליה לבין כמות תאי המערכים נסיק כי ניתן לחסום את אברי הרשימה ע"י חוליה המקיימת $imod2^{x+1} =$ 0. כאשר 2^x היא החוליה במיקום של החזקה הטבעית של 2 הגדולה ביותר ברשימה. מכך ש - $0 \le x \le$ $\log(n)$ נסיק כי החוליה החוסמת את אברי הרשימה נמצאת במיקום של $2^{\log(n)+1}$ (החוליה נמצאת מחוץ לאברי הרשימה נועדה רק כדי לחסום את כמות האיברים ברשימה.)

נבחין כי מתכונות החלוקה של המספרים הטבעיים החוליות במיקום האי זוגי (כאשר ה- x הגדול ביותר המקיים את המשוואה לעיל הוא 0) יופיעו $\frac{1}{2} * 2^{\log(n)+1}$ פעמים ולכל מערך יהיה x+1 תאים כלומר, תא 1 לכל מערך. כמו כן, החוליות שה – x הגדול ביותר המקיים את המשוואה לעיל הוא 1 יופיעו

$\left(\frac{1}{2}\right)^2 * 2^{\log(n)+1}$ פעמים ולכל מערך יהיה x+1 תאים כלומר, 2 תאים לכל מערך.

באופן דומה, החוליות שה – x הגדול ביותר המקיים את המשוואה לעיל הוא 2 יופיעו
$\left(\frac{1}{2}\right)^3 * 2^{\log(n)+1}$ פעמים ולכל מערך יהיה x+1 תאים כלומר, 3 תאים לכל מערך.

באופן כללי, נקבל שכמות התאים תהיה :

$$2 * 2^{\log(n)+1}[\sum_{k=1}^{\log(n)} k\left(\frac{1}{2}\right)^k] = 2 * 2^{\log(n)+1}[\sum_{k=1}^{\log(n)} \left(\frac{k}{2^k}\right)] = 2 * 2 * 2^{\log(n)} \sum_{n=1}^{\log(n)} \left(\frac{k}{2^k}\right) =$$

$$= 4n \sum_{n=1}^{\log(n)} \left(\frac{k}{2^k}\right) \leq 4n \sum_{n=1}^{\infty} \left(\frac{k}{2^k}\right) =$$

חישוב הסכום של הסיגמא:

תחילה, נסמן ב – s את הסכום.

$$s = \sum_{n=1}^{\infty} \left(\frac{k}{2^k}\right) = \left(\frac{1}{2^1}\right) + \left(\frac{2}{2^2}\right) + \left(\frac{3}{2^3}\right) + \left(\frac{4}{2^4}\right) + \cdots = \left(\frac{1}{2^1}\right) + \left(\frac{1+1}{2^2}\right) + \left(\frac{1+2}{2^3}\right) + \left(\frac{1+3}{2^4}\right) + \cdots =$$

$$[\left(\frac{1}{2^1}\right) + \left(\frac{1}{2^2}\right) + \left(\frac{1}{2^3}\right) + \left(\frac{1}{2^4}\right) + \cdots] + [\left(\frac{1}{2^2}\right) + \left(\frac{2}{2^3}\right) + \left(\frac{3}{2^4}\right) + (\frac{4}{2^5}) \ldots] =$$

נשים לב, שהסוגריים המרובעים הראשונים שווים ל:

$$\sum_{n=1}^{\infty} \left(\frac{1}{2^k}\right)$$

זוהי סדרה הנדסית אינסופית ו - $-1 < q < 1$ ולכן סכומה $\frac{a1}{1-q}$ במקרה שלנו, הסכום יוצא 1.

נטפל כעת בסוגריים המרובעים השניים:

$$\left[\left(\frac{1}{2^2}\right) + \left(\frac{2}{2^3}\right) + \left(\frac{3}{2^4}\right) + \left(\frac{4}{2^5}\right) \ldots\right] = \frac{1}{2} * \left[\left(\frac{1}{2^1}\right) + \left(\frac{2}{2^2}\right) + \left(\frac{3}{2^3}\right) + \left(\frac{4}{2^4}\right) \ldots\right] = \frac{1}{2}s$$

סך הכל נקבל:

$$s = 1 + \frac{1}{2}s \longrightarrow \frac{1}{2}s = 1 \longrightarrow s = 2$$

ולכן כל הביטוי יוצא בסוף של דבר:

$$8n$$

בתוספת של n החוליות ששומרות מעבר למערכים שדה של המפתח ושדה של גודל המערכים שלהן נקבל:

$$8n + n + n = 10n$$

נחסום את הביטוי הנ״ל: (ברור שכמות הזכרון הנדרשת חסומה מלמטה ע״י n מכיוון שיש n איברים ורק בשביל לשמור את המפתחות שלהן דרוש זכרון בגודל n)

$$n \leq 10n \leq 11n$$

ולכן המקום הדרוש הוא:

$$\Theta(n)$$