סיבוכיות

מבוא למדעי המחשב - 67101

מרצה: אריה שלזינגר

מתרגלים: יפעת חדד ואורי מאיר

סוכם ע"י: שריה אנסבכר

סמסטר ב' תשפ"ג, האונ' העברית

תוכן העניינים

אשמח לקבל הערות והארות על הסיכומים על מנת לשפרם בעתיד,
כל הערה ולו הפעוטה ביותר (אפילו פסיק שאינו במקום או רווח מיותר) תתקבל בברכה;
אתם מוזמנים לכתוב לי לתיבת הדוא"ל: sraya.ansbacher@mail.huji.ac.il.

לסיכומים נוספים היכנסו לאתר:
אקסיומת השלמות - סיכומי הרצאות במתמטיקה
https://srayaa.wixsite.com/math

1 הקדמה

אלגוריתם הוא עצם מופשט, סדרה של צעדים שעוצבה על מנת לבצע משימה מסוימת; תכנות הוא מימוש של אלגוריתם בשפה מסוימת ובדרך מסוימת. בבואנו לבדוק אם אלגוריתם מסוים הוא אלגוריתם טוב אנו מתעלמים משפות התכנות והמחשבים במסוימים שבאמצעותם ניתן להפעיל את האלגוריתם ומבוננים אך ורק באלגוריתם עצמו שכפי שציינו לעיל הוא עצם מופשט. אלגוריתמים נמדדים בשני דברים:

1. נכונות: האם האלגוריתם מבצע את המשימה נכון? האם הוא נותן פתרון נכון לבעיה?
2. יעילות: האם האלגוריתם מבצע את משימתו מהר מספיק? האם הוא צורך הרבה זיכרון?

♣ בקורס הזה נתמקד ביעילות של אלגוריתמים מבחינת הזמן שלוקח להם לבצע את המשימה ולא נעסוק בזיכרון או בנכונות.

♣ הסיבה לכך שלא נעסוק בנכונות היא שכל האלגוריתמים שנפגוש בהם יתנו תשובה נכונה תמיד (אם לא אז מבחינתנו הם אינם מבצעים את המשימה), לא נראה לי שנלמד על זה בקורס, אולם פעמים שאנו מוותרים על הדרישה לנכונות מוחלטת ומסתפקים בהסתברות גבוהה לפתרון נכון וזאת ע"מ לייעל את האלגוריתם, הדוגמה הקלאסית לכך היא אלגוריתמים הסתברותיים לבדיקת ראשוניות של מספר נתון.

דרישת הנכונות פשוטה מאד אבל כיצד ניתן לקבוע אם האלגוריתם פותר את הבעיה מהר מספיק? הרי הדבר תלוי בחומרה של המחשב עליו הוא רץ, בתוכנה, בשפת התכנות ועוד הרבה משתנים אחרים (כולל הטמפרטורה בחדר![1])...

[1] מחשבים עובדים מהר יותר בקור.

2 מדידת יעילות של אלגוריתם

התשובה היא שאנחנו מעוניינים למדוד את האלגוריתם שלנו ביחס לאלגוריתמים אחרים[2]ולכן, בגלל שאלגוריתמים הם עצמים מופשטים, כל המשתנים הללו אינם מעניינים אותנו.
טוב, אבל עכשיו בכלל אין לנו מושג כיצד למדוד את יעילות האלגוריתם, לא נשאר לנו שום דבר לעבוד איתו...
אז זהו, שלא; כאן בא לעזרתנו רעיון יפה: נמדוד אלגוריתמים ע"פ מספר הצעדים הבסיסיים שלוקח להם לבצע את המשימה המוטלת עליהם כתלות בגודל הקלט.

צעדים בסיסיים:

- פעולות אריתמטיות
- השוואת שני אובייקטים
- השמה במשתנה (c_3)
- גישה לאיבר ב-container
- קריאה לפונקציה או החזרה מפונקציה
- ועוד...

הנחות יסוד:

- כל צעד בסיסי לוקח זמן קבוע עבור קלט נתון.
- הזמן יכול להשתנות מצעד מסוג אחד לצעד מסוג אחר.
- הזמן תלוי גם במחשב, בחומרה, בתוכנה, בשפת התכנות ואפילו בטמפרטורה בחדר אך כפי שציינו לעיל אלו אינם מעניינים אותנו.

לכן, מבלי להיכנס לשאלה כמה זמן בדיוק כל פעולה לוקחת אנחנו נסמן אותן בקבועים מהצורה c_i (וכפי שנראה בהמשך זה לא ממש מעניין אותנו כמה הם שווים):

- פעולות אריתמטיות (c_1)
- השוואה בין מספרים (c_2)
- השמה במשתנה (c_3)
- גישה לאיבר ב-container (c_4)
- קריאה לפונקציה או החזרה מפונקציה (c_5)
- ועוד...

[2] הסיבה לכך שאנחנו מודדים את האלגוריתם ביחס לאחרים היא שאת כולם ניתן להריץ על אותם מחשבים עם אותן החומרות ואותן התוכנות, לכן מה שמעניין אותנו הוא הוא באיזה אלגוריתם עלינו להשתמש והתשובה לא תהיה תלויה בכל המשתנים הללו: אם אלגוריתם אחד יהיה עדיף על חברו על מחשב מסוים הוא יהיה עדיף על כל מחשב אחר (עד כמה כל זה יהיה נכון כשיהיה לנו מחשוב קוונטי???).

3 דוגמאות: אלגוריתמים שונים למציאת איבר ברשימה

בפרק זה נביא כמה דוגמאות שימחישו לנו מה בדיוק אנחנו מחפשים כשאנחנו רוצים למדוד יעילות של אלגוריתם, כל הדוגמאות תהיינה אלגוריתמים שמטרתם למצוא איבר (obj) ברשימה: אם הוא קיים הם יחזירו את אחד האינדקסים שבהם הוא מופיע ואם אינו מופיע ברשימה יחזירו False.

אלגוריתם 1 חיפוש ליניארי פשוט

הקלט הוא רשימה (lst) והאובייקט שאנחנו מחפשים ברשימה (obj).

1. אתחל $i := 0$ ו-$ans :=$ False.
2. כל עוד i קטן ממש מהאורך של lst:
   - אם obj שווה לאיבר ה-i ב-lst:
     הגדר את $ans := i$.

   בכל מקרה הגדר $i := i + 1$.
3. החזר את ans.

כאן גודל הקלט הוא אורך הרשימה ולכן אם נסמן אותו ב-n אז ה"זמן" (יסומן ב-t) שיקח לאלגוריתם לבצע את המשימה מקיים:

$$2c_3 + n \cdot (c_2 + c_3) + c_5 \le t \le 2c_3 + n \cdot (c_2 + 2c_3) + c_5$$

כלומר הזמן כתלות בקלט מתנהג כמו פונקציה ליניארית.

אלגוריתם 2 חיפוש ליניארי

1. אתחל $i := 0$.
2. כל עוד i קטן ממש מהאורך של lst:
   - אם obj שווה לאיבר ה-i ב-lst:
     החזר את i.
   - אחרת:
     הגדר $i := i + 1$.
3. החזר False.

כעת נקבל שהזמן מקיים:

$$c_3 + 2c_2 + c_5 \le t \le c_3 + n \cdot (c_2 + c_3) + c_5$$

ומכיוון שמה שמעניין אותנו הוא המקרה הגרוע ביותר נאמר שהזמן כתלות בגודל הקלט מתנהג כפונקציה ליניארית.

כעת נעסוק במקרה שבו הרשימה כבר ממוינת לפי יחס סדר מלא כלשהו[3].

אלגוריתם 3 חיפוש "נאיבי" ברשימה ממוינת

1. לכל i בין 0 (כולל) לאורך הרשימה (לא כולל):
   - אם obj שווה לאיבר ה-i ב-lst:
     החזר את i.
   - אם obj קטן מהאיבר ה-i ב-lst:
     החזר False.
2. החזר False.

כעת נקבל שהזמן מקיים[4]:

$$c_2 + c_3 + c_5 \leq t \leq n \cdot (2c_2 + c_3) + c_5$$

ושוב, מכיוון שמה שמעניין אותנו הוא המקרה הגרוע ביותר נאמר שהזמן כתלות בגודל הקלט מתנהג כפונקציה ליניארית.

אלגוריתם 4 חיפוש בינארי ברשימה ממוינת

1. אתחל $left := 0$ ו-$right := len(lst) - 1$.
2. כל עוד $left \leq right$:
   - הגדר $mid := \left\lfloor \frac{left+right}{2} \right\rfloor$.
   - אם obj שווה לאיבר ה-mid ב-lst:
     החזר את mid.
   - אחרת:
     - אם obj קטן מהאיבר ה-mid:
       הגדר $right := mid - 1$
     - אחרת:
       הגדר $left := mid + 1$
3. החזר False.

נשים לב שכאן גודל הרשימה קטן בחצי בכל איטרציה ולכן הזמן מקיים:

$$2c_3 + c_2 + c_3 + c_2 + c_5 \leq t \leq 2c_3 + \log_2 (n \cdot [c_2 + c_3 + 2c_2 + c_3]) + c_5 = 2c_3 + c_5 + \log_2 (c_2 + c_3 + 2c_2 + c_3) + \log_2 (n)$$

כלומר הזמן מתנהג כמו לוגריתם!

♣ רק כדי לסבר את האוזן עד כמה זה יעיל: נניח שלעבור על רשימה בגודל של $256 = 2^8$ איברים לוקח לאלגוריתם דקה אז כדי לעבור על רשימה באורך של $65,536 = 256^2 = 2^{16}$ יקח לאלגוריתם רק 2 דקות...

[3] לעניינו ניתן לחשוב שכל האיברים ברשימה הם מספרים והם מסודרים מהקטן לגדול.
[4] בלולאת for מתבצעת השמה של i בתחילת כל איטרציה.

4 מחלקות סיבוכיות

4.1 קצב גידול של פונקציות

את הנושא הזה כבר למדנו בקורס "מתמטיקה בדידה" אך בכל זאת נחזור כאן על כמה ההגדרות וטענות.

נכון לעכשיו הנושא הזה לא מופיע בסיכומים שלי ממתמטיקה בדידה.

4.1.1 הגדרות

תהיינה $f,g:\mathbb{N}\to\mathbb{R}^{+}$[5] פונקציות[6].

הגדרה 4.1. נאמר ש-g היא <u>O גדול של f</u> ונסמן $g\in O\left(f\right)$[7] אם קיימים $0<c\in\mathbb{R}$ ו-$N\in\mathbb{N}$ כך שלכל $N<n\in\mathbb{N}$ מתקיים $f\left(n\right)\leq c\cdot g\left(n\right)$.

הגדרה 4.2. נאמר ש-g היא <u>Ω של f</u> ונסמן $g\in\Omega\left(f\right)$ אם קיימים $0<c\in\mathbb{R}$ ו-$N\in\mathbb{N}$ כך שלכל $N<n\in\mathbb{N}$ מתקיים $c\cdot g\left(n\right)\leq f\left(n\right)$.

הגדרה 4.3. נאמר ש-g היא <u>Θ של f</u> ונסמן $g\in\Theta\left(f\right)$ אם קיימים $0<c_1,c_2\in\mathbb{R}$ ו-$N\in\mathbb{N}$ כך שלכל $N<n\in\mathbb{N}$ מתקיים $c_1\cdot g\left(n\right)\leq f\left(n\right)\leq c_2\cdot g\left(n\right)$.

♣ סימון נפוץ מאד לכך ש-g היא O גדול של f הוא $g=O\left(f\right)$ אבל הוא בעייתי מפני שייתכן שגם $h\in O\left(f\right)$ ולמרות זאת $g\neq h$ בסתירה לטרנזיטיביות השוויון, כלומר גם מי שמשתמש בסימון השוויון אינו מתכוון באמת לכך שמה שכתוב באגף ימין הוא בדיוק מה שכתוב באגף שמאל.

4.1.2 טענות

טענה 4.4. תהיינה $f,g,h:\mathbb{N}\to\mathbb{R}^{+}$, מתקיימים כל הפסוקים הבאים:

- $g\in\Theta\left(f\right)\Longleftrightarrow\Theta\left(f\right)=\Theta\left(g\right)$.
- אם קיימים $0<c\in\mathbb{R}$ ו/או $b\in\mathbb{R}$ כך ש-$f=c\cdot g+b$ אז $\Theta\left(f\right)=\Theta\left(g\right)$.
- $\Theta\left(f\right)=O\left(f\right)\cap\Omega\left(f\right)$.
- $f\in\Theta\left(f\right)$ וממילא גם $f\in O\left(f\right)$ ו-$f\in\Omega\left(f\right)$.
- $g\in O\left(f\right)\Longleftrightarrow f\in\Omega\left(g\right)$, לכן נוכל בטענות הבאות רק על O גדול וממילא הטענות תהיינה נכונות גם עבור Ω.
- $O\left(f\right)\subseteq O\left(g\right)\Longleftrightarrow g\in O\left(f\right)$.
- $g\in O\left(f\right)\land f\in O\left(h\right)\Rightarrow h\in O\left(h\right)$.

משפט 4.5. יהי $P\in\mathbb{R}\left[x\right]$ פולינום מדרגה $k\in\mathbb{N}_0$ כך שהמקדם של החזקה ה-k חיובי, מתקיים $P\in\Theta\left(n^{k}\right)$.

♣ באופן כללי כאשר מחברים פונקציות הולכים לפי זו ששואפת לאינסוף מהר יותר.

טענה 4.6. לכל $k,m\in\mathbb{N}_0$ כך ש-$k<m$ מתקיים $O\left(n^{k}\right)\subseteq O\left(n^{m}\right)$ ובפרט $n^{k}\in O\left(n^{m}\right)$.

משפט 4.7. מתקיים $O\left(1\right)\subseteq O\left(\log_{b}n\right)\subseteq O\left(\sqrt{n}\right)\subseteq O\left(n\right)\subseteq O\left(n\log_{b}n\right)\subseteq O\left(n^{k}\right)\subseteq O\left(a^{n}\right)\subseteq O\left(n!\right)$ (לכל $1<a,b,k\in\mathbb{N}$).

♣ לעומת זמן ריצה לוגריתמי[8], כל זמני הריצה הפולינומיים שדרגתם שונה וכל זמני הריצה האקספוננציאליים שבסיסם שונה - שונים זה מזה.

[5] נזכיר: $\mathbb{R}^{+}:=\left(0,\infty\right)$, למעשה אין צורך בכך שהטווח יהיה $\mathbb{R}^{+}$ אלא עבור $n\in\mathbb{N}$ גדול דיו מתקיים $0<f\left(n\right),g\left(n\right)$.

[6] למעשה מדובר בסדרות ואני לא רואה שום סיבה שההגדרות לא תהיינה תקפות גם לגבי פונקציות ממשיות רגילות ($f:\mathbb{R}^{+}\to\mathbb{R}^{+}$).

[7] בכך אנחנו למעשה מגדירים את הקבוצה $O\left(f\right)$.

[8] לכל $1<a,b\in\mathbb{R}$ מתקיים לכל $x\in\mathbb{R}$ השוויון $\log_{a}\left(x\right)=\log_{a}\left(b\right)\cdot\log_{b}\left(x\right)$ ולכן מדובר רק בכפל בקבוע שכמובן אינו משנה את מחלקת הסיבוכיות.

4.2 המחלקות

בגלל המשפט האחרון מחלקות הסיבוכיות המעניינות אותנו הן המחלקות שלהלן (ע"פ הסדר הרצוי).

סיבוכיות טובה:

- $O\left(1\right)$ - זמן ריצה קבוע שאינו תלוי בקלט
- $O\left(\log n\right)$ - זמן ריצה לוגריתמי
- $O\left(\sqrt{n}\right)$ - זמן ריצה "שורשי"

סיבוכיות סבירה:

- $O\left(n\right)$ - זמן ריצה ליניארי
- $O\left(n\log n\right)$ זמן ריצה ליניארי-לוגריתמי

סיבוכיות גרועה:

- $O\left(n^{k}\right)$ - זמן ריצה פולינומי

סיבוכיות נוראית:

- $O\left(a^{n}\right)$ - זמן ריצה מעריכי
- $O\left(n!\right)$ - זמן ריצה "עצרתי"

♣ כשהקלט הוא יותר ממשתנה אחד אז הסיבוכיות עובדת ע"פ פונקציה בשני משתנים (כגון $O\left(n\cdot m\right)$).

5 סיבוכיות של פעולות מובנות בפייתון

נסמן ב-n את האורך של ה-$container$ המדובר.

פעולה	דוגמה	סיבוכיות
גישה לאינדקס	$lst[i]$	$O(1)$
השמה (הערה 1)	$lst[i] \; = \; object$	$O(1)$
len (הערה 2)	$len(lst)$	$O(1)$
הוספת איבר בסוף הרשימה (הערה 3)	$lst.appen(object)$	$O(1)$
pop (הערה 4)	$lst.pop(i)$	$O(n)$
pop (הערה 5)	$lst.pop()$	$O(1)$
ניקוי הרשימה	$lst.clear()$	$O(1)$
slice	$lst[a:b:c]$	$O\left(\frac{b-a}{c}\right)$
extend	$lst.extend(container)$	$O(len(container))$
יצירת רשימה	$list(container)$	$O(len(container))$
השוואת רשימות	$lst1 == lst2, \; lst1! = lst2$	$O(n)$
השמה ע"י slicing (הערה 6)	$lst[a:b:c] \; = \; container$	$O(n)$
הוספת איבר באינדקס מסוים (הערה 4)	$lst.insert(i, object)$	$O(n)$
מחיקת איבר (הערה 4)	$del \; lst[i]$	$O(n)$
סילוק איבר מרשימה	$lst.remove(object)$	$O(n)$
בדיקה אם איבר נמצא ברשימה	$x \; in \; lst, \; x \; not \; in \; lst$	$O(n)$
העתקת רשימה	$lst.copy()$	$O(n)$
מציאת איבר מקסימלי/מינימלי	$min(lst), \; max(lst)$	$O(n)$
היפוך סדר הרשימה	$lst.reverse(lst)$	$O(n)$
מעבר על הרשימה בלולאה	$for \; i \; in \; lst$	$O(n)$
מיון הרשימה	$lst.sort()$	$O(n \cdot \log n)$
הכפלת רשימה במספר	$lst \cdot t$	$O(t \cdot n)$

טבלה 1: סיבוכיות של פעולות על רשימות ו-tuples

הערות:

1. פייתון הולך ל-id של הרשימה ומוסיף את i כדי להגיע למיקום המתאים.
2. בשפת התכנות C כחלק מהגדרת $container$ יש לכתוב מה אורכה.
3. לפעמים הסיבוכיות היא $O(n)$ מפני שיש להזיז את כל הרשימה ע"מ שיהיה מקום.
4. יש להזיז את כל האיברים שאחרי i, לכן זה בעצם $O(n-i)$ (משמעותי כש-i יחסית בסוף הרשימה).
5. שקול ל-$lst.pop(-1)$.
6. יכול להיות שזה $O(n + len(container))$?

פעולה	דוגמה	סיבוכיות
len (הערה 1)	$len\left(s\right)$	$O\left(1\right)$
הוספת איבר לקבוצה	$s.add\left(object\right)$	$O\left(1\right)$
pop	$s.pop\left(\right)$	$O\left(1\right)$
ניקוי הקבוצה	$s.clear\left(\right)$	$O\left(1\right)$
יצירת קבוצה	$set(container)$	$O\left(len\left(container\right)\right)$
השוואת קבוצות	$s1 == s2,\ \ s1! = s2$	$O\left(n\right)$
מחיקת איבר	$s.remove\left(object\right),\ \ s,discard\left(object\right)$	$O\left(1\right)$
בדיקה אם איבר בקבוצה	$x\ \ in\ \ s,\ \ x\ \ not\ \ in\ \ s$	$O\left(1\right)$
בדיקת הכלה	$s < t,\ \ s <= t,\ \ t > s,\ \ t >= s$	$O\left(n\right)$
איחוד קבוצות	$s \mid t$	$O\left(n + len\left(t\right)\right)$
חיתוך קבוצות	$s\&t$	$O\left(n + len\left(t\right)\right)$
חיסור קבוצות	$s - t$	$O\left(n + len\left(t\right)\right)$
חיסור סימטרי	s^t	$O\left(n + len\left(t\right)\right)$
העתקת קבוצה	$s.copy\left(\right)$	$O\left(n\right)$
מעבר על המילון בלולאה	$for\ \ i\ \ in\ \ lst$	$O\left(n\right)$

טבלה 2: סיבוכיות של פעולות על קבוצות

פעולה	דוגמה	סיבוכיות
גישה למפתח	$d[key]$	$O\left(1\right)$
השמה	$d[key]\ \ =\ \ object$	$O\left(1\right)$
get או setdefault	$d.get\left(key, object\right),\ \ d, setdefault\left(\right)$	$O\left(1\right)$
len (הערה 1)	$len\left(d\right)$	$O\left(1\right)$
pop	$d.pop\left(i\right)$	$O\left(1\right)$
popitem	$d.popitem\left(\right)$	$O\left(1\right)$
ניקוי המילון	$d.clear\left(\right)$	$O\left(1\right)$
יצירת מילון (הערה 2)	$dict(container)$	$O\left(len\left(container\right)\right)$
השוואת מילונים	$d1 == d2,\ \ d1! = d2$	$O\left(n\right)$
מחיקת איבר	$del\ \ d\left[key\right]$	$O\left(1\right)$
view	$d.keys\left(\right),\ \ d.values\left(\right),\ \ d.items\left(\right)$	$O\left(n\right)$
בדיקה אם מפתח נמצא במילון	$x\ \ in\ \ d,\ \ x\ \ not\ \ in\ \ d$	$O\left(1\right)$
בדיקה אם ערך נמצא במילון	$x\ \ in\ \ d.values$	$O\left(n\right)$
העתקת מילון	$lst.copy\left(\right)$	$O\left(n\right)$
מעבר על המילון בלולאה	$for\ \ i\ \ in\ \ lst$	$O\left(n\right)$

טבלה 3: סיבוכיות של פעולות על מילונים

הערות:

1. בשפת התכנות C כחלק מהגדרת $container$ יש לכתוב מה אורכה.

2. כל האיברים ב-$container$ חייבים להיות $containers$ בגודל 2.