

אינטואיציה והגדרת מושג הסיבוכיות

עכשיו שהבנו איך לכתוב אלגוריתמים, נרצה להבין איך למדוד אילו אלגוריתמים יעילים יותר מאלגוריתמים אחרים. דרך אחת לעשות את זה היא על ידי מדידת הסיבוכיות של האלגוריתם. הסיבוכיות של אלגוריתם תהיה אומדן על מספר הפעולות שהאלגוריתם מבצע (כשהכוונה בפעולה לפעולות בסיסיות, כמו לדוגמה פעולות חשבון, שינוי ערך של משתנה, קריאת ערך של משתנה).

למה למדוד פעולות ולא זמן ריצה של האלגוריתם אחרי שמתכנתים אותו? זמן ריצה של הקוד תלוי בחומרה הספציפית של המחשב שמריצים אותו עליו, אם מריצים את אותו הקוד עם אותו הקלט על מחשבים שונים נקבל זמנים שונים, ואנחנו צריכים דרך להשוות יעילות של אלגוריתמים שונים.

לדוגמה, נסתכל על האלגוריתם הבא, שמחשב סכום של מערך, כאשר המערך הוא $a[0], a[1], \dots, a[n-1]$ (מגודל n):

1. $0 \rightarrow s$
2. $0 \rightarrow i$
3. כל עוד $i < n$:
4. $s + a[i] \rightarrow s$
5. $i + 1 \rightarrow i$
6. תחזיר s

אפשר לראות שמספר הפעולות שהאלגוריתם מבצע הוא $4n+3$ מכיוון שהאלגוריתם מבצע בהתחלה שתי פעולות (מאתחל את הערכים s, i) ואחר כך בלולאה הוא בודק את הערך של i , ומעדכן את הערכים של s, i . סך הכל הוא יבדוק את הערך של i $n+1$ פעמים, ויעדכן את הערכים של s, i n פעמים כל אחד. בסוף הוא עושה פעולה נוספת כדי להחזיר את s . סך הכל קיבלנו $1 + 2n + 1 + n + 2 = 4n + 3$.

החישוב הזה שעשינו כדי לספור את מספר הפעולות המדויק שהאלגוריתם מבצע, מעבר לכך שהוא מתיש, גם לא באמת נותן לנו מידע מדויק כל כך, מכיוון שלדוגמה פעולת השוואה לוקחת למחשב פחות זמן מאשר פעולת חיבור. בנוסף, כמו שאמרנו, על מעבדים שונים בסוף האלגוריתמים הממומשים לוקחים זמן שונה, וההבדל יכול להיות קבוע (למשל אם מעבד אחד מריץ קוד פי 3 יותר מהר מהשני, אז $3n + 4$ פעולות של המעבד האיטי שקולות ל- $9n + 12$ של המעבד המהיר).

ובכלל, איך משווים איזה אלגוריתם יעיל יותר?

נניח שספרנו באלגוריתם אחר $3 + \frac{n^2}{100}$ פעולות, ונתעלם לרגע מעניין המעבדים השונים (שהיינו רוצים דרך למדוד אלגוריתם בלי קשר למכונה עליה הוא רץ).

מה יותר יעיל, $3 + \frac{n^2}{100}$ או $3n + 4$ פעולות?

אם $n=10$, האלגוריתם הראשון ירוץ ב-4 פעולות, והשני ב-34.

אבל אם הקלט (n) בגודל 1000, היינו מקבלים 10003 פעולות, לעומת 3004 באלגוריתם השני.

מה יקרה בקלט בגודל מליון? תניחו לרגע שהמעבד מבצע מליון פעולות בשניה, כמה זמן יקח ל-2 התוכניות לרוץ על קלט בגודל הזה?

ההבדל בין $3 + \frac{n^2}{100}$ לבין $3n + 4$ הוא הרבה יותר מקבוע. לא משנה על איזה מעבדים נריץ את 2 האלגוריתמים, ולא משנה איזה שיפור יהיה בעתיד במעבדים, מאיזשהו שלב, כש- n מספיק גדול, $3 + \frac{n^2}{100}$ יהיה גדול יותר. והסיבוכיות היא מדד בדיוק לזה - היא בודקת כמה האלגוריתם גדל כשהקלט גדל, מה סדר גודל הגדילה של מספר הפעולות של האלגוריתם ביחס לגדילה של הקלט, והיא מאפשרת לנו להשוות בין אלגוריתמים שונים, לקבוע האם הם יעילים "אותה המידה", או האם אחד מהם באמת יעיל מהותית מהשני. באופן פחות פורמלי, נרצה לקחת את הגורם הגדול ביותר בנוסחה לכמות הפעולות שלנו, ולהתעלם מכפל בקבוע, וזו תהיה הסיבוכיות.

זה אולי נשמע קצת תאורתי ומנותק מהמציאות, אבל זה מאוד שימושי! באולימפיאדה גודל הקלט מותאם לסיבוכיות האלגוריתם היעיל, והוא מספיק חופשי כדי שהקבוע באמת לא ישפיע. נניח למשל שיש לנו שאלה עם n עד מליון, שניה אחת חסם זמן ריצה, ומעבד שמריץ 100 מליון פעולות בשניה. אלא אם כן אנחנו מתכנתים ממש גרועים, אלגוריתם עם סדר גודל של n פעולות יעבוד. גם אם זה $3n + 5$ פעולות, גם אם $4n + 100$, אפילו אם זה $20n + 1000$ פעולות עדיין לא תהיה לנו בכלל בעיה. אבל אלגוריתם עם סדר גודל של n^2 פעולות לעולם לא יסיים לרוץ בשניה, גם אם הקבוע מאוד נמוך...

אז עכשיו כשקיבלנו אינטואיציה למה אנחנו רוצים למדוד את אופן הגדילה של מספר הפעולות כתלות ב n (או כתלות בכל דבר אחר שיכול להשפיע על זמן הריצה של האלגוריתם), נראה קצת איך כותבים את זה באופן פורמלי.

הסיבוכיות של אלגוריתם תסומן בסימון $O(g(n))$ (כאשר $g(n)$ היא פונקציה כלשהי שמהווה חסם עליון על סדר הגודל של מספר הפעולות של האלגוריתם). לדוגמה, הסיבוכיות של האלגוריתם שבדקנו קודם תהיה $O(n)$ (מכיוון ש- $3n$ הוא הגורם המשמעותי ביותר, ואנחנו מתעלמים מכפל בקבוע). אם לדוגמה יהיה לנו אלגוריתם שמבצע $5n^2 + \sqrt{n} + 3n \log_2 n$ פעולות, אז הסיבוכיות שלו תהיה $O(n^2)$, מכיוון שה $5n^2$ יותר גדול מכל השאר.

עכשיו כשאנחנו יודעים למדוד סיבוכיות, אנחנו יכולים לאמוד את היעילות של אלגוריתם. באופן כללי, אלגוריתם אחד יהיה יעיל יותר מאלגוריתם אחר אם הסיבוכיות שלו תהיה קטנה יותר מהסיבוכיות של האחר, כשהכוונה בסיבוכיות קטנה היא שהפונקציה אסימפטוטית (כלומר כשגודל הקלט שואף לאינסוף) היא קטנה יותר, לדוגמה n^2 גדולה יותר מ n .

סיבוכיות במקרה הגרוע

הרבה פעמים סיבוכיות זמן הריצה של האלגוריתם שנכתוב תהיה תלויה לא רק בגודל הקלט אלא גם בקלט הספציפי. לדוגמה, אלגוריתם שרוצה לבדוק האם מספר נמצא במערך על ידי מעבר על איברי המערך מההתחלה עד הסוף, ירוץ במקרה הטוב בזמן $O(1)$ (כלומר יעשה מספר קבוע של פעולות, שלא תלוי בגודל הקלט), וזה אם לדוגמה האיבר הראשון במערך יהיה האיבר שאנחנו מחפשים. לכן כשנשאל על הסיבוכיות של האלגוריתם נתכוון לרוב לסיבוכיות במקרה הגרוע ביותר ולא במקרה הטוב. באולימפיאדה, יריצו את התוכנה שלנו על קלטים שונים ומגוונים שנועדו בין היתר למצוא מקרים גרועים לכל מיני אלגוריתמים, והריצה האיטית ביותר תיחשב כזמן הריצה שלנו.

סיבוכיות מקום

בדומה לסיבוכיות זמן, יש משמעות גם לסיבוכיות מקום- בכמה זיכרון האלגוריתם שלנו משתמש. לדוגמה, אלגוריתם שיוצר טבלה בגודל $n \times n$, ישתמש ב $O(n^2)$ מקום. $O(1)$ של סיבוכיות מקום, בדומה לסיבוכיות זמן, היא סיבוכיות המקום של אלגוריתם המשתמש בכמות זיכרון שלא תלויה בקלט של התוכנה בצורה שאינה חסומה. למשל, אם האלגוריתם שלנו מגדיר שני משתנים המכילים אות, ומשתנה המכיל מספר שלם, תרומתם לסיבוכיות המקום של האלגוריתם שלנו קבועה, אזי $O(1)$.

דוגמה - חיפוש בינארי

נניח שנתון מערך ממויין a בגודל n , כלומר מתקיים $a[0] < a[1] < \dots < a[n-1]$. נרצה לכתוב אלגוריתם שבהינתן מספר x , יגיד האם המספר נמצא במערך, ואם כן מה האינדקס שלו (כלומר עבור איזה ערך i מתקיים $a[i] = x$).

הפתרון הנאיבי יהיה לעבור על כל ערכי המערך בלולאה ולבדוק אם הם שווים ל- x . בפסאודוקוד זה יראה ככה:

1. $0 \rightarrow i$
2. כל עוד $i < n$:
3. אם $a[i] = x$
4. תחזיר i
5. $i + 1 \rightarrow i$
6. תחזיר -1 (כלומר לא הצלחנו למצוא)

הפתרון הזה אמנם עובד, אבל הוא עובד בסיבוכיות זמן של $O(n)$, והוא לא מנצל את כך שהמערך ממויין. נוכל למצוא פתרון חכם יותר שמסתמך על העיקרון הבא: בכל פעם נבדוק את הערך במקום m . אם $a[m] = x$ אז סיימנו. אם $a[m] > x$ אז אנחנו יודעים (מכיוון שהמערך עולה) שבהכרח לכל $k > m$, מתקיים $a[k] > x$, ולכן בפרט $a[k] \neq x$. לכן אנחנו לא צריכים לבדוק את כל ערכי $k > m$ מכיוון שאנחנו כבר יודעים שהערך שאנחנו מחפשים לא יהיה שם.

אם ניעזר בהבחנה הזאת, נוכל בכל שלב לפסול חצי (לפעמים חצי פחות 1, תלוי בזוגיות) מהמערך שנותר לנו. כלומר, בכל פעם נשמור את הטווח במערך (התחלה וסוף) בו יכול להיות המספר שאנחנו מחפשים. נסמן זאת עם l ו- r , כך שאנחנו יודעים בהכרח בכל שלב שאם קיים i כך ש $a[i] = x$ אז בהכרח $l \leq i \leq r$ (זה טווח של מקומות אפשריים לערך שאנחנו מחפשים). מכיוון שאנחנו לא יודעים דבר על המערך בהתחלה, נאתחל את l, r להיות $l = 0, r = n - 1$. (כאשר n זהו אורך המערך). מעכשיו, בכל שלב, נבדוק את הערך האמצעי במערך $a[l], a[l + 1], \dots, a[r - 1], a[r]$. ולפי הגודל שלו נוכל לפסול חצי מהמערך שנותר לנו. למה דווקא נבדוק את הערך האמצעי? כי אנחנו מעוניינים בסיבוכיות הטובה ביותר במקרה הגרוע! אם לא נבחר את האמצע, תמיד קיים המקרה שבו האיבר שאנחנו מחפשים יהיה דווקא בטווח הגדול יותר שהשארנו, לכן תמיד נרצה ללכת לאמצע, כי כך נפסול הכי הרבה מקומות ונשאר עם הטווח הקטן ביותר אחרי כל ניחוש במקרה הגרוע.

בפסאודו קוד זה יראה ככה:

1. $0 \rightarrow l$
2. $n - 1 \rightarrow r$
3. כל עוד $l \leq r$:
4. $m \rightarrow \frac{l+r}{2}$ (מעוגל כלפי מטה)
5. אם $a[m] = x$
6. תחזיר m
7. אם $a[m] > x$
8. $m - 1 \rightarrow r$ (אנחנו לוקחים את החצי השמאלי של המערך שנותר לנו)
9. אחרת:
10. $m + 1 \rightarrow l$ (אנחנו לוקחים את החצי הימני של המערך שנותר לנו)
11. תחזיר -1 (כלומר לא הצלחנו למצוא)

לגבי סיבוכיות הזמן: הפתרון הזה בכל פעם מחלק את המערך הקיים ב 2, ומפסיק כאשר נותר מערך בגודל 0. לכן הלולאה תרוץ $\log_2(n)$ פעמים, ולכן סיבוכיות הזמן תהיה $O(\log(n)) = O(\log_2(n))$.

הערה: אם אינכם יודעים, \log , או לוגריתם, זה ההפך מחזקה. אם $\log_2(n) = k$, אז $2^k = n$.

למשל, $\log_2(1000)$ זה בערך 10 (כי $2^{10} = 1024$), ו- $\log_2(1000000)$ בערך 20 (:

עבור קלט בגודל מליארד, האלגוריתם של לחפש איבר יקח כמליארד פעולות במקרה הגרוע, וחיפוש בינארי (האלגוריתם שמוצג כאן) רק 30!

למידע נוסף ומקיף על יעילות, ניתן לקרוא [כאן](#).

במידה ואתם מעוניינים בהסבר מקיף על יעילות זמן ריצה באנגלית, שמותאם לתכנות תחרותי, אנחנו ממליצים על האתר הבא: <https://usaco.guide/bronze/time-comp?lang=cpp>