אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2021

<u>תרגיל בית מספר 5 - להגשה עד 01/06/2021 בשעה 23:55</u>

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

<u>הגשה</u>:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton5.py כבסיס לקובץ ה py אותו אתם מגישים.
  לא לשכוח לשנות את שם הקובץ למספר ת״ז שלכם לפני ההגשה, עם סיומת py.
- 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.
- מכיוון שניתן להגיש את התרגיל בזוגות, עליכם בנוסף למלא את המשתנה SUBMISSION_IDS שבתחילת קובץ השלד. רק אחת מהסטודנטיות בזוג צריכה להגיש את התרגיל במודל.
- הקפידו לענות על כל מה שנשאלתם.
- 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.

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2021

שאלה 1

א. היזכרו בהגדרה של $a \ mod \ b$ שראיתם בכיתה:

נאמר ש- $a \ mod \ b = r$ אם ניתן לרשום את a בתור $a = k \cdot b + r$ עבור $k \in \mathbb{Z}$, או באופן שקול אם ניתן לרשום את r בתור $r = t \cdot b + a$ עבור $t \in \mathbb{Z}$ (בפרט מתקיים הקשר $t = -k$).

הוכיחו את התכונות החשובות הבאות:

i. לכל $a, b, c \in \mathbb{N}$ מתקיים $\big((a \ mod \ b) + (c \ mod \ b)\big) mod \ b = (a + c) mod \ b$.

ii. לכל $a, b, c \in \mathbb{N}$ מתקיים $\big((a \ mod \ b) \cdot (c \ mod \ b)\big) mod \ b = (a \cdot c) mod \ b$.

iii. הסיקו מ-ii באינדוקציה כי לכל $a, b, c \in \mathbb{N}$ מתקיים: $(a \ mod \ b)^c \ mod \ b = a^c \ mod \ b$.

ב. ראינו בכיתה כי בפרוטוקול *Diffie-Helman*, בהינתן g, p המפתחות הפומביים, ו- $x = g^a \ mod \ p$ המסר שמחושב על ידי אליס ונשלח באופן גלוי, לא ידועה דרך יעילה למצוא את המפתח הסודי a (בעיה זו נקראת גם בעיית הלוג הדיסקרטי). להלן קטע קוד ממחברת תרגול 8 אשר מנסה בהינתן g, p, x למצוא את המפתח הסודי a על ידי מעבר על פני כל ערכי a האפשריים:

```
def crack_DH(p, g, x):
    ''' find secret "a" that satisfies g**a%p == x
        Not feasible for large p '''
    for a in range(1, p - 1):
        if pow(g, a, p) == x:
            return a
    return None #should never get here
```

שימו לב כי יתכן שהפונקציה תחזיר $a' \neq a$ עבורו מתקיים השוויון לעיל.

למשל, עבור הפרמטרים $p = 7, g = 13$ והמפתח הסודי $a = 4$, נקבל כי $x = 13^4 \ mod \ 7 = 1$ אבל מתקיים כי $13^2 \ mod \ 7 = 1$ ולכן האלגוריתם הנ"ל יחזיר את $a' = 2 \neq a$. נרצה להוכיח שגם במקרה כזה מציאת a' תעזור לנו "לפצח" את הפרוטוקול ולגלות את הסוד המשותף:

הוכיחו כי בהינתן $g, p, x = g^a \ mod \ p, y = g^b \ mod \ p$ ו- a' כך ש- $g^a \ mod \ p = g^{a'} \ mod \ p$ ניתן לחשב ביעילות את הסוד המשותף $g^{ab} \ mod \ p$. רמז – היעזרו בסעיף א'.

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2021

<u>שאלה 2</u>

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

$$\prod_{p \in P} k = n$$

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.

בשאלה זו נממש מספר מתודות במחלקה FactoredInteger אשר מייצגת מספרים טבעיים באמצעות רשימה 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>:

- בעיית הפירוק לגורמים ראשוניים (דהיינו: בהינתן מספר n, מצאו את רשימת הגורמים הראשוניים שלו) היא בעיה קשה שלא ידוע לה אלגוריתם קלאסי יעיל. 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 [מחשב קוונטי]() ועל [אלגוריתם שור]() בוויקיפדיה.

א. להלן מתודת האתחול של המחלקה FactoredInteger:

```
class FactoredInteger:

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

שימו לב שהמתודות is_sorted ו- is_prime (שראינו בתרגול) נתונות לכם בקובץ השלד.

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

i. תנו חסם עליון הדוק לסיבוכיות זמן הריצה במקרה הגרוע ביותר של פונקציית האתחול כתלות בפרמטרים n ו-k. הניחו כי פעולות ההשוואה $==, >, <$ נעשות בזמן $O(1)$. פרטו את חישוביכם.

ii. בהינתן n כלשהו, מהו טווח הערכים האפשריים של k? הסבירו.

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 $O(1)$.

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:

  $$< number\colon p_1, p_2, \dots, p_k >$$

  שימו לב שבמחרוזת אין כלל רווחים. לדוגמה, עבור המספר 12 הפונקציה מחזירה:

  `<12:2,2,3>`

- **`__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 $O(k+m)$.

- **`__floordiv__(self, other)`** – מתודה מובנית שתומכת באופרטור //
  מחזירה אובייקט מסוג FactoredInteger המייצג את תוצאת החלוקה $\frac{self}{other}$ במידה שהמספרים מתחלקים זה בזה ללא שארית, או None אם self לא מתחלק ב-other. הפונקציה צריכה לרוץ בסיבוכיות זמן $O(k+m)$.

<u>דוגמאות הרצה</u>:

```
>>> 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>
>>> n3 // n2                           # __floordiv__
<6:2,3>
>>> n2 // n1
None
```

ג. היזכרו בהגדרה של gcd שראיתם בכיתה: בהינתן $n, k \in \mathbb{N}$, **המחלק הגדול ביותר** של n ו-k (הנקרא גם **gcd** – greatest common divider), הוא המספר המקסימלי אשר <u>מחלק</u> גם את n וגם את k. 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 הפונקציה **`gcd(self, other)`** המחזירה אובייקט מטיפוס FactoredInteger המייצג את המחלק הגדול ביותר של self ו-other. הפונקציה צריכה לרוץ בסיבוכיות זמן של $O(k+m)$. שימו לב שאלגוריתם אוקלידס שראיתם בכיתה לא מקיים זאת.

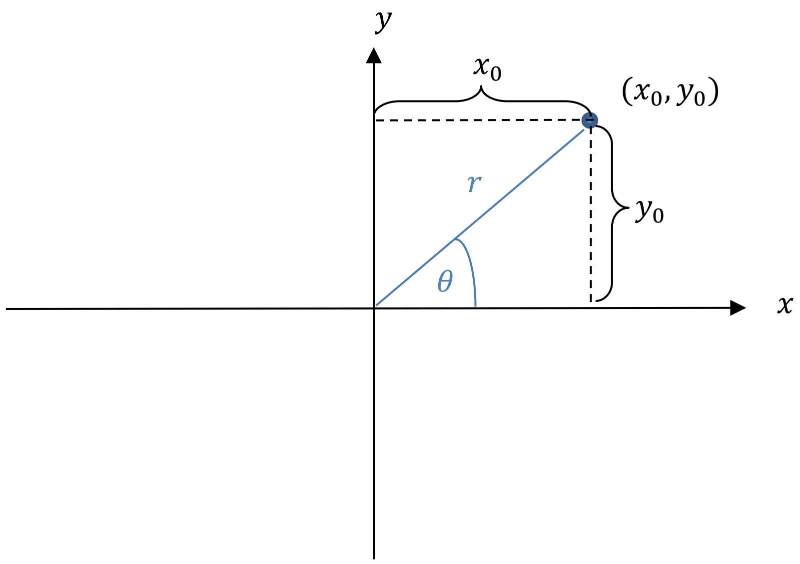
ד. בהינתן $n, k \in \mathbb{N}$, **הכפולה המשותפת המינימלית** (הנקראת גם **lcm** – least common multiple) היא המספר הקטן ביותר <u>שמתחלק</u> גם ב-n וגם ב-k. לדוגמה, הכפולה המשותפת המינימלית של 6 ו-15 היא 30 שכן 30 מתחלק גם ב-6 וגם ב-15, וכל מספר קטן מ-30 לא מתחלק בשניהם (זהו ה"מכנה המשותף" המינימלי, כפי שאתם מכירים מחיבור שברים).
ממשו את הפונקציה **`lcm(self, other)`** המחזירה אובייקט מטיפוס FactoredInteger המייצג את הכפולה המשותפת המינימלית של self ו-other. הפונקציה צריכה לרוץ בסיבוכיות זמן $O(k+m)$.

דוגמאות הרצה:

```
>>> n1 = FactoredInteger([2, 2, 3])   # n1.number = 12
>>> n2 = FactoredInteger([2, 2, 2])   # n2.number = 8
>>> FactoredInteger.gcd(n1, n2) # Equivalent to n1.gcd(n2)
<4:2,2>
>>> n3 = FactoredInteger([2, 3])
>>> n4 = FactoredInteger([3, 5])
>>> FactoredInteger.lcm(n3, n4) # Equivalent to n3.lcm(n4)
<30:2,3,5>
```

<u>שאלה 3</u>

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)
```

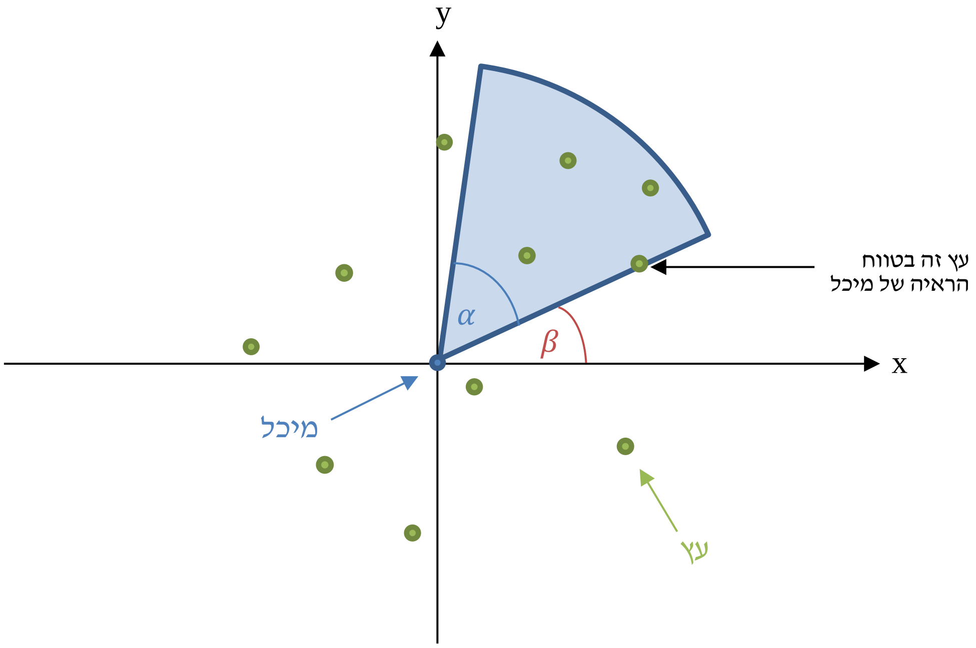
<u>הערה</u>: atan2 היא פונקציה מספריית math שמחשבת את הזווית הרצויה בטווח בין $-\pi$ ל-π.

<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, אך בשאלה זו נתייחס לחישוב כאילו הוא מדויק. בפרט, ניתן להתעלם משגיאות הנובעות מחוסר דיוק של float.

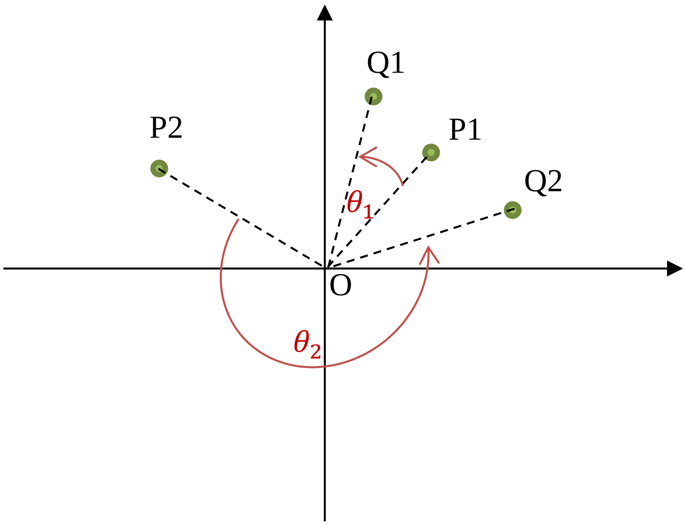
בשאלה זו נרצה להשתמש במחלקה Point על מנת לעזור למיכל ולאלחנן בפתרון שתי בעיות.

א. מיכל עומדת ביער, ומסביבה n עצים. בהינתן זווית ראיה α, עליה למצוא את הזווית β אליה היא צריכה להסתכל על מנת ללכוד בזווית הראיה שלה כמה שיותר עצים. תחילה נרצה למדל את השאלה באמצעות מונחים איתם יהיה לנו קל יותר לעבוד: נייצג את העצים באמצעות נקודות במישור, כאשר $(0,0)$ היא הנקודה בה עומדת מיכל, במטרה לדמות "מבט על" של היער. ראו שרטוט להמחשה:



עלינו למצוא את הזווית β שתמקסם את מספר הנקודות הירוקות בתוך משולש הפיצה הכחול (כולל שפת המשולש). הניחו כי אף עץ לא מוחץ את מיכל – כלומר אין עץ בנקודה $(0,0)$.

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 ל-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 $\text{alpha} \in (0, 2\pi)$, ומחזירה את הזווית $\beta \in [0,2\pi)$ הממקסמת את מספר העצים שרואה מיכל (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
```

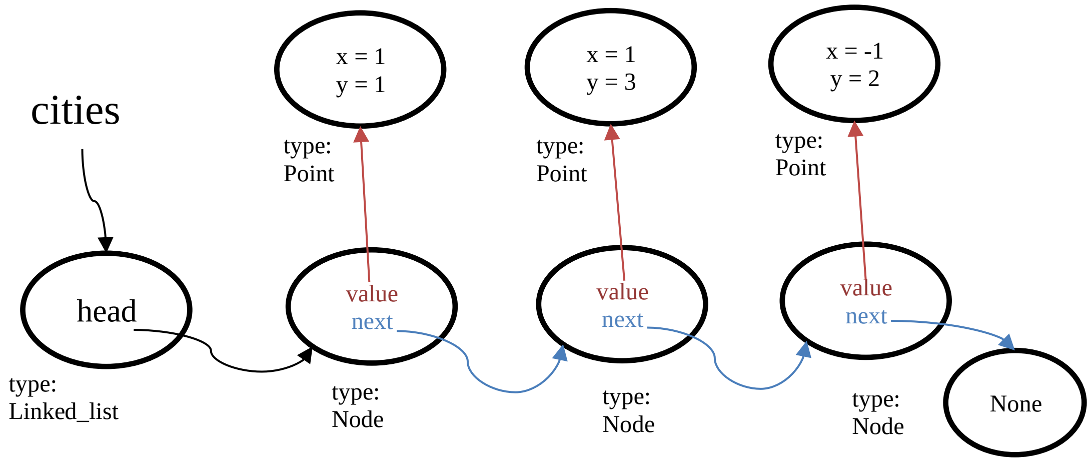
השתכנעו שבדוגמה זו יש זווית **אחת ויחידה** שעונה על השאלה. כאמור, בדוגמאות אחרות יתכן כי יותר מזווית אחת מתאימה (בפרט, יתכנו אינסוף זוויות מתאימות).

ב. לקראת נסיעה לחו״ל, אלחנן תכנן מסלול העובר ב-n ערים שונות. על מנת שלא ירדם על ההגה, אלחנן רוצה לחלק את המסלול ל-d ימים כך שבכל יום לא ייסע יותר מ-k קילומטרים בסך הכל, וסדר הערים יישאר זהה לאחר החלוקה. כמו כן, כדי למזער עלויות אלחנן רוצה למצוא את מספר ימי הטיול d המינימלי שעבורו התנאי יתקיים.

<u>נמדל את הבעיה באופן הבא</u>: נייצג את המפה של המדינה בה מבקר אלחנן ע״י המישור x-y, ונייצג את הערים בהן מבקר אלחנן ע״י נקודות במישור. המרחק (בקילומטרים) בין כל זוג ערים יוגדר להיות המרחק האוקלידי בין זוג הנקודות המייצגות את הערים. לדוגמה, המרחק בין הערים המיוצגות ע״י הנקודות $(1,3)$ ו-$(-1,2)$ הוא:

$$\sqrt{(-1-1)^2+(2-3)^2}=\sqrt{5}\,km$$

המסלול ייוצג על ידי **רשימה מקושרת** cities של נקודות במישור. ראש הרשימה יצביע על Node אשר שדה ה-value שלו יצביע על נקודת ההתחלה (אובייקט מטיפוס Point). השדה next יצביע על ה-Node שערכו יצביע על הנקודה הבאה במסלול, וכך הלאה. לדוגמה, מסלול המתחיל בנקודה $(1,1)$, ממשיך אל הנקודה $(1,3)$ ומסתיים בנקודה $(-1,2)$ ייוצג על ידי הרשימה המקושרת הבאה:



.i ממשו את הפונקציה split(self, k) במחלקה Linked_list אשר פועלת על הרשימה המקושרת self ומפצלת אותה לשתי רשימות מקושרות **חדשות** – הרשימה הראשונה תכיל את k האיברים הראשונים של self והרשימה השנייה תכיל את יתר האיברים, ללא שינוי בסדר האיברים המקורי. במהלך הפעולה של split **אין** לייצר אובייקטים חדשים מטיפוס Node. הפונקציה תחזיר tuple של שתי הרשימות המקושרות החדשות. הניחו כי $1 \le k \le len(self)-1$.
שימו לב שמתודה זו **הרסנית** כלפי הרשימה self. לאחר ביצוע הפעולה הרשימה self תשתנה.
<u>דוגמאות הרצה</u> (כתובות הזיכרון שמודפסות משתנות בין הרצה להרצה ובין מחשבים שונים):

```
>>> lst = Linked_list("abcde")
>>> lst1, lst2 = lst.split(2)
```

```
>>> lst1
[(a, next: 2073168845312), (b, next: 140703363964120)]
>>> lst2
[(c, next: 2073168845504),
 (d, next: 2073168845648),
 (e, next: 140703363964120)]
>>> lst1.len
2
>>> lst2.len
3
>>> lst       # lst is now changed
[(a, next: 2073168845312), (b, next: 140703363964120)]
```

ii. ממשו את הפונקציה divide_route(cities, k) אשר מקבלת את רשימת הערים cities כמתואר לעיל, ואת k כמות הקילומטרים המותרת לנסוע ביום, ומחזירה **רשימה לא מקושרת** (כלומר רשימת פייתון רגילה מסוג list) באורך d אשר באינדקס ה-i של הרשימה נמצאת **רשימה מקושרת** המייצגת את המסלול ביום ה-i (היום הראשון מיוצג באינדקס 0), כך שאורך כל מסלול הוא לכל היותר k ומספר הימים d הוא מינימלי. תיתכן יותר מחלוקה אחת מתאימה – החזירו חלוקה חוקית כלשהי.

על הפונקציה לרוץ בסיבוכיות זמן $O(n)$ ובסיבוכיות מקום $O(d)$.

<u>הנחיות והנחות</u>:

- **אסור** לפונקציה לייצר אובייקטים חדשים מטיפוס Node או מטיפוס Point. כמו כן, **אסור** לשנות ישירות את השדות הפנימיים של האובייקטים Node, Point, Linked_list (מותר לגשת אליהם).
- המחלקות Linked_list ו-Node שראיתם בכיתה נתונות לכם בקובץ השלד – אין לשנות אותן פרט למתודה split שמימשתם בסעיף הראשון. מומלץ להיעזר ב-split בפתרון שלכם.
- בתוך קובץ השלד, במחלקה Point, נתונה לכם המתודה distance(self, other) אשר מקבלת שתי נקודות ומחזירה את המרחק האוקלידי ביניהן. ניתן להיעזר במתודה בפתרון שלכם.
- הניחו כי המרחק בין כל זוג ערים עוקבות במסלול קטן מ-k.

<u>דוגמת הרצה</u> (מומלץ לשרטט את הנקודות ולחשב את המרחקים בכדי להבין את פשר החלוקה):

```
>>> cities = Linked_list([Point(0,1), Point(0,0), Point(3,3),
Point(-2,3), Point(-2,-5), Point(-4,-5)])
>>> divide_route(cities, 10)        # ______________________
[[((0,1), next: 2687346155040),      #
  ((0,0), next: 2687346155136),      # 1st day of the trip
  ((3,3), next: 140703363964120)],   # ______________________
 [((-2,3), next: 140703363964120)],  # 2nd day of the trip
 [((-2,-5), next: 2687346155424),    # ______________________
  ((-4,-5), next: 140703363964120)]] # 3rd day of the trip
```

<u>שאלה 4</u>

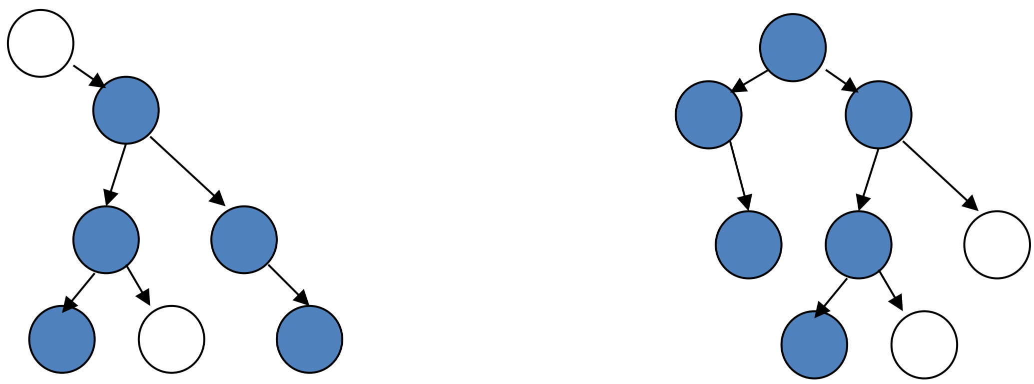
השאלה עוסקת בעצים בינאריים, ובמחלקה Binary_search_tree. הניחו בשאלה זו שהמפתחות (השדה key) בצמתים הינם מחרוזות והערכים (השדה val) בצמתים הינם מספרים מטיפוס int גדולים ממש מ-0 וקטנים או שווים ל- 100. שימו לב שבכל הדוגמאות בסעיפים הבאים המפתחות הינם מטיפוס str.

א. **קוטר של עץ** מוגדר להיות אורכו של מסלול ארוך ביותר בין שני צמתים בעץ, כאשר צומת לא מופיע במסלול יותר מפעם אחת, ואין חשיבות לכיוון הקשתות (ראו דוגמה בהמשך). אורך המסלול במקרה זה נקבע לפי מספר הצמתים במסלול. שימו לב שייתכנו מספר מסלולים שונים שאורכם הוא קוטר העץ. ממשו את המתודה diam(self) שמחזירה את קוטר העץ. עבור עץ ריק המתודה תחזיר 0.

<u>הנחיות</u>:

1. על המתודה לרוץ בסיבוכיות זמן $O(n)$ כאשר n מספר הצמתים בעץ.
2. אין להוסיף שדות ל- Tree_node.

להלן שתי דוגמאות בהן מודגש בכחול מסלול שאורכו הינו הקוטר:

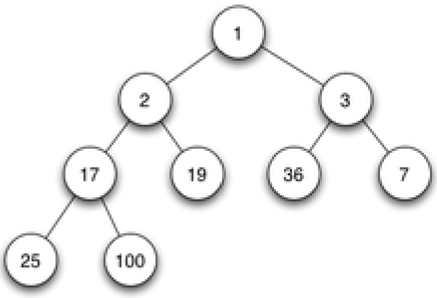


דוגמאות הרצה:

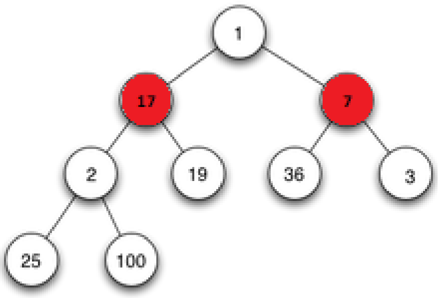
```
>>> t2 = Binary_search_tree()
>>> t2.insert('c', 10)
>>> t2.insert('a', 10)
>>> t2.insert('b', 10)
>>> t2.insert('g', 10)
>>> t2.insert('e', 10)
>>> t2.insert('d', 10)
>>> t2.insert('f', 10)
>>> t2.insert('h', 10)
>>> t2.diam()
6
>>> t3 = Binary_search_tree()
>>> t3.insert('c', 1)
```

```
>>> t3.insert('g', 3)
>>> t3.insert('e', 5)
>>> t3.insert('d', 7)
>>> t3.insert('f', 8)
>>> t3.insert('h', 6)
>>> t3.insert('z', 6)
>>> t3.diam()
5
```

ב. "ערימת מינימום" היא עץ בינארי "כמעט מושלם", כלומר כזה שכל השורות, פרט אולי לאחרונה, מלאות, והאחרונה מלאה משמאל עד לנקודה מסוימת. בנוסף, היא מקיימת: כל **ערך (value)** של צומת אינו גדול משני בניו (שימו לב שהבדיקה היא על הערך שנמצא בצומת ולא על המפתח!).
דוגמה לערימת מינימום בינארית (שימו לב שהמספר שרשום על כל צומת הוא הערך ולא המפתח כפי שראיתם בדוגמאות בכיתה):



לא ערימת מינימום:



.i בקובץ השלד ממשו את המתודה is_min_heap של Binary_search_tree שבודקת האם כל ה-values של הצמתים שלו (ולא ה keys) מקיימים את תכונת "ערימת המינימום". שימו לב שרק המפתחות רלוונטיים לסידור המבנה כעץ חיפוש, ורק ה- values רלוונטיים לסידור המבנה כערימת מינימום.

<u>הנחיות והנחות</u>:

- על הפתרון להיות <u>רקורסיבי</u>.
- אפשר להניח שהקלט תקין, כלומר עץ בינארי כמעט מושלם.
- **אסור** להשתמש בלולאות for / while.
- הפונקציה תחזיר True אם העץ הוא ערימת מינימום ביחס ל- values שלו ו False אחרת.

ii. ציינו בקובץ ה pdf מהי סיבוכיות זמן הריצה עבור עץ בגודל n. הסבירו את ניתוח הסיבוכיות ונמקו את תשובתכם.

<u>הנחיות והבהרות לכלל השאלה</u>:

- בכל מקום בו נדרש לממש מתודה באופן רקורסיבי, מותר להגדיר פונקציית עזר רקורסיבית שתיקרא על ידי המתודה (שבמקרה זה תהיה "מתודת מעטפת").

  רצוי ואף מומלץ לממש את פונקציית העזר הרקורסיבית כפונקציה פנימית של המתודה. לדוגמה:

```
def some_method(self):
    def helper_rec( <some parameters> ):
        # some code
    return helper_rec( <some arguments> )
```

- המחלקה Tree_node שראיתם בכיתה מופיעה בקובץ השלד. כאמור, אין לשנות מחלקה זו.

שאלה 5

בחלק זה נדון בסיבוכיות של פעולות חיבור, כפל והעלאה בחזקה של מספרים בכתיב בינארי.

חיבור וכפל של מספרים בינאריים יכול להתבצע בצורה דומה מאוד לזו של מספרים עשרוניים. להלן המחשה של אלגוריתם החיבור והכפל של שני מספרים בינאריים A,B:

```
 1 1 1 1     (carried digits)

     1 1 1 1 (A)
+  1 1 0 1 0 (B)
-------------
= 1 0 1 0 0 1
```

להלן המחשה של אלגוריתם ההכפלה AxB:

```
          1 0 1    (A)
      × 1 0 1 0    (B)
      ---------
          0 0 0    ← Corresponds to a zero in B
+       1 0 1      ← Corresponds to a one in B
+     0 0 0
+   1 0 1
---------------
=   1 1 0 0 1 0
```

בדוגמה הנ"ל יש 19 פעולות. 12 עבור הכפל והשאר עבור החיבור.

בהרצאה נלמד אלגוריתם iterated squaring להעלאה בחזקה של מספרים שלמים חיוביים (ללא מודולו). הקוד מופיע בהמשך. הניחו כי מספר הביטים בייצוג הבינארי של a הינו n ומספר הביטים בייצוג הבינארי של b הינו m.

עליכם לנתח את סיבוכיות זמן הריצה של הפעולה a^b בשיטה זו כתלות ב-m ו/או n ולתת חסם הדוק ככל האפשר במונחי $O(\cdot)$. שימו לב שהמספרים המוכפלים גדלים עם התקדמות האלגוריתם, ויש לקחת זאת בחשבון. כמו כן, בניתוח נתייחס רק לפעולות הכפל המופיעות באלגוריתם. אמנם באלגוריתם ישנן פעולות נוספות מלבד כפל. אך לפעולות אלו סיבוכיות זמן מסדר גודל זניח לעומת פעולות הכפל. למשל, ההשוואה b>0 רצה בזמן $O(1)$ (לא נכנסנו לעומק של ייצוג מספרים שלמים שליליים, אבל ראינו ברפרוף את שיטת המשלים ל- 2 ושם ברור שמספיק לקרוא את הביט השמאלי כדי להכריע האם מספר שלם הוא חיובי או שלילי). הפעולה b%2 דורשת בדיקת הביט הימני של b בלבד ולכן רצה בזמן $O(1)$. פעולת החילוק b//2 כוללת העתקה של b ללא הביט הימני ביותר למקום חדש בזיכרון, פעולה שגם כן רצה בזמן ליניארי בגודל של b. על ההסבר להיות תמציתי ומדויק. מומלץ לחשוב על המקרה ה"גרוע" מבחינת מספר פעולות הכפל ולתאר כיצד הגעתם לחסם עבור פעולות אלו.

```
def power(a,b):
    """ computes a**b using iterated squaring
        assumes b is nonnegative integer """
    result = 1
    while b>0: # b is nonzero
        if b%2 == 1: # b is odd
            result = result*a
        a = a*a
        b = b//2
    return result
```

סוף