**סיבוכיות – Complexity**

זהו תחום אקדמי בעולם מדעי המחשב, שיש לו הרבה ביטוי במציאות, כלומר הוא לא תאורטי לגמרי, והוא מטפל בנושא של יעילות וביצועים.

הרעיון הוא להיות מסוגלים להסתכל על קטע קוד (למשל, פונצקיה), ולהסיק מה מידת הסיבוכיות שלו.

מכיוון שאין שום שני קטעי קוד זהים, וכשאנו בודקים סיבוכיות, אנו בדכ משווים לקוד אחר, הסיבוכיות לא מדוייקת לעולם, ותמיד תימדד על ידינו בסדרי גודל.

הפרמטר העיקרי של חישוב הסיבוכיות, נעוץ בלולאות.

מדוע? מכיוון שלולאה היא "בור ללא תחתית". יכולה להיות לה אירטציה אחת, ויכולות להיות לה מיליארד איטרציות. אבל, כל זמן שאין אני יודעים מראש מה מספר המדוייק של האיטרציות, אנו נניח את הגרוע ביותר.

for (let I = 0; I < n; i++) {

// do something

}

סיבוכיות הנגזרת מלולאה כמו זו, תלויה לגמרי ב-n. ומכיוון שאין לנו את n, נניח תמיד שהוא גדול.

**Big O Notation**

יחידת המידה לסיבוכיות היא O, הנגזרת מהמילה הגרמנית Ordnung שמציינת Order. באנגלית סדר גודל זה Order of Magnitude.

מה מציינים באמצעות O?

סדר גודל עבור אלגוריתם מסויים.

O(1)

זהו סדר גודל של קטע קוד שבו אין אף לולאה.

O(n)

אלגוריתם שסדר הגדול שלו הוא זה, תהיה בו לולאה אחת בלבד

(n - מספר הריצות של הלולאה. לכן O(n) סדר גודל של ריצה של לולאה אחת)

O(n^2)

לולאה מקוננת (לולאה בתוך לולאה)

O(log n)

הנה מערך בן 8 תאים (**ממויין** מימין לשמאל)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 20 | 32 | 41 | 57 | 89 | 93 | 94 |

|  |  |  |  |
| --- | --- | --- | --- |
| 57 | 89 | 93 | 94 |

|  |  |
| --- | --- |
| 93 | 94 |

|  |
| --- |
| 94 |

2^3=8

Log – באיזה מספר אני צריך להעלות מספר נתון בחזקה, כדי לקבל תוצאה נתונה.

למשל, באיזה חזקה צריך להעלות את 10 כדי לקבל 1000?

התשובה: בחזקת 3. 10 \* 10 \* 10 = 1000

Log(10)1000 = 3

ומכאן השם O(logn)

...

**אריתמטיקה של סיבוכיות**

האם הרצה של קוד O(n^2) עדיף על הרצה של קוד O(n) + O(n)?

N=1

O(n^2) = 1

O(n) + O(n) = O(1) + O(1) = 2

N = 10

O(n^2) = 100

O(n) + O(n) = 20

N= 100

O(n^2) = 10,000

O(n) + O(n) = 200

התשובה היא לא. נעדיף 2 \* O(n) על פני O(n^2)

כללי האריתמטיקה:

פעולות חיבור של סדרי גודל. התוצאה תהיה סדר הגודל הגדול ביותר.

O(1) + O(1) + O(1) = 3 \* O(1) = O(1)

O(n^2) + O(n) = O(n^2)

N = 1000

// O(n^2)

    for (let i=0; i < arr.length; i++ ) {

        for (let z = 0; z < arr[i].length; z++) {

            sum += arr[i][z];

        }

    }

    // O(n)

    for (let i = 0; i < arr.length; i++) {

        sum += arr[i];

    }

O(n^2) = 1000 \* 1000 = 1M

O(n) = 1000

1000000 + 1000 ~= 1000000

ה-1000 נעשה **זניח**

**Set**

זהו סוג של מבנה נתונים המכיל ריבוי ערכים (כמו מערך).

מה שמאפיין את Set הם שני תנאים:

1. לא ניתן להכניס שני פרטים זהים
2. גישה לפריט ב-Set היא פעולה בסדר גודל של O(1)

היופי של פונצקיות hash הוא שתמיד עבור אותו אינפוט הן יחזירו את אותו hash.

**Map**

זהו סוג של מבנה נתונים שמאד מזכיר את ה-Set בהבדל אחד. ה-Set קובע לעצמו את ההאש (hash) על ידי הפעלת פונקציה קריפטוגרפית על האובייקט שנכנס ל-Set. ה-Map לעומתו, דורש מאיתנו לספק לו את המפתח. כלומר את ה-ID של הרשומה. וה-Map ידאג כבר לעשות לו האשינג בעצמו.

**רקורסיה**

פונקציה שבמהלך הביצוע שלה, קוראת לעצמה:

Function doSomething(x) {

Let a = 10;

doSomething(n)

a = a + 5;

}

יש נוסחה די קבועה לכתיבת רקורסיה שמכילה את אותם 4 שלבים:

1. חתימת הפונקציה, דהיינו השם שלה ורשימת הפרמטרים, זהה לפונקציה שאינה רקורסיבית. הקוד שקורא לפונקציה ראשון, דהיינו הקריאה מחוץ לרקורסיה, לא יודעת שהפונקציה היא רקורסיבית. **המשמעות היא שצרכן הפונקציה יקרא לה בצורה זהה הן אם היא רקורסיבית והן אם לא.**
2. תנאי יציאה – יש לכלול תנאי שקובע מתי עלינו לבצע return ולא להמשיך בביצוע הפונקציה.
3. יש להבין מהי כלל המטלה לביצוע ומהו שלב אחד במטלה. ואז לבצע רק את השלב האחד.
4. קריאה רקורסיבית, המכילה את שאר המטלה (דהיינו, כלל המטלה פחות אותו צעד שכבר ביצענו)

אם לבעיה מסויימת יש פתרון רקורסיבי וגם פתרון איטרטיבי (מלשון iterate, כלומר לרוץ על לולאה) תמיד נעדיף את הפתרון האיטרטיבי מפני שהוא תמיד יותר זול מבחינת משאבים מהפתרון הרקורסיבי.

ישנן מספר בעיות, למשל ניווט בעץ קבצים, שלא אפשריות ללא רקורסיה. רק אז נבחר ברקורסיה כדי ליישם את הפתרון (כלומר אם אין שום דרך אחרת)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \* | \* | \* | \* | \* | \* | \* | \* | \* | \* |

printStars(10);

function printStars (number) {

    // 2. exit condition

    if (number === 0) return;

    // 3. do the single thing

    // \* \* \* \* \* \* \* \* \* \*

    document.write('\* ')

    // 4. call the recursion with the rest of the task

    printStars(number - 1);

}

