תרגיל בית מספר 4 ~ מבוא מורחב למדעי־המחשב

שחר פרץ

27 ביולי 2024

1

א' נסכם את הממצאים בטבלה: (זמן בשניות)

$t=$	$n=$	rand=True	rand=False
10	10^0	$3.1019 \cdot 10^{-6}$	$1.2795 \cdot 10^{-6}$
5	10^1	$3.6131 \cdot 10^{-5}$	$2.2674 \cdot 10^{-5}$
2	10^2	0.0004105	0.000313
2	10^3	0.00570	0.00485
1	10^4	0.0752	0.0579
1	10^5	0.9293	0.6556
1	10^6	9.54367	7.9088
1	10^7	120.3766	111.65889
1	10^8	1660.6307	1466.0910

נשים לב, שבאופן קבוע, כמעט ללא תלות בגודל n, שימוש ב־quicksort עם אקראיות יותר מהיר מבאשר בלי. בחרתי את ערך ה־t לקטון עד לכדי 1 כאשר מגיעים לערכי n גבוהים, כי ההפרש בין התוצאות קטן ולכן הדיוק גדל, ואין צורך במספר רב של ריצות.

ב' נסכם את הממצאים בטבלה (זמן בשניות):

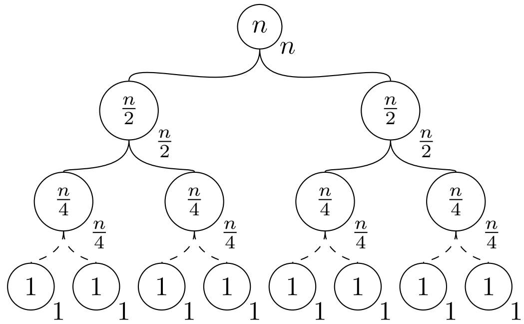
$t=$	$n=$	rand=True	rand=False
10	10^0	$5.3730 \cdot 10^{-6}$	$2.0479 \cdot 10^{-6}$
5	10^1	$8.05937 \cdot 10^{-5}$	$6.6903 \cdot 10^{-5}$
3	10^2	0.0005981	0.00158
2	10^3	0.0063425	0.1328
2	10^4	0.06749	11.5116

(לא הצלחתי להריץ עבור $n=5$ בגלל מחסור ב־RAM, ובכל מקרה בשביל $n=4$ כבר היה דרוש sys.setrecursionlimit). מהנתונים האלו נסיק כי באופן קבוע, כמעט ללת תלות בגודל n, שימוש ב־quicksort בלי אקראיות יותר מהיר מאשר עם. זאת כי באופן קבוע כאשר rand=Trueאז זמן הריצה קצר באופן משמעותי, כפי שאפשר לראות בטבלה.

המשך בעמוד הבא

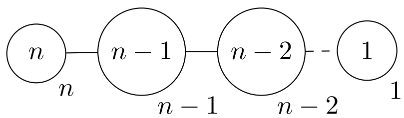
2

א' i. נסטט עץ (כמות הפעולות מצד ימין מטה לגודל הקלט שמתקבל)



לפי העץ, בשכבה ה־i יהיו 2^i צמתים, בכל צומת $\frac{n}{2^i}$ פעולות, וסה"כ נקבל $\sum_{i=1}^{\log n} 2^i i \cdot \frac{n}{2^i} = n \log n$ פעולות, כלומר הסיבוכיות של max_v1תהיה $O(n \log n)$.

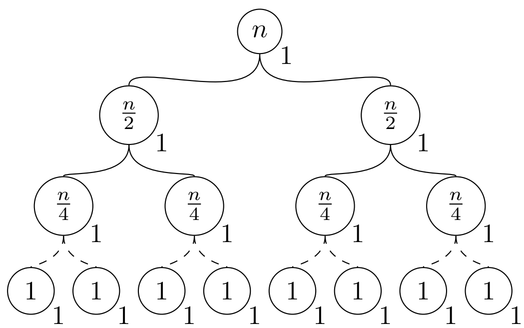
ii. נסרטט עץ גם למקטע הקוד הזה:



כאן, יהיו n node־ים בעץ, ב־node ה־i יתבצעו i פעולות, כלומר סה"כ הסיבוכיות של max_v2 תהיה $\sum_{i=1}^{n} i = \frac{n^2+n}{2} = \boldsymbol{O(n^2)}$.

ב' ננתח את סיבוכיות זמן הריצה של הפונקציות המשופרות.

i. הפעם, הסיבוכיות של כל צומת בעץ היא 1, כי חישוב מקסימום בין 2 מספרים, לקיחת אינדקס מרשימה ופעולות אריתמטיות – כולן לוקחות זמן ריצה קבוע. נצייר את העץ:



הפעם, בשכבה ה־i עדיין יהיו 2^i צמתים, אך בכל אחד מהם רק פעולה אחת, וסה"כ $\sum_{i=0}^{\log n} 2^i = 2^{\log n+1} - 1 = 2n - 1$. נקבל שסיבוכיות הפונקציה max_v1_improved תהיה $\boldsymbol{O(n)}$ – ליניארית, בהשוואה ל־max_v1שסיבוכיותו הייתה $O(n \log n)$.

ii. בכל קריאה רקורסיבית תתרחש קריאה רקורסיבית יחידה ל־$i-1$, עד שמגיעים ל־1, דבר שתירחש לאחר $n-1$ פעולות. עלותה של קריאה רקורסיבית במימוש החדש של max_v2_improvedהוא קבוע, כלומר סה"כ נקבל סיבוכיות של $n - 1 = \boldsymbol{O(n)}$ – שהיא ליניארית, בהשוואה ל־$O(n^2)$ שארך ל־max_v2.

ג' הפונקציה עומדת בדרישות הסיבוכיות, כי בכל קריאה רקורסיבית מספר בערך i היא תקרא לקריאה נוספת ב־$i-1$ עד הדעה ל־$i = 1$, שעבור $i = n$ יארוך $n - 1$ קריאות. כל קריאה מתרחשת בזמן ריצה קבוע, שכן גישה לאינדקס ברשימה ופעולות אריתמטיות כולן קבועים, ולכן נקבל סיבוכיות $n - 1 = O(n)$ כדרוש.

המשך בעמוד הבא

ב. ידוע ש־s ניתן לייצוג ע"י $O(\log n)$ ביטים, ולכן $s \leq 2^{O(\log n)}$ כלומר קיים קבוע c כך ש־$s \leq 2^{c\log n} = 2^c \cdot 2^{\log n} = 2^c n = O(n)$ (כי 2^c קבוע). בפתרון הבעיה, אנו ניעזרים בממואיזה – כלומר, כל הקריאות שפשוט לוקחות ערך מהזכרון יהיו זניחות, ונוכל לחסום מלמעלה את הפתרון ע"י חישוב גודל הטבלה (כי כל קריאה רקורסיבית שדורשת חישוב שומרת ערך בטבלה). משום שלא נוסיף מקומות נוספים לזכרון מעבר למה שנוצר ב־wrapper, גודל הזכרון הוא חסם עליון לזמן הריצה. בחרנו בזכרון בגודל של $n \times (s+1)$ כלומר $n \cdot (O(n)+1)$, לכן קיים a קבוע כך שהסיבוכיות תהיה $n \cdot (an+a) = an^2 + an = O(an^2) = \boldsymbol{O(n^2)}$, כדרוש.

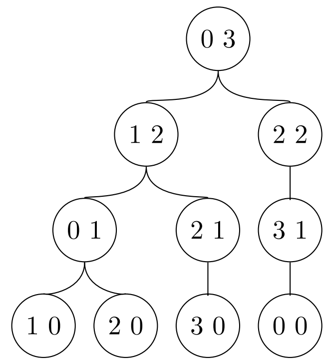
ג. נשווה את זמני הריצה.

n	`subset_sum`	`subset_sum_efficient`
3	$2.7021 \cdot 10^{-5}$	$4.73389 \cdot 10^{-5}$
6	$9.9999 \cdot 10^{-5}$	$6.08550 \cdot 10^{-5}$
9	0.0007083	0.00012372
12	0.0062107	0.00018791
15	0.047123	0.00028109
18	0.437716	0.00038526
21	3.01997	0.0003838
24	23.20511	0.000440071

נשים לב שבגלל הסיבוכיות המערכית של הפונקציה הראשונה, הפונקציה השנייה בסיבוכיות הריבועית (שעולה הרבה יותר לאט) מהירה בהרבה, וההפרש גדל ככל שהקלט גדל – בכמה סדרי גודל.

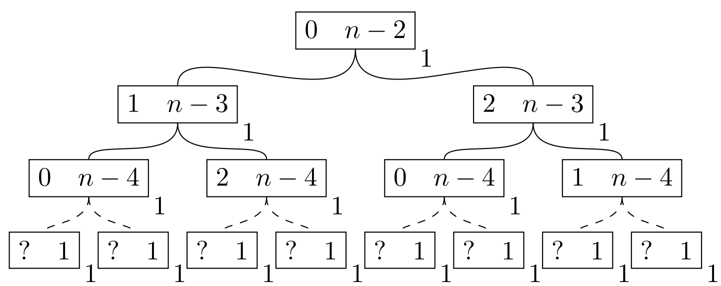
המשך בעמוד הבא

ב' i. נצייר את עץ הרקורסיה הנדרש (כל הפעולות בזמן ריצה קבוע, ולא אוסיף label ליד הצמתים כדי לא להעמיס על הסרטוט):



ii. צ.ל. קיום קלטים עבורם זמן הריצה אקספוננציאלי ב־n.

הוכחה. למען הנוחות, לכל $n \in \mathbb{N}$, נסמן $[n] = \{k \in \mathbb{N} \mid 0 \leq k < n\}$. ניגש להוכחה. נבחר $n_0 = 4$, ויהי $n \geq n_0$. נבחר $|V| = n$, $V = [n]$, יהי $c \in \mathbb{N} \wedge 2 \leq c < n$, ונבחר $E = [c] \times [c]$, $G = \langle V, E \rangle$. כלומר, תת הגרף על $[c]$ ב־G הוא קליקה. נמיר את הגרף למריצת השכנויות בהצגה של `list[list[int]]`, בגודל $n \times n$, כמתואר בשאלה. אזי, ונבחר $s = 0 \in [c], t = n - 1 \notin [c]$. נבחר $c = 3$, דבר שמתאפשר בהינתן $c < n < 4 = n_0$, ו־$t = n - 2 < n$. נקבל את עץ הרקורסיה הבא:



יהיו n שכבות בעץ הרקורסיה, כאשר בכל שכבה יש 2^i צמתים שדורשים סיבוכיות קבועה. נסכם; הסיבוכיות תהיה:

$$\sum_{i=1}^{n} 2^i = 2^n - 1 = \boldsymbol{O(2^n)}$$

בהתאם לנדרש. ■

ג' i. עבור:

```python
A = [[0, 0, 0] for _ in range(3)]
path_v2(A, 0, 1, 1)
```

במקום לקבל False נקבל `RecursionError`, כי בכל קריאה רקורסיבית עבורה נניח $t = 1$, נקבל $mid = t//2 = 0$ ולכן $k - mid = 1 - 0 = 1$, כלומר תמיד תהיה קריאה רקורסיבית שתמשיך עם $k = 1$.

iii. צ.ל. הפונקציה סופר־פולינומיאלית:

הוכחה. נגדיר בדומה לסעיף הקודם $[n] := \mathbb{N} \cap [0, n), \implies |[n]| = n$. לכן, נוכל לבחור $V = [n], E = [n-1] \times [n-1],\ G = \langle V, E \rangle$. בחירה כזו חוקית לכל $n \geq 2 := n_0$. נבנה את A מתוך הגרף כמתואר בשאלה, ע"י מציאת מטריצת השכנויות וייצוגה כרשימה דו־ממדית. נבחר $s = 0 \in [n],\ \ t = n - 1 \in [n] \setminus [n-1]$ ו־$t = n - 1 < n$. משום שאין נתיב, הרצת הפונקציה `path_v2(A, s, t ,k)` תיאלץ בהכרח לעבור כל הקריאות הרקורסיבות שלה. בכל "שכבה" של עומק רקורסיה בעץ, נקבל $(n-1)^i$ צמתים, ובגלל ש־k קטן פי 2 עד להיותו 1 או 0, הרקורסיה תיעצר רק לאחר $i = \log n$, כלומר בשכבה הנמוכה ביותר נקבל $(n-1)^{\log(n-1)}$ צמתים, שזה חסם תחתון סופר־פולינומיאלי מנתוני השאלה. סה"כ הוכחנו שהפונקציה סופר־פולינומיאלית. ■

ד' i. a. **לא קיים** – החל מנקודת היציאה, הפונקציה תעבור על כל ההמשכים האפשריים, ובאינדוקציה תגיע בסופו של דבר לכל המסלולים האפשריים (תחת ההנחה שהיא מגיעה לכל האיטרציות, כלומר היא גומרת לרוץ, שמותר לנו להניח כי הנחנו בשלילה שהיא מחזירה False, כלומר מסתיימת).

b. **קיים** – עבור `A = [[1, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [0, 0, 0, 0]]` ו־`s = 0, t = 3`.

c. **לא קיים** – כאשר הפונקציה עוברת בין שני צמתים, היא תבדוק תקינות במטריצת השכנויות, כלומר לא יכול להיות שהיא תעבור במסלול לא תקין.

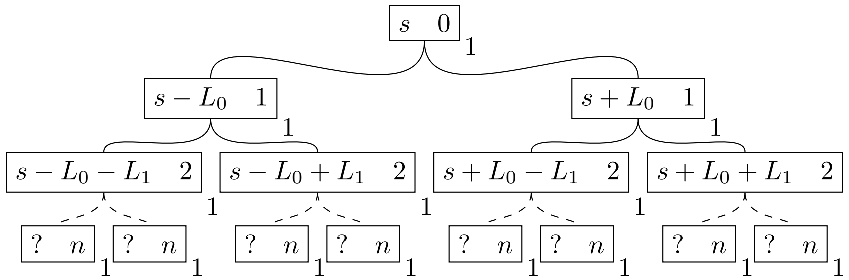
d. **קיים** – עבור `A = [[1, 0, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [0, 0, 0, 0]]` ו-`s = 0, t = 3`.

ii. הקוד משתמש בממואיזה עבור טבלה בגודל n, ולכן עם ידרוש מעל n^2 חזרות זו סתירה, כי זה אומר שניסינו לכתוב לטבלה מעל מה שנמצא בה (תוך ספירת הלולאה עצמה). הפתרון תקין, כי אם היינו במיקום מסויים ואז חזרנו אליו – אז המסלול מעגלי ולא יגיע ליעד, ולכן אפשר לסמן את הצומת ככזו שלא רלוונטית.

המשך בעמוד הבא

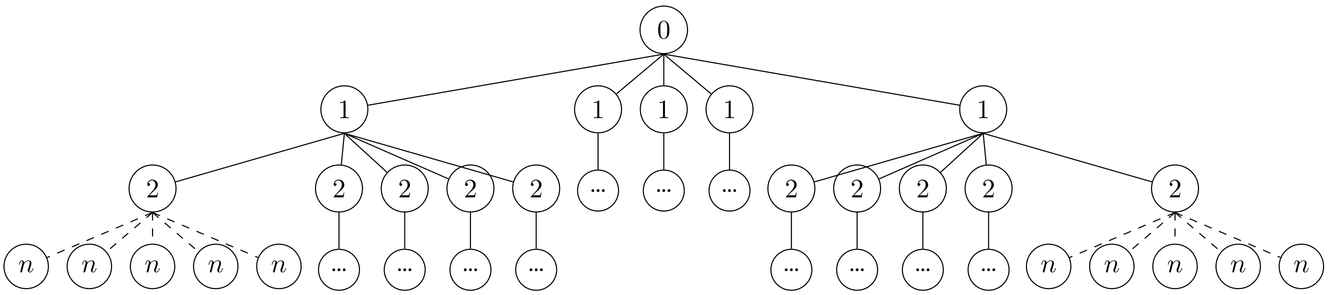
5

א' להלן עץ הרקורסיה של הקוד שלי כתלות בפרמטרים $s, start := s, 0$ (כי L קבוע, ו־$start = 0$ כברירת מחדל). נניח $|L| = n$. נסמן L[i]=: L_i.



יהיו n שכבות, בכל אחת 2^i צמתים כמות הצמתים (גודל העץ) תהיה $O(2^n)$ $\sum_{i=1}^{n} 2^i = \mathbf{2^n - 1} = O(2^n)$.

ב' באופן דומה לסעיף הקודם, אך הפעם לא נצייר את s שכן ערכו ארוך יותר במקרה זה ואין זה קובע את הסיבוכיות.



באופן דומה, גם כאן יש i שכבות, אך הפעם 5^i צמתים בשכבה, כמות גודל עץ הרקורסיה יהיה:

$$\sum_{i=1}^{n} 5^i = 1 \cdot \frac{5^n - 1}{5 - 1}$$

שנחסם ע"י $O(5^n)$.

ג' נבנה בתאם לפתרון נוסחת נסיגה:

$$C_n = \sum_{i=1}^{n} C_{n-i} C_i$$

כאשר, תחת ההנחה שהאורך הוא m, נגדיר $n = \frac{m-1}{2}$ (נוסחאת הנסיגה נכונה ישירות מהקוד, שבו יש לולאה שמבצעת את הקריאות הרקורסיביות הן C_i, C_{n-i}). נסתמך על העובדה שידוע פתרון לנוסחא הזו, הוא מספר קטלן. זאת, בהתאמה לעובדה שכמות האפשרויות למבני סוגריים מאוזנים מוגדרת (או נחשבת להגדרה מקובלת) להיות מספר קטלן. ידוע:

$$C_n = \prod_{k=2}^{n} \frac{n+k}{k} \geq \prod_{k=2}^{n} \frac{n}{k} = \frac{2}{n!} \cdot n^{n-1} = \Omega(n!)$$

המשך בעמוד הבא

6

א' צ.ל. $\forall n \in \mathbb{Q}. \exists p, q, \in \mathbb{N}. \gcd(p,q) \neq 1 \wedge \frac{p}{q} = n$, ובשלילה לוגית, באופן שקול $\neg(\exists n \in \mathbb{Q}. \forall p, q \in \mathbb{N}. \frac{p}{q} = n \implies \gcd(p,q) \neq 1)$ (לפי משפטי דה־מורגן על לוגיקה, הפיכת כמתים והגדרת גרירה).

הוכחה. נניח בשלילה שקיים $n \in \mathbb{N}$ כך שלכל $p, q \in \mathbb{Q}$ אם $\frac{p}{q} = n$ אז $\gcd(p,q) \neq 1$ (זה שקול לוגית לשלילה של הטענה, הצרנתית). משום ש־$n \in \mathbb{Q}$ אז קיימים $q, p \in \mathbb{N}$ כך ש־$n = \frac{p}{q}$ (לפי הגדרה). נתבונן ב־$\gcd(p,q) = c \neq 1$, משום ש־$\forall a, b \in \mathbb{N}. \gcd(a,b) \leq 1$ וטבעי, אז $c \leq 2$. נוסף על כך, ידוע $c \mid p \wedge c \mid q$ לפי הגדרת gcd, ונסמן $p', q' = \frac{p}{c}, \frac{q}{c} \in \mathbb{N}$ בהתאמה, סה"כ $\frac{p}{q} = \frac{\frac{p'}{c}}{\frac{q'}{c}} = \frac{p'}{q'}$. מהנחת השלילה, $\gcd(p', q') := c' \neq 1$, כלומר קיים $c' \mid p' \wedge c' \mid q'$, ולכן $c' \mid \frac{p}{c'}$ וגם $c' \mid \frac{q}{c'}$. ידוע $c' \in \mathbb{N}$. בהתאם לטענות הקודמות, $G = c'c$ מקיים $G \mid p \wedge G \mid q$, כלומר $\gcd(p,q) \geq G$, סה"כ $c \leq c'c$ כלומר $1 \leq c'$, בסתירה לכך ש־$c' \leq 2$. סה"כ הנחת השלילה הוכחה כשגויה, וההוכחה הושלמה. ■

המשך בעמוד הבא

סיכום התוצאות:

- **הפונקציה string_to_int**: הבוט כשל. הקוד שלי מגדיר `n = len(s)`, כאשר נאמר לרובוט ובשאלה ש-k הוא האורך של s, אך כתוצאה מהגדרה זו הבוט ראה שאני חוזר על n וכתב:

  "Your implementation has a time complexity of O(n), while the correct time complexity is O(k)."

  למרות העובדה ש־n בכלל לא מוגדר מחוץ לקוד שלי, ואין תלות בשני פרמטרים.

  (אומנם זו בחירה לא טובה שלי של שמות משתנים, אך אין דבר פגם בקוד או בסיבוכיות שלו)

- **הפונקציה int_to_string**: הקוד כתב, נכונה, שצדקתי והסיבוכיות מתאימה.
- **הפונקציה sort_strings1**: הקוד כתב, נכונה, שצדקתי והסיבוכיות מתאימה.
- **הפונקציה sort_strings2**: הקוד כתב, נכונה, שצדקתי והסיבוכיות מתאימה.

בכל המקרים בהם הבוט כתב שצדקתי, הוא הסתייג בטענה שהוא עוזר AI ועלול לעשות טעויות.

לקונטקסט, הקוד שלי מצורף כאן:

```python
# Q3_a
def char_to_int(c: str) -> int:
    """helper for functions in Q11 (run in O(1))"""
    if c not in ["a", "b", "c", "d", "e"]:
        raise ValueError("only str a, b, c, d, e are allowed")
    else:
        return ord(c) - 97


def string_to_int(s: str) -> int:
    n: int = len(s)
    s = s[::-1]

    output: int = 0
    for i in range(n):
        output += 5 ** (i) * char_to_int(s[i])

    return output


# Q3_b
def int_to_char(i: int) -> str:
    """helper for functions in Q11 (run in O(1))"""
    if i not in range(0, 5):
        raise ValueError("only intgers in range(1, 6) are allowed")
    else:
        return chr(i + 97)


def int_to_string(k: int, n: int) -> str:
    output = ""

    for i in range(k):
        scale = 5 ** (k - i - 1)
        output += int_to_char(
            n // scale
        )  # e.g. 69 // 25 = 2, then 19 // 5 = 3 and last 4 // = 4 => (2, 3, 4) = "bcd"
        remove = scale * (n // scale)  # equal to n % scale
        n -= remove

    return output


# Q3_c
def sort_strings1(lst: list[str], k: int) -> list[str]:
    """sorts strings in alphabetic order"""
    n: int = len(lst)
    output: list[str] = []

    # create an empty list
    memory: list[list[int, str]] = [[0, ""] for _ in range(5**k)]  # exactly O(5**k)

```

```python
    # assign to the `memory` the number of times that each value [=index] is shown in the list.
    for i in range(n):
        value = string_to_int(lst[i])  # exactly \Theta(kn)
        memory[value][0] += 1
        memory[value][1] = lst[i]

    # filling `output`
    for i in range(5**k):
        curVal: int = memory[i][0]  # exactly \Theta(5**k)
        if curVal == 0:
            continue

        revalue = memory[i][1]
        for _ in range(curVal):
            output.append(revalue)  # exactly \Theta(n)

    return output


# Q3_e
def sort_strings2(lst: list[str], k: int) -> list[str]:
    n = len(lst)
    output: list[str] = []

    for i in range(5**k):
        cur = int_to_string(k, i)
    for j in range(n):
        if lst[j] == cur:
            output.append(lst[j])

    return output
```