מבוא מורחב למדעי המחשב ∼ עמית ווינשטין ∼ חזרה על חומר

שחר פרץ

2 ליולי 2024

Q5, HW2 ..... (1)

נתבונן בקוד הבא:

```
x = 8
y = 'cs'
z = x
y = x
y += 12
lst1 = [x, z]            #1
lst2 = [x, y, lst1]
def what(x, lst):
    x = 10
    lst[0] = "i"
    lst2 = [x, y, what]   #2
    return lst2
lst1 = lst1 + lst2        #3
lst3 = what(x, lst1)      #4
```

אני לא ממש עומד להתיק את זה אבל הפתרון שלי זמין בפתרון לשיעורי הבית הללו בענן.

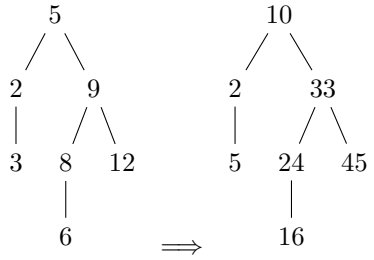
```
def append(x, lst=[]):
    lst.append(x)
    print(lst)
append(1)
append(2)
```

טענה: יודפס `[0, 1]`. הערך הדיפולטיבי של `lst1` מאותחל עוד בקריאה של הפונקציה, ולכן הוא ישאר אח"כ. לכן לא מומלץ לשים default value שהוא mutable.

CUM SUM ..... (2)

(they knew)

שאלה מתוך המבחן. תנו דעתכם על העץ הבא:



תכונה: כל המפתחות בתת עץ הימני, עבור צומת נתון, גדולים ממנו, וכל המפתחות בתת העץ השמאלי של צומת נתון, קטנים ממנו. התכונה הזו נכונה לכל עץ חיפוש בינארי.

נרצה להגדיר sumsum כפונקציה שמשנה את ערך המפתח של כל צומת להיות סכום כל המפתחות שתחתיו. דוגמה להפעלת הפעולה מצויירת לעיל.

שאלה כללית: איך למיין עצי חיפוש?

- in-order: שמאל ← אני ← ימין
- pre-order: אני ← שמאל ← ימין
- post-order: שמאל ← ימין ← אני

את כולם אפשר לממש בזמן ריצה ליניארי. ב־cumsum נרצה לעשות מיון in-order. ניעזר בתבנית שניתנה במבחן:

```
def cumsum(self):
    def cumsum_rec(node, s) -> int:
        # edge case
        if node is None:
            return s
        s = cumsum_rec(node.left, s)
        node.key += s
        s = cumsum_rec(node.left, s)
        node.key += s
        s = cumsum_rec(node.right, node.key)
        return s

    cumsum_rec(self.root, 0)
```

איך מגיעים לפתרון: כאשר מטיילים על עץ חיפוש בינארי, יש לנו סדר שאפשר לעבוד איתו לפתור ליניארית.

שאלה: איך היה אפשר לסדר את העץ הזה מחדש, כך ש־pre-order היה מחזיר את הכל ממוין: תשובה: סרוך שהולך ימינה.

GENERATORS . . . . . (3)

שאלות על גנרטורים מהמבחן. **הגדרה:** גנרטור הוא בעל השהייה סופית אמ"מ כל קריאה ל־next מסתיימת תוך זמן סופי. בכל סעיף, צריך להגיד האם ניתן לממש אותו בעזרת גנרטור עם השהייה סופית

1. ייצור מספרים ראשוניים – **אפשר**. הגנרטור זוכר מהראשוני האחרון שהוא בדק נעבור על כל הטבעיים, ונחזיר כאשר נמצא ראשוני. בין שני ראשוניים יש מספר סופי של מספרים.
2. בהינתן g עם השהייה סופית שמחזיר מספרים, להחזיר את איברי g החיוביים. **אי־אפשר**. דוגמה נידית: נתבונן ב־$g = \lambda n \in \mathbb{N}. -1$, ואז הגנרטור יאלץ לעשות ∞ פעולות על מציאת איבר חיובי (כי אין כאלו).
3. בהינתן g עם השהייה סופית שמחזיר מספרים, להחזיר את הסכומים החלקיים של g. **אפשר**. אפשר לשמור את הסכומים החלקיים שהוחזרו קודם וכל פעם מוסיפים את מה שהיה קודם.