סיבוכיות

מאת : אבי אדרי

JOHN BRYCE
פתרונות למידה והדרכה
a matrix company

תוכן עניינים

- למה צריך לדעת סיבוכיות
- מה זה בכלל סיבוכיות ?
- איך מודדי סיבוכיות
- מה סיבוכיות זמן גישה לתא במערך ?
- מה זה בדיוק O(1)
- מוזר אבל נכון
- רשימה מקושרת כמבנה נתונים
- רשימה מקושרת חד כיוונית
- מה סיבוכיות זמן הגישה לתא ברשימה מקושרת ?
- שאלות
- תשובות

תוכן עניינים

- דרגות נוספות של סיבוכיות
- כלל: איך מחשבים סיבוכיות בפונקציה מורכבת ?
- שיפור ביצועים באמצעות מידע ממויין
- חיפוש בינארי - O(logn)
- חיפוש בעץ בינארי
- עץ אדום שחור
- סיבוכיות מיון
- סיבוכיות של בעיות בלתי פתירות
- סיכום עד כה
- תרגילי בסיס
- טיפים לפתרון התרגילים

תוכן העניינים

- סיבוכיות ב – Collections
- מה זה hash ?
- פרקטית - מה זה hash
- מה קורה אם 2 איברים ממופים לאותו התא ?
- HashMap vs TreeMap
- מתי נרצה להשתמש ב TreeMap ?
- ההבדל בין HashSet ל – TreeSet ?
- סיבוכיות ב – ArrayList
- תרגילים
- סיבוכיות ב – Collections

למה צריך לדעת סיבוכיות ?

הסיטואציה: התבקשתם לפתח קוד לפתרון בעיה.

אחת הדרישות מפתרון הבעיה היא ביצועים מהירים (לדוגמא: אלגוריתמי מסחר בכספים, אלגוריתמי גרפיקה, אלגוריתמי יירוט טילים ועוד).

הוצגו לכם על ידי חברי צוות, הצעות למספר פתרונות – כולם עובדים.

כיצד תדעו לבחור פתרון? או לחילופין, כיצד תדעו לפסול פתרונות שלא עומדים בסטנדרטים הרצויים?

למה צריך לדעת סיבוכיות ?

כמעט בכל ראיון עבודה, מוצגות שאלות הנוגעות לסיבוכיות. מבחינת המראיין, הבנה בסיבוכיות ממחישה את עומק ההבנה של התוכניתן במבני הנתונים, מה שמשפיע בסופו של דבר על ביצועי הקוד אותו הוא כותב.

סיבוכיות היא מדד שחובה לדעת מצוין, בין היתר היות ומדובר בנושא שעולה דרך קבע בראיונות עבודה.

מה זה בכלל סיבוכיות ?

סיבוכיות היא מדד גס שמטרתו לאפשר לתוכניתנים, להעריך את המהירות של האלגוריתמים שלהם, או לחילופין את המהירות המתקבלת בבחירת מבני נתונים מסוימים. כלומר, היות וכמעט כל בעיה תכנותית ניתן לפתור במספר דרכים, הבנה במדד שנקרא סיבוכיות, מאפשרת לבחור את הפתרון הטוב ביותר מבחינת מהירות.

איך מודדים סיבוכיות ?

דבר ראשון – סיבוכיות אינה מודדת מהירות, סיבוכיות מודדת בעיקר את השפעת כמות הפרמטרים על המהירות.

לדוגמא: פיתחתם תוכנה שסורקת מוצרים באיביי ומחפשת מציאות.

הבדיקות מראות שהתוכנה עובדת, העתיד נראה מבטיח, אבל...

עבור n מוצרים, האלגוריתם המשמעותי בתוכנית מבצע n! פעולות.

איך מודדים סיבוכיות ?

למעשה, הבדיקות היו על 4 מוצרים בלבד, שביצעו !4 פעולות, כלומר 24 פעולות שהסתיימו מהר מאוד, מה שיצר מצג שווא שהכל תקין.

אולם, כאשר המוצר יידרש לבצע פעולות על 10 מוצרים הוא יידרש לבצע !10 פעולות (שהן 3628800 פעולות), ואילו מעבר על כמות לא גדולה של 24 מוצרים תדרוש מהמעבד לבצע:

620448401733239439360000 פעולות...

איך מודדים סיבוכיות ?

סיבוכיות היא מדד שממחיש את מהירות האלגוריתם, כפונקציה לכמות הפריטים שאיתם הוא צריך להתמודד.

בדיקת אלגוריתם אל מול 4 פריטים הייתה נראית תקינה, אבל כמות לא גדולה של 24 פריטים תגרום לתקיעה מוחלטת של התוכנה.

איך מודדים סיבוכיות ?

סיבוכיות של אלגוריתם נמדדת לפי כמות סיבובי הלולאה שעליו לעשות, במקרה הגרוע ביותר.

דוגמא: במידה ואלגוריתם נדרש לסרוק מערך בן n תאים, ולחפש האם הערך 7 נמצא בו.

יתכן ויתמזל מזלו למצוא את המספר כבר בתא השני, אולם במקרה הגרוע ביותר המספר 7 בכלל לא ימצא במערך.

איך מודדים סיבוכיות ?

במקרה שכזה, האלגוריתם יצטרך לסרוק את כל המערך על מנת לזהות זאת (סריקה של n איברים).

סיבוכיות האלגוריתם במקרה זה תיקרא O(n).

המינוח: O של n.

מה סיבוכיות זמן גישה לתא במערך ?

נתון מערך בן מיליון תאים ושמו array (new = int[] array

(int[1000000]

שאלה : מה סיבוכיות זמן הגישה לתא ה – 500,000 ?

תרגום השאלה למילים פשוטות : עד כמה צריך לעבוד קשה

מבחינת סיבובי לולאה, על מנת להגיע לתא ה – 500,000 ?

מה סיבוכיות זמן גישה לתא במערך?

תשובה:

מאחורי שם המערך מתחבא מצביע לתא הראשון של המערך (בפועל – כתובת מספרית של התא הראשון במערך). כל איבר במערך הוא int. ב – Java איבר מסוג int תופס בדיוק 4 תאי זיכרון.

מערך הוא מבנה נתונים רציף, שיושב כמקשה אחת בזכרון. לדוגמא: מערך של 100 איברים מסוג int יתפוס בלוק רציף בזיכרון של 400 תאים (4 X 100).

מה סיבוכיות זמן גישה לתא במערך?

על בסיס שלושת הנקודות האחרונות, נקבל שניתן לחשב במדויק את מיקום התא ה – 500,000 על פי הנוסחה הבאה: שם המערך (כתובת התא הראשון) + 500,000 * 4 (גודל int בזיכרון).

חישוב זה מתבצע ללא צורך בלולאה, ולכן:

סיבוכיות זמן הגישה לתא בודד במערך היא O(1)

מינוח: O של 1.

מה זה בדיוק O(1)

O(1) אומרת למעשה שאין משמעות לכמות הקלט, המאמץ החישובי הולך להיות זניח.

סיבוכיות אינה מדד שמתיימר להיות מדוייק (מדד **גס**). סיבוכיות היא מדד שמקטלג רמות שונות של מהירות. כל אלגוריתם המבצע כמות קבועה של פעולות, הסיבוכיות שלו היא O(1).

כלומר O(100) = O(300,000) = O(5,000,000) = O(1)

מוזר אבל נכון

בסיבוכיות אין משמעות להוספה או הכפלה של מספרים קבועים.

לדוגמא: O(1) = O(1000)

לא משנה כמה איברים יש במבנה הנתונים, תמיד יתבצעו 1000 פעולות, כלומר האלגוריתם לא מושפע מגודל הקלט, כלומר – O(1)

מוזר אבל נכון

דוגמא נוספת:

אלגוריתם מבצע 44 לולאות for על n איברים, ולאחר מכן 400,000 שורות קוד בדיוק.

$O(44n+400,000) = O(n)$

הסיבוכיות היא O(n) היות ומתעלמים לחלוטין מ**מכפלה** או **הוספה** של קבועים מספריים.

רשימה מקושרת כמבנה נתונים

מבנה נתונים שבא לתת מענה לצורך:

1. כמות הנתונים לא ידועה מראש.
2. אי נכונות לבזבז זיכרון.

שימוש ברשימה מקושרת כמבנה נתונים, יכול בהחלט להחביא בתוכו אמירה "אני מוותר על מהירות לטובת חסכון בזיכרון", אולם ישנם מצבים שבהם שימוש ברשימה מקושרת לא מייצר אובדן מהירות כלל וכלל.

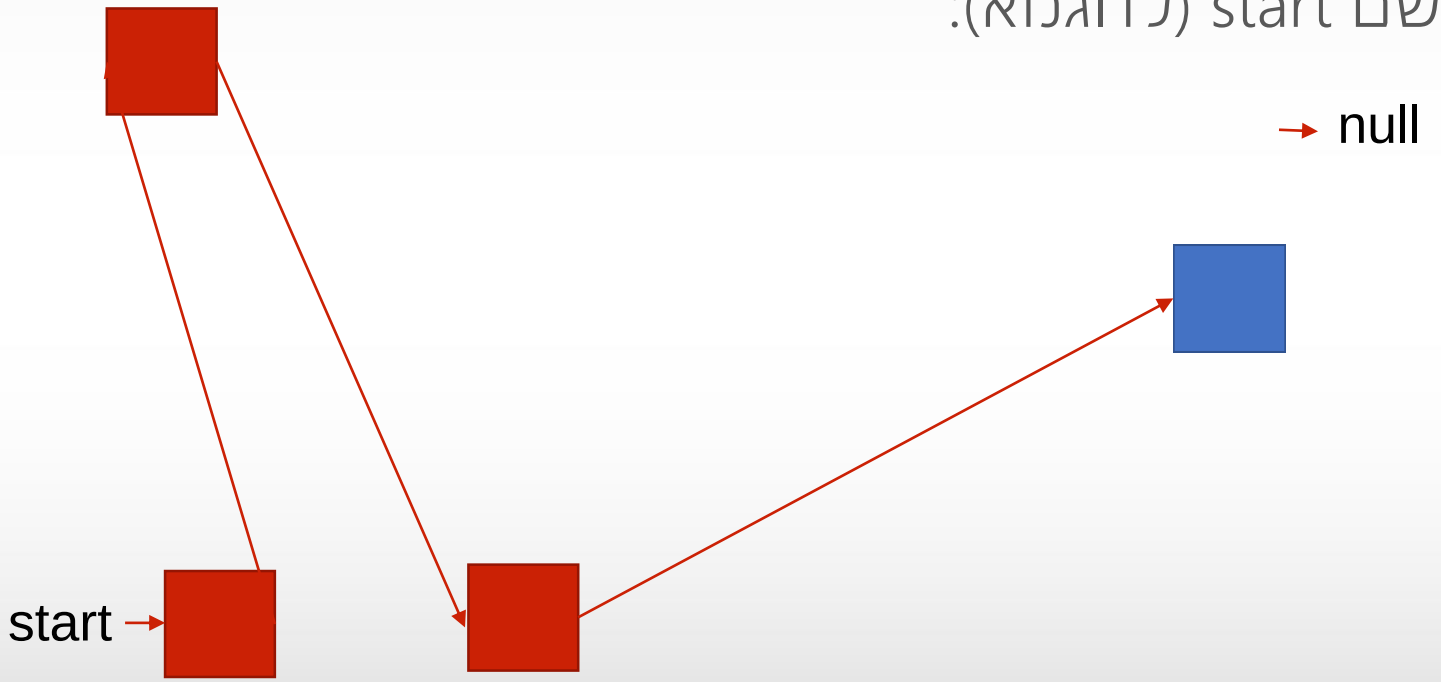
רשימה מקושרת כמבנה נתונים

בניגוד למערך, רשימה מקושרת אינה בלוק רציף בזיכרון, אלא מורכבת מאוסף תאים (אובייקטים) שפזורים בזיכרון.

בכל פעם שצריך לשמור נתון חדש, מוקצה מקום בזיכרון בזיכרון, והתא החדש מתחבר לרשימה (לפעמים בתחילת הרשימה ולפעמים בסופה, החלטה של התוכניתן ו... בהקשרים מסוימים יש משמעות רבה להחלטה הזו).

רשימה מקושרת חד כיוונית

גישה תמיד מתבצעת מתחילת הרשימה, באמצעות Pointer בשם start (לדוגמא):



מה סיבוכיות זמן הגישה לתא ברשימה מקושרת ?

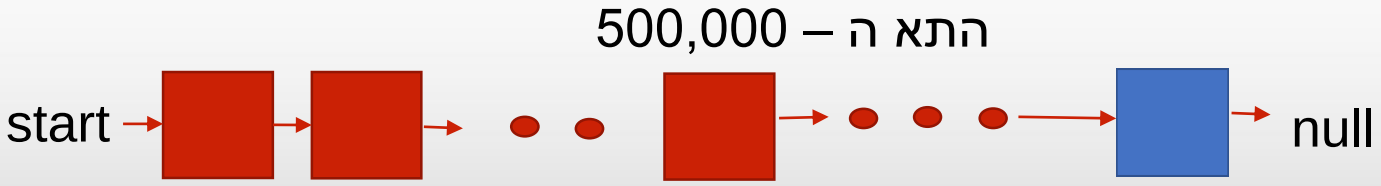
לשם נוחיות הציור, התאים מצוירים בשורה.

בפועל, כל התאים פזורים בזיכרון בצורה לא רציפה, מחוברים

בחצים (מצביעים). לכן, רציפות המידע שהתקיימה במערך

לא מתקיימת כאן כלל וכלל (שוב – במערך כל המידע יושב

באופן רציף בזיכרון).



מה סיבוכיות זמן הגישה לתא ברשימה מקושרת ?

כאשר ניגשים לרשימה מקושרת, מתחילים (תמיד) מהאיבר הראשון, ובאמצעות חץ עוברים לאיבר הבא, וחוזר חלילה. כך מתקדמים איבר איבר.

כלומר, על מנת להגיע לאיבר ה – 500,000 יש צורך בלולאה שתבצע 500,000 סיבובים.

מה סיבוכיות זמן הגישה לתא ברשימה מקושרת ?

משמעות : אם נרצה לגשת לתא ה – n/2 ברשימה מקושרת בת n תאים, נצטרך לבצע O(n/2) פעולות.

וכעת חישוב קטן :

$$O(n/2) = O(1/2 * n) = O(n)$$

כלומר – סיבוכיות הגישה לתא ברשימה מקושרת היא O(n).

שאלות

1. איך ניתן להגיע לתא האחרון ברשימה מקושרת ב – O(1) ?
2. אם רוצים שתהיה תמיד את היכולת להסיר את התא האחרון ברשימה מקושרת ב – O(1) מה חייבים לעשות?
3. אם רוצים מכל תא ברשימה, לדעת מיהו התא הקודם ו.. ב O(1), מה צריך לעשות?

תשובות בעמוד הבא

תשובות

שימו לב לכלל – הקצאת זיכרון עוזרת לשפר ביצועים

1. שומרים משתנה שתמיד יצביע על התא האחרון.
2. שומרים משתנה שתמיד יצביע על התא הלפני אחרון ברשימה, או לחילופין את היכולת להגיע מהתא האחרון לתא שקדם לו, ב – O(1).
3. הופכים את הרשימה לדו כיוונית (ישפיע כמובן על תהליך הבניה שלה. מכל תא יצאו 2 מצביעים (Pointerים) אחד קדימה ואחד אחורה.

דרגות נוספות של סיבוכיות

במידה והיינו צריכים לעבור על טבלה (מטריצה) עם כמות זהה של עמודות ושורות (נסמנן כ – n).

סריקת כל השורות והעמודות, הייתה דורשת לעבור על n שורות, ובכל שורה לסרוק n עמודות.

עלות סריקה שכזו – $O(n^2)$

אותה שאלה במרחב תלת ממדי (n שורות לרוחב, n עמודות, n יחידות עומק) : $O(n^3)$

כלל: איך מחשבים סיבוכיות בפונקציה מורכבת ?

פו' f() מבצעת 2 לולאות for אחת בתוך השנייה, כאשר כל אחת מבצעת n סיבובים. לאחר מכן היא מבצעת בדיוק 4 פעמים, לולאת for שמתבצעת n פעמים בכל פעם (סה"כ 4n סיבובי לולאה). בסופו של דבר, הפונקציה מבצעת 200 פעולות.

שאלה: מה סיבוכיות הפונקציה f()?

כלל: איך מחשבים סיבוכיות בפונקציה מורכבת ?

תשובה :

$4n < n^2$ $4n < 200$

$$O(n*n) + O(4n) + O(200) < O(n^2) + O(n^2) + O(n^2) =$$

$$3\ O(n^2) = O(3\ n^2) = O(n*n) = O(n^2)$$

מטרת סיבוכיות היא להעריך חסם עליון של כמות העבודה שהמעבד יצטרך לבצע, ומכאן השימוש באי שוויון.

תזכורת – $O(3n^2) = O(n^2)$ כי בסיבוכיות אין משמעות לכפל או חיבור/חיסור בקבוע (מספר).

כלל: איך מחשבים סיבוכיות בפונקציה מורכבת ?

החישוב בעמוד הקודם מסביר את התוצאה הסופית.

טכנית – סיבוכיות אלגוריתם תמיד תהיה הביטוי הגדול ביותר שמכיל בתוכו את n.

כלומר:

$$O(n^3+200n^2+300n+400) = O(n^3)$$

שיפור ביצועים באמצעות מידע ממוין

במידה והיינו מחזיקים במערך של מספרים (int) בשם array, והיינו מעוניינים למצוא האם המספר 7 קיים בו. במקרה הגרוע ביותר היינו צריכים לבצע סריקות כמספר התאים במערך (קיים בתא האחרון, או לא קיים בכלל).

כלומר, אם n הוא מיליון, אזי היינו מבצעים מיליון סיבובי לולאה בכדי לקבל תשובה.

מה צריך בכדי לשפר את התשובה ל – 19 סיבובי לולאה?

חיפוש בינארי - O(logn)

במדעי המחשב ניתן דגש משמעותי למידע ממוין, היות וסריקה שלו היא מהירה משמעותית.

במידה ויש צורך לבצע חיפוש במערך, במידה והוא ממוין, ניתן לחפש במערך בטכניקה הנקראת חיפוש בינארי.

בטכניקה זו, בכל סיבוב של הלולאה או שמקבלים תשובה (ואז סיימנו), או שכמות האיברים שנותרה לסרוק קטנה בחצי.

חיפוש בינארי - O(logn)

12	23	25	28	45	70	113
0	1	2	3	4	5	6

גבול שמאלי

האיבר האמצעי בבדיקה הראשונה

גבול ימני

במידה ונחפש האם המספר 6 נמצא במערך, ניגש תחילה למרכז המערך, לאיבר מספר 3.

היות ו – 3 קטן מ -6, ניתן להסיק שכל הערכים לפני תא 3 קטנים גם הם, ולכן ניתן לוותר לחלוטין על הסריקה שלהם, להזיז את הגבול השמאלי לתא מספר 4 ולחזור על הפעולה (חישוב האמצע מתבצע על ידי חיבור גבול שמאלי וימני וחלוקה ב – 2).

חיפוש בינארי - O(logn)

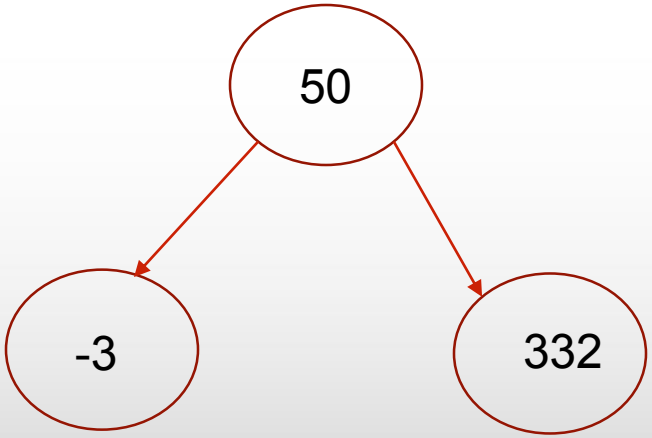
בצורה זו, בכל סיבוב של הלולאה אנחנו מורידים בחצי את כמות האיברים שנותר לסרוק. לכן, חישוב הסיבוכיות מיתרגם לשאלה: כמה פעמים אפשר לחלק n איברים ב – 2, עד שנקבל איבר בודד? התשובה : log(n) לפי בסיס 2.

בעבור n = 1,000,000 log(1,000,000) לפי בסיס 2, שקול לשאלה: מהי החזקה הקטנה ביותר של 2, שעוברת את מיליון.

התשובה: 19.

חיפוש בעץ בינארי

עץ בינארי הינו מבנה נתונים הדומה מאוד לרשימה מקושרת (שקף 17), אולם בניגוד לרשימה מקושרת שלה מצביע אחד (קדימה), לכל צומת בעץ בינארי יש 2 מצביעים: left ו – right.
כלל: עבור כל צומת, הבן השמאלי יהיה קטן ממנה ואילו הבן הימני גדול.



חיפוש בעץ בינארי

יש שיחשבו שחיפוש בעץ בינארי, מתבצע ב – O(logn), היות ובכל פעם שהחיפוש בוחר כיוון במורד העץ (ימין או שמאל), חצי מכמות הצמתים שבעץ יורדת, ולמעשה קיבלנו ווריאציה של חיפוש בינארי.

אולם, במידה ומייצרים עץ מרצף ערכים עולים או יורדים, מקבלים עץ שפונה רק ימינה או עץ שפונה רק שמאלה

חיפוש בעץ בינארי

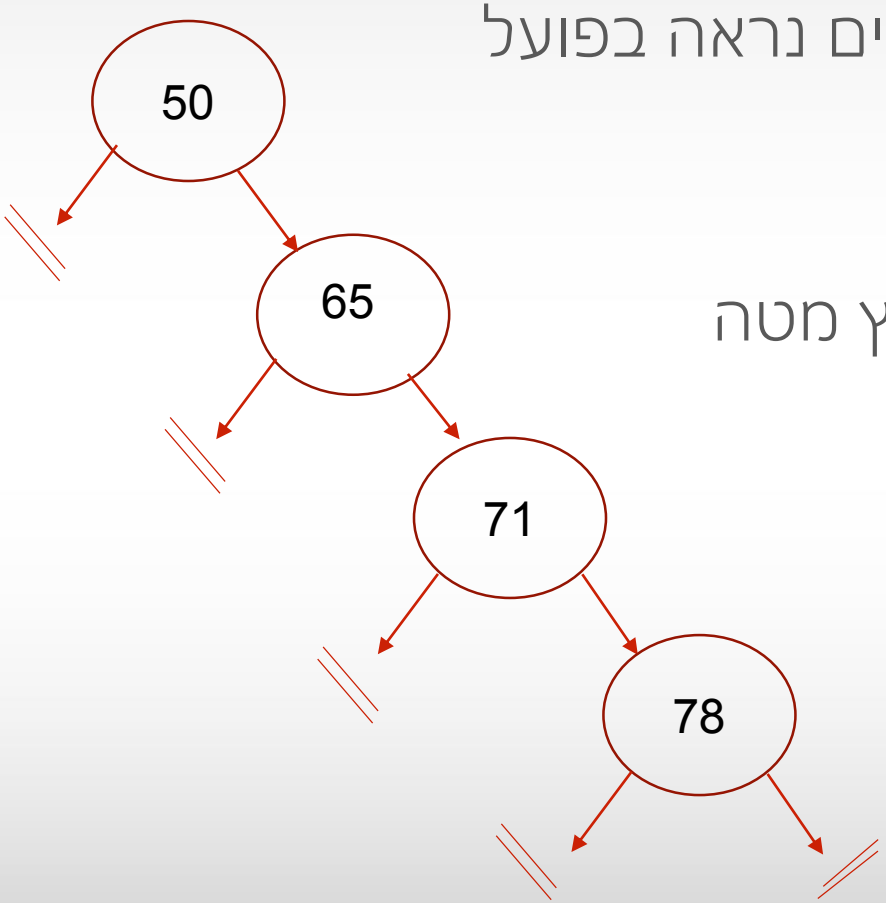
עץ שנוצר מרצף ערכים עולים נראה בפועל

כמו רשימה מקושרת.

המקרה הגרוע ביותר של עץ מטה

את הכף, ולכן חיפוש בעץ

בינארי מתבצע ב – O(n)



עץ אדום שחור

אחד הפתרונות לעץ בינארי שאינו מאוזן, הינו עץ אדום שחור.

כל צומת בעץ מקבלת תכונה חדשה, צבע (אדום או שחור).

בנוסף, מוגדרים בעץ מספר חוקים שחייבים להישמר.

בכל הכנסה, עדכון או מחיקה של איבר, התנאים נבדקים ואם יש צורך, צורת העץ משתנה על מנת לשמור על החוקים.

הערך המוסף בעץ אדום שחור, הוא שהאיזון בו מובטח, ולכן חיפוש בעץ אדום שחור מתבצע ב – O(logn).

סיבוכיות מיון

עלות מיון n איברים היא O(nlogn) (n כפול logn).

לא ניתן לבנות אלגוריתם מהיר יותר (הוכח מתמטית).

טיפ חשוב: בשאלות בהן תתבקשו לכתוב אלגוריתם / קוד הפועל בסיבוכיות של O(nlogn), לפני הכל – מיינו את האיברים.

סיבוכיות של בעיות בלתי פתירות

בעיית הסוכן הנוסע: נתון סוכן מכירות, שקיבל משימה לעבור ב – 25 ערים. הערים מחוברים ברשת רכבות, דרכים, מטוסים וכו' ביניהן. הסוכן מעוניין לעבור ב – 25 הערים תוך בזבוז זמן מינימלי בדרכים.

האלגוריתם פשוט, יש לעבור על כל המסלולים האפשריים ולהחזיר את זה עם סכום הדרכים המינימלי.

כמה מסלולים כאלו יש ? .. !25

סיבוכיות של בעיות בלתי פתירות

בעיית הסוכן הנוסע הינה בעיה ידועה שהוכח מתמטית שלא ניתן לפתור אותה בפחות מ – $O(n!)$.

סיבוכיות של $O(n!)$ כמו גם סיבוכיות נוספת של $O(2^h)$ (שתיים בחזקת n) נקראת סיבוכיות אקספוננציאלית.

סיבוכיות אקספוננציאלית לאלגוריתם, משמעה שהאלגוריתם הוא בלתי פתיר בפרק זמן סביר (הרבה שנות עבודה של מחשבי על).

סיכום עד כה

O(1) – ללא לולאות

O(logn) – חיפוש במידע ממוין

O(n) – מעבר על כל n האיברים

O(nlogn) – ביצוע מיון (אופטימלי)

$O(n^2)$ – לדוגמא: for בתוך for, כל אחת מתבצעת n סיבובים

$O(n^3)$ – לדוגמא: 3 לולאות for, מקוננות, n סיבובים כל אחת

...

$O(2^n)$ – לדוגמא: חישוב מספר פיבונאצ'י ברקורסיה

O(n!) – לדוגמא: חישוב בעיית הסוכן הנוסע

תרגילי בסיס

1. ממשו פונקציה המקבלת מספר, הפונקציה תדפיס את כל הספרות שלא קיימות במספר. n היא כמות הספרות של המספר, ממשו את שאלה 1 בסיבוכיות O(n).
2. כתבו פונקציה המחזירה את איבר פיבונצ'י ב – O(n) ,כאשר n הוא אינדקס איבר הפיבונצ'י המבוקש (לדוגמא האיבר החמישי).
3. תכננו באופן עצמאי מבנה נתונים שאמור לשבת בשרת גוגל, ולשרת את הפיצ'ר "השלמה אוטומטית של טקסט בזמן חיפוש".

כתבו מה היו הדגשים שלכם בתכנון מבנה הנתונים ומה הנחות היסוד שלכם בנוגע למהירות ולבזבוז זיכרון, במידה והיו לכם.

תרגילי בסיס

4. כתבו פונקציה המקבלת מערך ממוין ומספר num. הפונקציה תחזיר כמה פעמים num מופיע במערך. יש לכתוב את הפונקציה היעילה ביותר שניתן מבחינת סיבוכיות (חלק מהאתגר – זיהוי הסיבוכיות האופטימלית).

5. קראו אינטרנטית וסכמו בקצרה מהו גרף.

6. סכמו **בקצרה** אילו 2 מימושים שונים יש לגרף, ומיהו המימוש העדיף מבחינת סיבוכיות.

תרגילי בסיס

7. סכמו **בקצרה** על אלגוריתם דייקסטרה (מה תפקידו, סיבוכיות ולא בהכרח כיצד הוא עובד) והסבירו כיצד אפשר לשלב אותו במשחק מחשב.

8. כתבו אלגוריתם המקבל 2 מערכים בני n איברים. ידוע שערכי האיברים במערכים נעים בין 0 ל – 100,000, אך הם אינם ממוינים. הנחת יסוד, בכל מערך כל המספרים שונים. הסבירו אלגוריתמית, כיצד ניתן לזהות ב – O(n) אילו מספרים משותפים ל – 2 המערכים,. אין הגבלה מבחינת הקצאת זיכרון, אסור להשתמש ב – Set.

טיפים לפתרון התרגילים

1. מערך בוליאני בן 10 תאים, כל תא ייצג ספרה 0-9.
2. תחזוקת 2 משתנים, שזוכרים 2 איברים אחורה. לולאה, ובנייה בתוך הלולאה, של מספרי פיבונאצ'י עד שמגיעים למספר הרצוי.
3. עץ שכל איבר בצומת שלו, מייצג תו בודד. העץ לא יהיה בינארי וכל צומת תחזיק מערך של 27 תווים (a-z ורווח). יש לחשוב איך מוחקים משפט מהעץ מבלי להרוס משפטים אחרים.
4. חיפוש בינארי, ולאחר זיהוי האיבר, 2 קריאות לפונקציות שונות "כמו חיפוש בינארי" למציאת גבול שמאלי וימני. רק אם דבקים בהתנהגות של מעין חיפוש בינארי מקבלים O(logn), כל חיפוש לינארי (איבר איבר) ייצר O(n).

טיפים לפתרון התרגילים

5. שווה לחשוב תוך כדי, איך Waze מחזיקים את המידע לגבי מערכת
6. מטריצה, רשימת סמיכויות
7. דייקסטרה הוא אלגוריתם מסלול קצר ביותר מנקודה לנקודה על גרף

טיפים לפתרון התרגילים

8. היות וערכי המספרים נעים בין 0 ל – 100,000, ניתן לעשות מערך של intים בן 100,000 תאים, כאשר כל תא מייצג את כמות ההופעות של המספר אותו הוא מייצג, ופשוט לעבור על 2 המערכים תוך קידום התא המתאים עבור כל מספר. מעבר אחד על כל מערך ב – O(n), מעבר אחד על 100,000 תאים ב – O(100,000)=O(1) נותן תשובה ב – O(n).

סיבוכיות ב – Collections

Objects

Array

מה זה hash ?

פונקציית hash היא פונקציה מתמטית, שניתן להשתמש בערכים שהיא מייצרת בשלל דרכים.

אחת הדרכים היא מיפוי יעיל של איברים בתוך אובייקט, שימוש ב key value pair

בהכנסה לאובייקט, פונקציית ה – hash מחשבת על סמך המפתח, לאיזה מספר תא יש להכניס את ה – value.

פרקטית - מה זה hash

מבחינת הגישה בתעשייה (עולם ההייטק): פו' hash מייצרת O(1) בגישה לנתונים (זה לא מדויק ב – 100%, אבל זה ה"סלנג").

סיבוכיות ב – Javascript Object

אובייקט ב Js זהו מבנה נתונים המורכב מ key value pair

כאשר כל key מצביע בדיוק על value אחד.

אובייקט יכול להכיל את אותם ערכים ב keys שונים, אך אינו יכול להכיל key יותר מפעם אחת באובייקט.

עצם השימוש באובייקט, משמעו O(1) בשליפה של כל value מתוך האובייקט

כנל לגבי הכנסה \ מחיקה של מידע לאובייקט.

סיבוכיות ב - Array

Array ב JavaScript הוא גם אובייקט בפני עצמו, כלומר ה prototype שלו (בסופו של דבר) מצביע על אובייקט.

כדי למצוא איבר במערך, צריך לבצע ריצה על כל המערך (לכל היותר) לבצע השוואה מסוימת ולמצוא את האיבר הנדרש, לכן סיבוכיות הריצה לחיפוש במערך היא O(n) .

במידה ואנחנו רוצים למחוק או לעדכן איבר, בהנחה שאין לנו את המיקום שלו שמור או ממופה מאובייקט \ DOM נאלץ לבצע ריצה כדי למצוא את האיבר הנדרש ורק לאחר מכן לבצע את הפעולה (מחיקה, עריכה) ולכן גם כאן הסיבוכיות ריצה שלנו היא O(n)

תרגילים

1. ממשו אובייקט Queue שמחזיק כמות דינאמית של מספרים, באמצעות רשימה מקושרת.
   על האובייקט לממש את הפעולות:
   1. הכנסה לתור O(1)
   2. הוצאת מהתור O(1)
   3. האם התור ריק O(1)
   4. שליפת גודל התור O(1)

טיפ: משתנים מחלקתיים (=השקעת זכרון) משפרים ביצועים.

שאלה חשובה: האם הוספת איבר צריכה להתבצע בסוף הרשימה או בתחילתה ?

תרגילים

2. כתבו פונקציה המקבלת מערך של מספרים. הפונקציה תבדוק האם המערך מאוזן.

מערך מאוזן הוא מערך שניתן לסכום איברים שלו מתחילת המערך ועד לתא X, כך שהסכום שהתקבל זהה לסכום האיברים החל מהתא ה X+1 ועד לסוף המערך.

על הפונקציה להתבצע ב – O(n)

תרגילים

3. בכל פעם שמשתמש מבצע פעולה באתר, יש לוודא שהוא ביצע login. פעולת ה – login היא פעולה איטית שמתרחשת מול ה – DB.

הציעו תהליך הכולל מבנה נתונים שיתוחזק בשרת, שלאחר login מוצלח ראשוני, ווידוא זהותו עבור הקליקים הבאים לא יכלול דיבור מול ה – DB ויוכל להתבצע במהירות.

צאו מנקודת הנחה ששרת יכול לשתול מידע אצל הלקוח בזמן ה – login, ולדאוג שהמידע ישלח אליו בכל בקשה.

תרגילים

4. יש צורך בתכנון אובייקט שיתנהג כמו "מערך מיוחד". ניתן יהיה לשמור באובייקט מספרים, כפי ששומרים במערך, לשלוף ולמחוק. אובייקט המערך המיוחד ידע לשמור n איברים. המספר n נקבע בזמן עליית התוכנה, והוא לא ישתנה לאחר מכן. על מנת לתת ביצועי מערך מבחינת סיבוכיות, האובייקט יכיל משתנה מחלקתי מסוג מערך, אולם איברי המערך המדויקים עדיין לא ידועים.

תרגילים – המשך תרגיל 4

איברי המערך צריכים לשמור מספרים מבחינה פונקציונלית, אולם הם יכולים להיות מורכבים יותר – לשיקולכם.

תנאי: לאחר יצירת המערך בפעם הראשונה, אסור להקצות יותר זיכרון למערך נוסף.

תרגילים – המשך תרגיל 4

איברי המערך צריכים לשמור מספרים מבחינה פונקציונלית, אולם הם יכולים להיות מורכבים יותר – לשיקולכם.

תנאי: לאחר יצירת המערך בפעם הראשונה, אסור להקצות יותר זיכרון למערך נוסף.

מותר לכם לתכנן את איברי המערך כרצונכם (מותר לכם לבזבז שם זיכרון), אבל אחרי שנוצר המערך הראשוני, אסור להקצות עוד זיכרון למערך נוסף.

תרגילים - המשך תרגיל 4

על האובייקט לספק את הפעולות הבאות:

add(int num, int index) - O(1)

getNum(int index) – O(1) אם אין – null

delete(int index) – O(1)

deleteAll() – O(1)

האתגר מתמקד ב - איך ממשים deleteAll ב – O(1) בלי הקצאות זיכרון נוספות בזמן ריצה.

תרגילים

5. השתמשו בנתונים החוזרים מ הAPI הבא: מערך של מדינות
https://restcountries.eu/rest/v2/all

נסו להמיר את המבנה נתונים כך שהתקבל מזהה מדינה :
שם \ alpha3code נוכל ל/עדכן \ למחוק את המדינה ב O(1)

רמז: יש להמיר את המערך ל...(מבנה נתונים יעיל יותר)

טיפים לפתרון התרגילים

1. תחזוק משתנה מחלקתי שמבטא את גודל התור, יהפוך את getSize() ואתה isEmpty() לפו' פשוטות ביותר. תיחזוק המשתנה לא ישנה את הסיבוכיות של שאר הפו', והוא פשוט ביותר.

יש לתחזק באובייקט משתנה מחלקתי נוסף מסוג Node, מצביע לתא הראשון ברשימה המקושרת, ובנוסף מצביע לתא האחרון.

בכל פעם שיתווסף איבר, יש להוסיף בסוף. בכל פעם שיצא איבר, יש להוציא את האיבר הראשון ברשימה.

טיפים לפתרון התרגילים

2. לולאת for ראשונה תסכום את איברי המערך לתוך משתנה שנסמנו – arraySum.

לולאת for שניה, תסכום בשנית את איברי המערך.

אם במהלך הבדיקה, הסכום הגיע למחצית מ arraySum משמע – המערך מאוזן. אם הסכום הנוכחי עבר את arraySum/2 אזי המערך לא מאוזן.

טיפים לפתרון התרגילים

3. יצירת מנגנון Cache, דמוי מפה בשרת. בזמן login, השרת יוצר עבור המשתמש token, טקסט מוצפן על ידי פונקציית hash. הטקסט יישתל אצל המשתמש וישלח בכל פקודה לשרת. בשרת תתוחזק מפה שתמפה את ה – token לאובייקט שיווצר בפעם הראשונה בעת ה login, ויכיל את כל המידע הדרוש לטובת בדיקת הרשאות למשתמש.
עצם העובדה שהמידע מוחזק בזיכרון, חוסכת עבודה מול ה-DB.

טיפים לפתרון התרגילים

תשובה מצוינת תדע להתייחס לעובדה שישנם מספר שרתים, שכולם צריכים לחלוק את המידע הנ"ל, ולכן המפה צריכה להסתנכרן בין כל השרתים, על כל עדכון שמתבצע בה.

טיפים לפתרון התרגילים

4. אם נחזיק עבור כל אובייקט במערך Integer ולצידו – timestamp, ובנוסף משתנה מחלקתי נוסף שזוכר מתי התרחש ה deleteAll האחרון (עוד timestamp), הפתרון הופך להיות פשוט מאוד.

תודה רבה על ההקשבה!

JOHN BRYCE
פתרונות למידה והדרכה
a matrix company