אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

תרגיל בית מספר 5 - להגשה עד 26/05/2022 בשעה 23:55

קיראו בעיון את הנחיות העבודה וההגשה המופיעות באתר הקורס, תחת התיקייה assignments. חריגה מההנחיות תגרור ירידת ציון / פסילת התרגיל.

הנחיות לצורת ההגשה:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם hw5_012345678.pdf ו- hw5_012345678.py.
- עבור קובץ ה-pdf: מומלץ מאוד להקליד את התשובות בו. ניתן גם לכתוב את התשובות בכתב יד ברור ולסרוק אותן, אבל שימו לב שתשובות שיכתבו בכתב יד לא ברור לא יבדקו, וערעורים בנושא זה לא יתקבלו.
- עבור קובץ ה-py: השתמשו בקובץ השלד skeleton5.py כבסיס לקובץ אותו אתם מגישים.
- לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת py.

הנחיות לפתרון:

- הקפידו לענות על כל מה שנשאלתם.
- בכל שאלה, אלא אם מצוין אחרת באופן מפורש, ניתן להניח כי הקלט תקין.
- אין להשתמש בספריות חיצוניות פרט לספריות math, random, אלא אם נאמר במפורש אחרת.
- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
  להנחיה זו מטרה כפולה:
  1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  2. כדי להרגיל אתכם להבעת טיעונים באופן מתומצת ויעיל, ללא פרטים חסרים מצד אחד אך ללא עודף בלתי הכרחי מצד שני. זוהי פרקטיקה חשובה במדעי המחשב.
- נדרוש שכל הפונקציות שאנו מממשים תהיינה יעילות ככל הניתן. לדוגמה, אם ניתן לממש פתרון לבעיה בסיבוכיות $O(logn)$, ואתם מימשתם פתרון בסיבוכיות $\Theta(n)$, תקבלו ניקוד חלקי על הפתרון.
- בשאלות שבהן ישנה דרישה לניתוח סיבוכיות זמן הריצה, הכוונה היא לסיבוכיות זמן הריצה של המקרה הגרוע ביותר (worst-case complexity).

טבלת גרסאות:

11.05.2022	גרסה ראשונה
13.05.2022	גרסה נוכחית – תיקון טעויות כתיב

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

__שאלה 1__

בתרגול 9 ראינו את מבנה הנתונים רשימה מקושרת לוגריתמית והצגנו (בין השאר) את מתודת __contains__. הקלט למתודה היה רשימה מקושרת לוגריתמית self וערך val. הפונקציה מניחה כי הרשימה self ממוינת (כלומר, ערכי ה-val של כל צומת ברשימה ממוינים בסדר עולה) ומחזירה True אס"ם יש צומת ברשימה שערכו הוא val.

א. בתרגול ראינו כי זמן הריצה של המתודה __contains__ הוא __**לכל היותר**__ $O(\log^2 n)$. הראו כי חסם זה הדוק. כלומר, עליכם להראות כי קיימים שני __**קבועים**__ $n_0, c > 0$ כך שלכל $n > n_0$ קיימת __**לפחות**__ רשימה אחת באורך n __**ולפחות**__ ערך val אחד עבורם זמן הריצה של המתודה הוא __**לפחות**__ $c \cdot \log^2 n$.

ב. בשלד הקובץ מופיעה המתודה __contains__ כפי שהוצגה בתרגול. שפרו את המתודה כך שזמן הריצה יהיה יעיל אסימפטוטית מ-$\Theta(\log^2 n)$. כלומר, עליכם לשנות את המתודה כך שהיא תבצע (כמו מקודם) חיפוש ברשימה מקושרת לוגריתמית ממוינת בזמן T כך ש-$T = O(\log^2 n)$ אבל $\log^2 n \neq O(T)$.
הסבירו בקובץ ה-PDF בקצרה כיצד פועל האלגוריתם החדש ומהו החסם ההדוק ביותר על זמן הריצה שלו.

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

<u>שאלה 2</u>

בהינתן מספר טבעי $n > 0$, **הגורמים הראשוניים שמרכיבים את n** הם רשימת המספרים הראשוניים **הגדולים מ- 1** שמכפלתם שווה ל-n. כלומר, אם נסמן ב-P את רשימת הגורמים הראשוניים של n (כולל חזרות) באורך k. מתקיים:

$$\prod_{0 \leq i < k} P[i] = n$$

וכן כל $p \in P$ הוא מספר ראשוני.

לדוגמה, אם $n = 12$ אז $P = [2, 2, 3]$ שכן $2 \cdot 2 \cdot 3 = 12$, וכל האיברים ב-P הם ראשוניים. שימו לב שאותו גורם ראשוני יכול להופיע מספר פעמים. שימו לב, עבור $n = 1$ הרשימה היא $P = []$.

בשאלה זו נממש מספר מתודות במחלקה FactoredInteger אשר מייצגת מספרים טבעיים באמצעות רשימה ממוינת בסדר עולה של הגורמים הראשוניים שלהם. ניתן להיווכח שייצוג זה הוא **ייצוג טוב** – כלומר שיש התאמה חח״ע ועל בין קבוצת המספרים הטבעיים לבין קבוצת כל הסדרות העולות הסופיות של מספרים ראשוניים (רשות: הוכיחו זאת).

<u>הערה</u> – קושיה של בעיית הפירוק לגורמים ראשוניים:
בעיית הפירוק לגורמים ראשוניים: בהינתן מספר טבעי n עם ייצוג בינארי באורך b, יש למצוא את רשימת הגורמים הראשוניים שלו. זוהי בעיה קשה שלא ידוע לה אלגוריתם בעל זמן ריצה פולינומי ב-b. בדומה לבעיית הלוג הדיסקרטי עליה דיברנו בכיתה, על קושי זה מסתמכים רבים מפרוטוקולי ההצפנה המודרניים.
עובדה מעניינת היא שקיים **אלגוריתם קוונטי** יעיל המפרק מספר לגורמיו הראשוניים. עובדה זו היא אחת הסיבות לכך שפיתוח מחשב קוונטי חזק הפכה למטרה מרכזית במדעי המחשב בשנים האחרונות.
אתם מוזמנים לקרוא עוד על <u>מחשב קוונטי</u> ועל <u>אלגוריתם שור</u> בוויקיפדיה.

א. להלן מתודת האתחול של המחלקה FactoredInteger:

```
class FactoredInteger:

    def __init__(self, factors):
        """ Represents an integer by its prime factorization """
        assert is_sorted(factors)
        self.factors = factors
        number = 1
        for p in factors:
            assert(is_prime(p))
            number *= p
        self.number = number
```

שימו לב שהמתודות is_sorted ו- is_prime (שאותה ראינו בתרגול) נתונות לכם בקובץ השלד. הפונקציה is_sorted בודקת האם רשימת הקלט ממויינת ומחזירה True אם כן ו-False אחרת.

נסמן ב-k את אורך הרשימה factors, וב-n את מספר הביטים בייצוג הבינארי של המספר number שמתקבל בסוף המתודה.

i. בהינתן n כלשהו, מהו טווח הערכים האפשריים של k בהנחה שהרשימה מאותחלת באופן תקין? הסבירו.

ii. הראו שסיבוכיות הזמן של פונקציית האתחול היא $O(n^3)$ במקרה הגרוע ביותר, ותנו דוגמה שמוכיחה שחסם זה הוא הדוק, כלומר לכל n תנו דוגמה לקלט שעבורו זמן הריצה של הפונקציה הוא $\Theta(n^3)$. ניתן להניח כי זמן הריצה של הפונקציה modpower שרצה על קלט בעל b ביטים הוא $\Theta(b^3)$.

רמז: עבור רשימת הראשוניים $P = [p_1, \ldots, p_k]$ סמנו את הרשימה המתאימה $A = [a_1, \ldots, a_k]$ כאשר a_i מייצג את כמות הביטים במספר p_i . נתחו את סיבוכיות הפונקציה כתלות בערכי a_i . שימו לב שהתוצאה הסופית צריכה להיות תלויה רק ב n.

בסעיפים הבאים נממש מספר פונקציות המטפלות באובייקטים מסוג FactoredInteger. דרישות הסיבוכיות מוגדרות כתלות ב-k ו-m, שהם אורכי הרשימות factors של הפרמטרים self ו-other בהתאמה. בפרט, הניחו שפעולות אריתמטיות על האיברים בתוך factors ועל number נעשות בזמן $O(1)$. כמו כן, הניחו שהאובייקטים כבר קיימים, זאת אומרת שחישובי הסיבוכיות אינם צריכים לכלול את האתחול.

ב. ממשו את הפונקציות המובנות הבאות של המחלקה FactoredInteger בקובץ השלד. שימו לב כי self ו-other הם אובייקטים מטיפוס FactoredInteger.

- **`__repr__(self)`** - מחזירה מחרוזת המייצגת את המספר באופן הבא:

$$< number\colon p_1 * p_2 \ldots * p_k >$$

שימו לב שבמחרוזת אין כלל רווחים. לדוגמה, עבור המספרים 12 ו-1 הפונקציה מחזירה בהתאמה:

```
<12:2*2*3>
<1:>
```

- **`__eq__(self, other)`** – מחזירה True אם self ו-other מייצגים את אותו מספר, ו-False אחרת. הפונקציה צריכה לרוץ בזמן $O(1)$.

- **`__mul__(self, other)`** – מתודה מובנית שתומכת באופרטור *
מחזירה אובייקט מסוג FactoredInteger המייצג את תוצאת המכפלה בין self ו-other. הפונקציה צריכה לרוץ בסיבוכיות זמן $O(k + m)$.

- **`__pow__(self, other)`** – מתודה מובנית שתומכת באופרטור **
מחזירה אובייקט מסוג FactoredInteger המייצג את תוצאת החזקה $self^{other}$ הפונקציה צריכה לרוץ בסיבוכיות זמן $O(k * other)$. שימו לב שאין להשתמש באופרטור * של המחלקה.

דוגמאות הרצה:

```
>>> n1 = FactoredInteger([2, 3])       # n1.number = 6
>>> n2 = FactoredInteger([2, 5])       # n2.number = 10
>>> n3 = FactoredInteger([2, 2, 3, 5]) # n3.number = 60
>>> n3                                 # __repr__
<60:2*2*3*5>
>>> n1 == FactoredInteger([2,3])       # __eq__
True
>>> n1 * n2                            # __mult__
<60:2*2*3*5>
>>> n1 ** n2                           # __pow__
 <60466176:2*2*2*2*2*2*2*2*2*2*3*3*3*3*3*3*3*3*3*3>
```

ג. היזכרו בהגדרה של gcd שראיתם בכיתה: בהינתן $a, b \in \mathbb{N}$, **המחלק המשותף הגדול ביותר** של a ו-b (הנקרא גם **gcd** – greatest common divider), הוא המספר המקסימלי אשר מחלק גם את a וגם את b. לדוגמה, ה-gcd של 12 ו-8 הוא 4, שכן 4 מחלק גם את 8 וגם את 12, וכל מספר גדול מ-4 לא מחלק את שניהם. ממשו את המתודה **`gcd(self, other)`** המחלקה FactoredInteger המקבלת שני אובייקטים מהמחלקה, self ו-other ומחזירה אובייקט מטיפוס FactoredInteger המייצג את המחלק הגדול ביותר של self ו-other. נסמן את אורכי הרשימות של self ו-other ב-k_s, k_o. הפונקציה צריכה לרוץ בסיבוכיות זמן של $O(k_s + k_o)$. שימו לב שהאלגוריתם של אוקלידס שראיתם בכיתה לא מקיים זאת.

ד. בהינתן $a, b \in \mathbb{N}$, **הכפולה המשותפת המינימלית** (הנקראת גם **lcm** – least common multiple) היא המספר הקטן ביותר שמתחלק גם ב-a וגם ב-b. לדוגמה, הכפולה המשותפת המינימלית של 6 ו-15 היא 30 שכן 30 מתחלק גם ב-6 וגם ב-15, וכל מספר קטן מ-30 לא מתחלק בשניהם (זהו ה"מכנה המשותף" המינימלי, כפי שאתם מכירים מחיבור שברים).

i. בקובץ השלד נתון לכם מימוש של הפונקציה **`lcm(self, others)`**. הסבירו בקצרה מה עושה הפונקציה ואיך היא מחשבת את ה-lcm של self וכל ה-FactoredInteger ברשימה **`others`**.

ii. נניח שסה"כ משתתפים בחישוב s מספרים ראשוניים (סה"כ כל אורכי הרשימות בכל האובייקטים שמשתתפים נסכמים ל-s). נתחו את סיבוכיות זמן הריצה של הפונקציה על בסיס s.

דוגמאות הרצה:

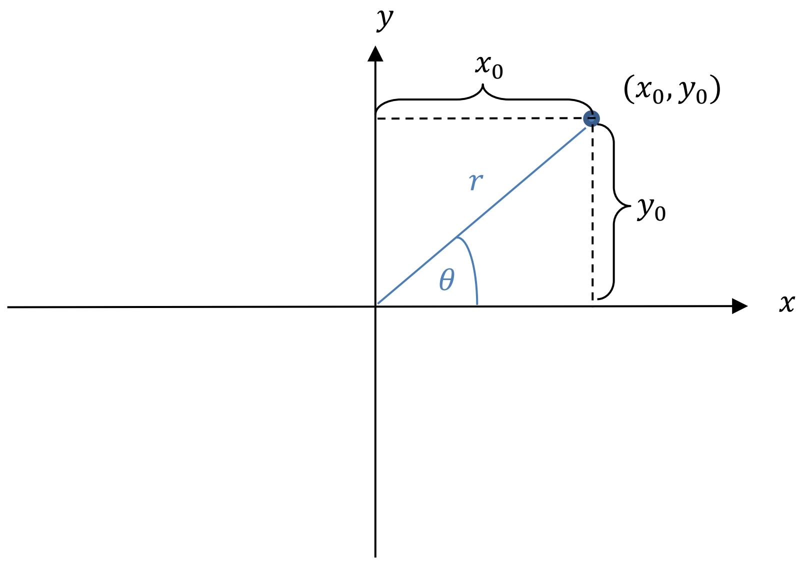
```
>>> n1 = FactoredInteger([2, 2, 3])   # n1.number = 12
>>> n2 = FactoredInteger([2, 2, 2])   # n2.number = 8
>>> n1.gcd(n2)
<4:2*2>
>>> n3 = FactoredInteger([2, 3])
>>> n4 = FactoredInteger([3, 5])
>>> n3.lcm([n4,n2])
<60:2*2*3*5>
```

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

<u>שאלה 3</u>

בשאלה זו נעסוק בנקודות במישור, כלומר נקודות שניתן למקם על מערכת צריכים דו-מימדית. כל נקודה (x_0, y_0) ניתנת לייצוג על ידי **קואורדינטות פולריות** $(r(x_0, y_0), \theta(x_0, y_0))$, כאשר $r \in \mathbb{R}^+$ הוא המרחק של הנקודה (x_0, y_0)מראשית הצירים, ו- $\theta \in [0,2\pi)$ היא הזווית ברדיאנים בין ציר ה-x לנקודה (x_0, y_0).

ראו שרטוט להמחשה:



נגדיר את המחלקה Point אשר תייצג נקודה במישור. נרצה לשמור גם את הקואורדינטות הסטנדרטיות (הנקראות **קואורדינטות קרטזיות**) וגם את הקואורדינטות הפולריות. להלן מתודת האתחול של המחלקה:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.r = math.sqrt(x**2 + y**2)
        self.theta = math.atan2(y, x)
        if self.theta < 0:
            self.theta = angle + 2*math.pi
```

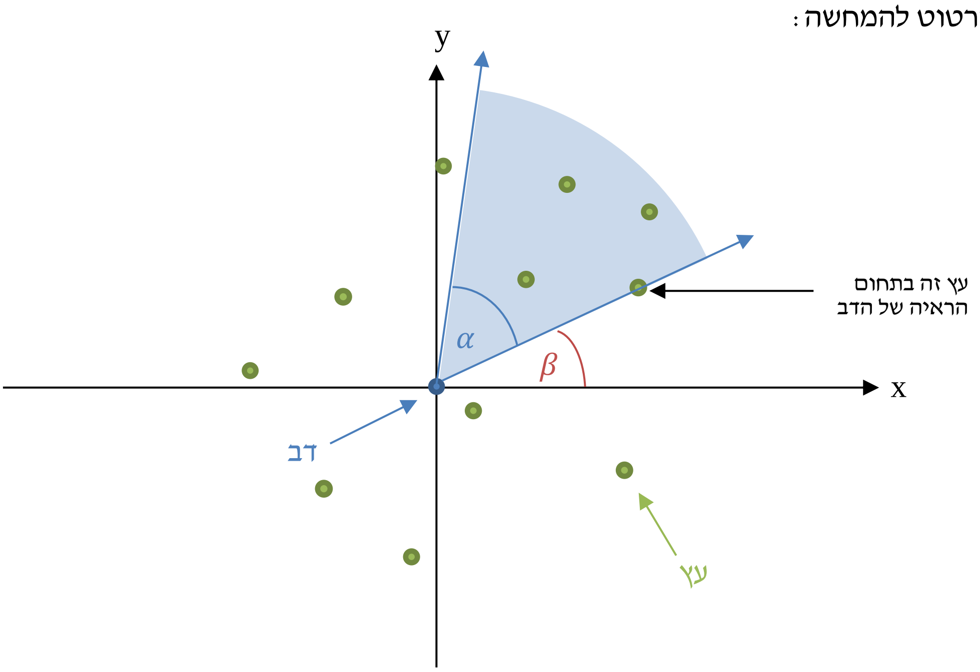
<u>הערה</u>: atan2 היא פונקציה מספריית math שמחשבת את הזווית הרצויה בטווח בין $-\pi$ ל-π, אבל ב-Point self.theta תמיד תהיה בטווח $[0,2\pi)$.

<u>הערות כלליות לשאלה</u>:

- עקרונית לרוב לא נרצה לאפשר למשתמש חיצוני לגשת אל השדות הפנימיים של המחלקה. עם זאת, לשמירה על פשטות המחלקה, בתרגיל זה ניתן לגשת לשדות של Point באופן ישיר.
- השדות במחלקה הם מטיפוס float. כזכור, חישובים אריתמטיים ב-float הם לא מדויקים מטבעם. בפרט, בשאלה זו ניתן להתעלם משגיאות הנובעות מחוסר דיוק של float.

בשאלה זו נרצה להשתמש במחלקה Point על מנת לעזור בפתרון שתי בעיות.

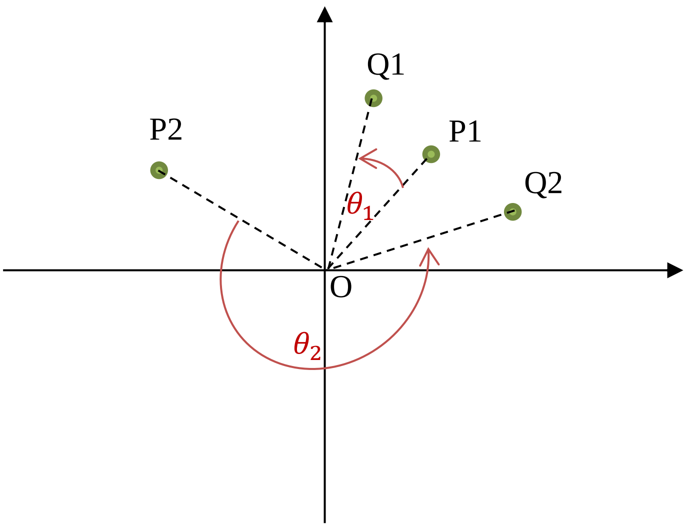
א. בעיית העצים: דב עומד בנקודה מסויימת ביער, ומסביבו n עצים. לדב יש זווית ראיה α והוא יכול לראות למרחק אינסופי בתחום זווית ראיה זו. הדב יכול להסתובב על מקומו ובכך לראות חלקים שונים של היער (ומספר שונה של עצים, בהתאמה).
תחילה נרצה למדל את השאלה באמצעות מונחים איתם יהיה לנו קל יותר לעבוד: נייצג את העצים באמצעות נקודות במישור xy, כאשר $(0,0)$ היא הנקודה בה עומד הדב, במטרה לדמות "מבט על" של היער. בכדי למדל את הכיוון שאליו הדב מסתכל, נגדיר את β להיות הזוית בין ציר x ובין שולי תחום הראיה הימני של הדב. ראו שרטוט להמחשה:



המשימה: בהינתן זווית ראיה α, עלינו למצוא את הזווית β שתמקסם את מספר הנקודות הירוקות בתוך משולש הפיצה הכחול (כולל שפת המשולש). הניחו כי אף עץ חלילה לא מוחץ את הדב – כלומר אין עץ בנקודה $(0,0)$.

נפתור את השאלה בשני שלבים:

i. נסמן ב-O את ראשית הצירים. ממשו את הפונקציה angle_between_points(self, other) במחלקה Point, המקבלת 2 נקודות (אובייקטים מסוג Point), נסמן $P = self, Q = other$, ומחזירה את הזווית



בה יש לסובב את הקטע PO נגד כיוון השעון כדי להגיע לקטע QO. הפונקציה מחזירה זווית בטווח $[0, 2\pi)$. לפניכם שרטוט של שני זוגות נקודות, והזוויות המתאימות להן:

דוגמאות הרצה:

```
>>> p1 = Point(1, 1)            # p1.theta = 0.25 * pi
>>> p2 = Point(0, 3)            # p2.theta = 0.5 * pi
>>> p1.angle_between_points(p2) # self = p1, other = p2
0.78539816339                   # 0.25 * pi
>>> p2.angle_between_points(p1) # self = p2, other = p1
5.49778714378                   # 1.75 * pi
```

ii. ממשו את הפונקציה **`find_optimal_angle(trees, alpha)`** המקבלת רשימה trees של n נקודות (אובייקטים מטיפוס Point), וזווית ראיה alpha $\in (0, 2\pi)$, ומחזירה את הזווית $\beta \in [0,2\pi)$ הממקסמת את מספר העצים שרואה הדב (תיתכן יותר מזווית β אחת מתאימה – החזירו זווית כלשהי).

<u>הנחיה</u>: על הפונקציה לרוץ בסיבוכיות זמן $O(nlogn)$ כאשר n מספר העצים.
<u>רמז</u>: התחילו בלמיין את רשימת הנקודות על פי ערכי ה-θ שלהן והיעזרו בפונקציה שמימשתם בסעיף הקודם.

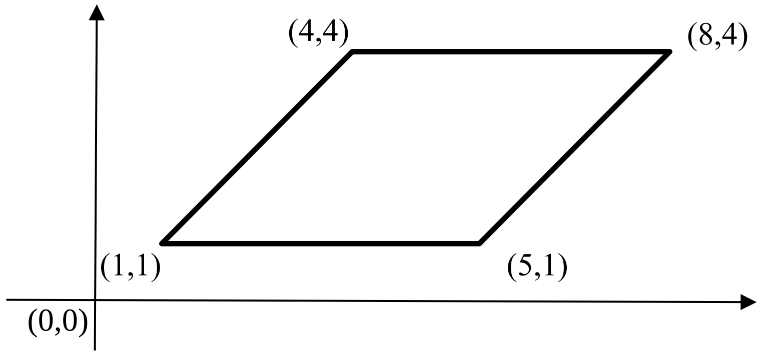
דוגמת הרצה (מומלץ לשרטט את הנקודות על מערכת צירים כדי להבין מדוע זה ערך ההחזרה):

```
>>> trees = [Point(2,1),Point(0,3),Point(-1,3),
Point(-1,1),Point(-1,-1),Point(0,-5)]
>>> find_optimal_angle(trees, 0.25 * math.pi)
1.5707963267948966  # 0.5 * pi
```

השתכנעו שבדוגמה זו יש זווית **אחת ויחידה** שעונה על השאלה. כאמור, בדוגמאות אחרות יתכן כי יותר מזווית אחת מתאימה (בפרט, יתכנו אינסוף זוויות מתאימות).

ב. **בעיית המצולעים:** בגאומטריה, מצולע (באנגלית, *Polygon*) הוא חלק ממישור המתוחם על ידי מספר סופי של קטעים. כל אחד מהקטעים הללו נקרא צלע וכל נקודה בה נפגשות שתי צלעות סמוכות נקראת קודקוד. במקרה שלנו, נרצה למדל מצולע על ידי רשימה מקושרת של אובייקטים מטיפוס `Point`. שימו לב למחלקה `Polygon` שנמצאת בקובץ השלד. שימו לב שהקטע האחרון במצולע הוא הקטע בין האיבר האחרון לבין האיבר הראשון ברשימה. נסמן את מספר הקודקודים במצולע ב-n.

לדוגמא המקבילית הבאה:

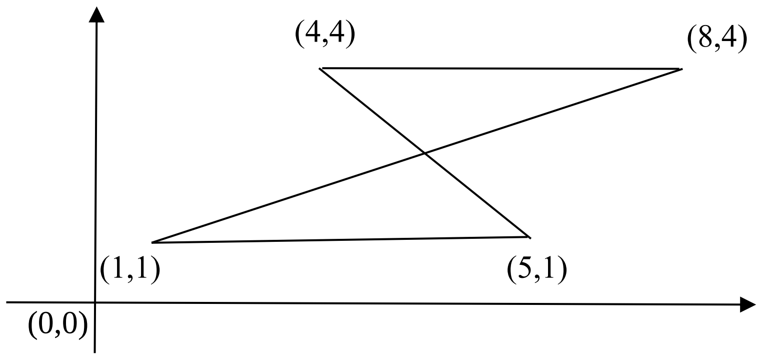


תאותחל על ידי הפקודה הבאה:

```
>>> parallelogram =
Polygon(Linked_list([Point(1,1),Point(4,4),Point(8,4),Point(5,1)]))
```

שימו לב, כי סדר הנקודות משנה. זאת אומרת שסדר שונה של נקודות קובע מצולע אחר.

הגדרה (מצולע פשוט). מצולע פשוט הוא מצולע שצלעותיו נחתכות אחד על ידי השניה בקצותיהן בלבד. שימו לב שהמצולע הבא אינו מצולע פשוט:



והוא מאותחל על ידי הפקודה הבאה:

```
>>> not_simple =
Polygon(Linked_list([Point(1,1),Point(8,4),Point(4,4),Point(5,1)]))
```

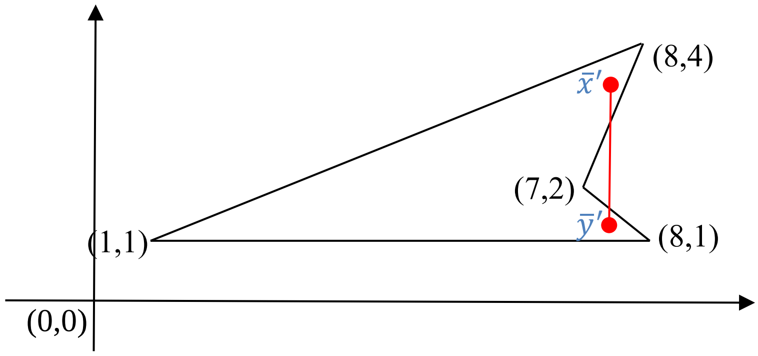
i. ממשו את המתודה edges(self) של המחלקה `Polygon` שמקבלת אובייקט ממחלקת `Polygon` המייצג **מצולע פשוט** self, ומחזירה רשימה (list של פייתון) של הזווית הפנימית (הזוויות שנמצאות בתוך המישור התחום על ידי המצולע) במעלות (כלומר מספר בין 0 ל-360) לכל קודקוד במצולע. זאת אומרת, המקום ה-i של הרשימה המוחזרת תהיה הזווית שמתאימה לקודקוד ה-i ברשימה המקושרת של הפוליגון עם השכן שלו מימין והשכן שלו משמאל.

הערות:

- הזוויות של נקודה במצולע מחושבת על ידי הזווית בין שני הקטעים שמתחילים בנקודה הזו. זאת אומרת, שתצטרכו לחשב את הזווית בין שלשות של נקודות. הקוד לחישוב הזווית נמצא בקובץ השלד.
- שימו לב שבסעיף זה אנחנו רוצים להשתמש במעלות ולא ברדיאנים כמו בסעיף א. זאת כדי להקל עליכם, מכיוון שבמעלות יהיה לכם קל יותר להבין מציור מה בערך הזווית בכל קודקוד של המצולע.
- אל תשכחו את הקצוות של הרשימה.
- על המתודה לרוץ בסיבוכיות זמן $O(n)$.

הגדרה (מצולע קמור). יהי P מצולע פשוט. P יקרא מצולע קמור אם לכל זוג נקודות $\bar{x},\bar{y}\in\mathbb{R}^2$ כך שהנקודות $\bar{x},\bar{y}$ נמצאות בתוך P, מתקיים כי הקטע שעובר בין $\bar{x}$ ו- $\bar{y}$ נמצא כולו בתוך P.

המקבילית היא דוגמה למצולע שהוא קמור. לפנינו דוגמה למצולע שאינו קמור:



והוא מאותחל על ידי הפקודה הבאה:

```
>>> not_convex =
Polygon(Linked_list([Point(1,1),Point(8,1),Point(7,2),Point(8,4)]))
```

כדוגמה, מסומנים $\bar{x}',\bar{y}'$ והקטע שעובר ביניהם. ניתן להבחין שהקטע שבין $\bar{x}'$ ו-$\bar{y}'$ אינו נמצא כולו בתוך המצולע.

ii. ממשו את המתודה is_convex(self) של המחלקה `Polygon`, שבודקת האם הוא מצולע קמור.

הערות:

- ניתן להשתמש במתודה edges(self).
- על המתודה לרוץ בסיבוכיות זמן $O(n)$.
- אל תשכחו את הקצוות של הרשימה.

רמז: בהינתן רשימת זוויות של מצולע פשוט, ניתן לבדוק לכל זווית ברשימה שהיא מקיימת תנאי מקומי (כלומר ביחס למספר קבוע של זוויות ברשימה לפניה ואחריה) בכדי להכריע האם מצולע הוא קמור. חישבו מהו תנאי זה.

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

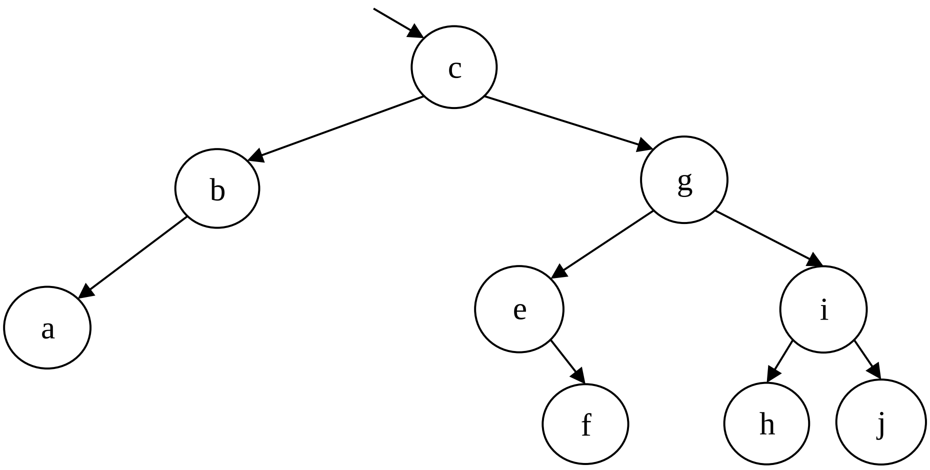
<u>שאלה 4</u>

השאלה עוסקת בעצי חיפוש בינאריים, ובמחלקה Binary_search_tree. הניחו בשאלה זו שהמפתחות (השדה key) בצמתים הינם מחרוזות למטרת השאלה, אין חשיבות לערכי השדה val.

הגדרה (עץ בינארי q-מאוזן). יהי $q \in (0{,}0.5)$. נאמר שעץ בינארי T הוא q-מאוזן אם אחד התנאים הבאים מתקיים:

1. T הוא עלה (שני בניו הם `None`).
2. T הוא עץ שלשורשו יש בן ימני `None` ובן שמאלי עלה.
3. T הוא עץ שלשורשו יש בן ימני עלה ובן שמאלי `None`.
4. T הוא עץ ששני הבנים של שורשו הם עצים בינאריים q-מאוזנים, ובנוסף, נסמן את כמות הצמתים של בן ימני ב n_r ואת כמות הצמתים של בן שמאלי ב n_l ואת $n = n_l + n_r$ אז מתקיים:
$$\min\left\{\frac{n_l}{n}, \frac{n_r}{n}\right\} \geq q.$$

זוהי הגדרה רקורסיבית שבה תנאים 1-3 מתארים מקרי בסיס של עצים קטנים שהם q-מאוזנים. תנאי 4 הוא התנאי שמתאר את כלל הנסיגה בהגדרה. להלן דוגמה לעץ חיפוש בינארי.



עבור (למשל) $q = 0.25$, זהו עץ q-מאוזן:

- הצמתים עם המפתחות a,b,e,f,h,j עונים על תנאי הבסיס 1-3.
- צומת עם מפתח i: $n_l = 1$, $n_r = 1$, $n = 1 + 1 = 2$. כיוון ש $q = \frac{1}{4} \leq \frac{1}{2}$, ובניה גם הם עצים q-מאוזנים, אז צומת זו היא עץ q-מאוזן.
- צומת עם מפתח g: $n_l = 2$, $n_r = 3$, $n = 2 + 3 = 5$. כיוון ש $q = \frac{1}{4} \leq \frac{2}{5}$, ובניה גם הם עצים q-מאוזנים, אז צומת זו היא עץ q-מאוזן.
- צומת עם מפתח c: $n_l = 2$, $n_r = 6$, $n = 2 + 6 = 8$. כיוון ש $q = \frac{1}{4} \leq \frac{2}{8}$, ובניה גם הם עצים q-מאוזנים, אז צומת זו היא עץ q-מאוזן.

שימו לב:

- עבור (למשל) $q = \frac{1}{3}$, עץ זה הוא לא q-מאוזן.
- עץ ריק (ששורשו הוא `None`) הוא לא q-מאוזן (עבור כל q), לפי ההגדרה.

א. ממשו את המתודה is_q_balanced(self, q) של המחלקה Binary_search_tree. המתודה מקבלת כקלט עץ self ומספר $q \in (0,0.5)$ (מטיפוס float) ומחזירה שני ערכים, האחד בוליאני והשני שלם (int) (כלומר יוחזר tuple של שני ערכים אלה). הערך הבוליאני יחזור True אמ"ם העץ הוא עץ q-מאוזן. במקרה שהמתודה מחזירה True, אז הערך השלם שחוזר הוא כמות הצמתים בעץ, אחרת יחזור הערך -1.

<u>הנחיות</u>:
1. על המתודה לרוץ בסיבוכיות זמן $O(n)$ כאשר n מספר הצמתים בעץ.
2. אין להוסיף שדות למחלקות Tree_node ו- Binary_search_tree.

<u>דוגמאות הרצה</u>:

```
# t1 is balanced for some q
>>> t1 = Binary_search_tree()
>>> t1.insert('c', 10)
>>> t1.insert('b', 10)
>>> t1.insert('a', 10)
>>> t1.insert('g', 10)
>>> t1.insert('e', 10)
>>> t1.insert('f', 10)
>>> t1.insert('i', 10)
>>> t1.insert('h', 10)
>>> t1.insert('j', 10)
>>> t1.is_q_balanced(0.25)
(True, 9)
>>> t1.is_q_balanced(0.3)
(False, -1)
```

```
# t2 is not balanced for any q
>>> t2 = Binary_search_tree()
>>> t2.insert('f', 13)
>>> t2.insert('e', 13)
>>> t2.insert('c', 13)
>>> t2.insert('b', 13)
>>> t2.insert('a', 13)
>>> t2.insert('g', 13)
>>> t2.insert('h', 13)
>>> t2.insert('i', 13)
>>> t2.insert('j', 13)
>>> t2.is_q_balanced(0.1)
(False, -1)
```

```
# t3 is balanced for some q
>>> t3 = Binary_search_tree()
>>> t3.insert('b', 0)
>>> t3.insert('a', 0)
>>> t3.insert('g', 0)
>>> t3.insert('e', 0)
>>> t3.insert('c', 0)
>>> t3.insert('f', 0)
>>> t3.insert('i', 0)
>>> t3.insert('h', 0)
>>> t3.insert('j', 0)
>>> t3.is_q_balanced(0.125)
(True, 9)
>>> t3.is_q_balanced(0.13)
(False, -1)
```

```
# t4 not balanced for any q
>>> t4 = Binary_search_tree()
>>> t4.insert('b', 10)
>>> t4.insert('a', 10)
>>> t4.insert('c', 10)
>>> t4.insert('g', 10)
>>> t4.insert('e', 10)
>>> t4.insert('f', 10)
>>> t4.insert('i', 10)
>>> t4.insert('h', 10)
>>> t4.insert('j', 10)
>>> t4.is_q_balanced(0.01)
(False, -1)
```

ב. כיתבו בקובץ ה-pdf מהי סיבוכיות זמן הריצה של מתודת חיפוש מפתח (המתודה lookup המופיעה במחלקה BinarySearchTree בקובץ השלד) בעץ q-מאוזן עבור $q \in (0,0.5)$ קבוע כלשהו, כפונקציה של n כאשר n הוא מספר הצמתים בעץ. הסבירו את תשובתכם בקצרה.

שאלה 5 – Hashing

נתונה רשימה של n מחרוזות $[s_0, s_1, \dots, s_{n-1}]$, לאו דווקא שונות זו מזו. בנוסף נתון $k > 0$, וידוע שכל המחרוזות באורך לפחות k (ניתן להניח זאת בכל הפתרונות שלכם ואין צורך לבדוק או לטפל במקרים אחרים).
אנו מעוניינים למצוא את כל הזוגות הסדורים של אינדקסים שונים (i, j), כך שקיימת חפיפה באורך k בדיוק בין רישא (התחלה) של s_i לסיפא (סיומת) של s_j. כלומר $s_i[:k] == s_j[-k:]$.
לדוגמה, אם האוסף מכיל את המחרוזות הבאות:

```
s0 = "a"*10
s1 = "b"*4 + "a"*6
s2 = "c"*5 + "b"*4 + "a"
```

אז עבור $k = 5$ יש חפיפה באורך k בין הרישא של s0 לבין הסיפא של s1, ויש חפיפה באורך k בין הרישא של s1 לבין הסיפא של s2. שימו לב שאנו לא מתעניינים בחפיפות אפשריות של מחרוזות עם עצמן, כמו למשל החפיפה באורך 5 בין רישא של s0 לסיפא של עצמה. לכן, הפלט במקרה זה יהיה שני הזוגות $(0,1)$ ו-$(1,2)$. אבל ייתכן שיש שתי מחרוזות זהות, ואז כן נתעניין בחפיפה כזו. למשל עבור s0=s1="aaa" ועבור $k = 1$ הפלט אמור להיות $(0,1)$ ו-$(1,0)$.

א. נציע תחילה את השיטה הבאה למציאת כל החפיפות הנ"ל: לכל מחרוזת נבדוק את הרישא באורך k שלה אל מול כל הסיפות באורך k של כל המחרוזות האחרות. ממשו את הפתרון הזה בקובץ השלד, בפונקציה **`prefix_suffix_overlap(lst,k)`**, אשר מקבלת רשימה (מסוג list של פייתון) של מחרוזות, וערך מספרי k, ומחזירה רשימה עם כל זוגות האינדקסים של מחרוזות שיש ביניהן חפיפה כנ"ל. אין חשיבות לסדר הזוגות ברשימה, אך יש כמובן חשיבות לסדר הפנימי של האינדקסים בכל זוג.

דוגמאות הרצה:

```
>>> s0 = "a"*10
>>> s1 = "b"*4 + "a"*6
>>> s2 = "c"*5 + "b"*4 + "a"
>>> prefix_suffix_overlap([s0,s1,s2], 5)
[(0, 1), (1, 2)] #could also be [(1, 2), (0, 1)]
```

ב. ציינו מהי סיבוכיות הזמן של הפתרון הזה במקרה הגרוע, כתלות ב-n וב-k במונחים של $O(\dots)$. הניחו כי השוואה בין שתי תת מחרוזות באורך k דורשת $O(k)$ פעולות במקרה הגרוע. ציינו גם מתי מתקבל המקרה הגרוע, בהנחה שהשוואת מחרוזות עוברת תו-תו בשתי המחרוזות במקביל משמאל לימין, ומפסיקה ברגע שהתגלו תווים שונים.

ג. כעת נייעל את המימוש ונשפר את סיבוכיות הזמן (בממוצע), ע"י שימוש במנגנון של טבלאות hash. לשם כך נשתמש במחלקה חדשה בשם Dict, שחלק מהמימוש שלה מופיע בקובץ השלד. מחלקה זו מזכירה מאוד את המחלקה Hashtable שראיתם בהרצאה, אבל ישנם שני הבדלים:

1) בקוד מההרצאה האיברים בטבלה הכילו רק מפתחות (keys), בדומה ל-set של פייתון, ואילו אנחנו צריכים לשמור גם מפתחות וגם ערכים נלווים (values), בדומה לטיפוס dict של פייתון. המפתחות במקרה שלנו יהיו רישות באורך k של המחרוזות הנתונות, ואילו הערך שנלווה לכל רישא כזו הוא האינדקס של המחרוזת ממנה הגיעה הרישא (מספר בין 0 ל-$n-1$). חישוב ה-hash לצורך הכנסה וחיפוש במילון מתבצע על המפתח בלבד.

2) מכיוון שיכולות להיות רישות זהות למחרוזות הנתונות, נרצה לאפשר חזרות של מפתחות ב-Dict (ראו בדוגמה בהמשך).

השלימו בקובץ השלד את המימוש של המתודה `find(self, key)` של המחלקה Dict, המתודה מחזירה רשימה (list של פייתון) עם **כל** ה-values שמתאימים למפתח key הנתון (לא חשוב באיזה סדר). אם אין כאלו תוחזר רשימה ריקה.

דוגמאות הרצה:

```
>>> d = Dict(3)
>>> d.insert("a", 56)
>>> d.insert("a", 34)
>>> d #calls __repr__
0 []
1 []
2 [['a', 56], ['a', 34]]

>>> d.find("a")
[56, 34] #order does not matter
>>> d.find("b")
[]
```

ד. השלימו את מימוש הפונקציה `prefix_suffix_overlap_hash1(lst, k)`, שהגדרתה זהה לזו של `prefix_suffix_overlap(lst, k)`, אלא שהיא תשתמש במחלקה Dict מהסעיף הקודם. כאמור, כל הרישות יוכנסו למילון תחילה, ואז נעבור על כל הסיפות ונבדוק לכל אחת אם היא נמצאת במילון.

ה. לצורך סעיף זה בלבד, הניחו כי אין שתי מחרוזות עם אותו סיפא, אותה רישא, או רישא של מחרוזת כלשהי ששווה לסיפא של מחרוזת כלשהי. בפרט, התנאי האחרון מבטיח שהפלט של `prefix_suffix_overlap` יהיה רשימה ריקה (אין התאמות). ציינו מהי סיבוכיות הזמן של הפתרון מסעיף ד' **בממוצע** (על פני הקלטים שמקיימים את התנאי של סעיף זה), כתלות ב-n וב-k במונחים של $O(\ldots)$. הניחו כי השוואה בין שתי תת מחרוזות באורך k דורשת $O(k)$ פעולות במקרה הגרוע, וכך גם חישוב hash על מחרוזת באורך k נמקו את תשובתכם בקצרה.

<u>סוף</u>