אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2020

תרגיל בית מספר 6 - להגשה עד 28.6.20 בשעה 23:55

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

הגשה:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton6.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 hw6_012345678.pdf ו- hw6_012345678.py.

- 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.

<u>שאלה 1 – גנרטורים</u>

בשאלה זו נבנה מחלקה בשם ImprovedGenerator שתהווה, כשמה, שדרוג למחלקת הגנרטורים שפגשנו בהרצאה.

כזכור, בהנתן גנרטור g הפקודה x = next(g) מציבה ב-x את האיבר הנוכחי בגנרטור ומתקדמת לאיבר הבא (או לסופו, אם האיבר הנוכחי הוא האחרון אותו הגנרטור מחזיר). לפעולה כזו נקרא מיצוי של x. אם ישנם איברים ב-g אשר לא מוצו נאמר כי g לא מוצה.

בהנתן גנרטור g, האובייקט imp_g = ImprovedGenerator(g) יאפשר לנו למצות את איברי הגנרטור g באמצעות פקודת next או איטרציה בלולאה (כמו גנרטור רגיל). בנוסף, האובייקט יתמוך בפעולות נוספות אותם נממש לאורך השאלה. לאובייקט imp_g נקרא גנרטור משופר.

הערות והנחיות כלליות לשאלה:

- ניתן להניח כי כל גנרטור g לא ינתן כקלט יותר מפעם אחת.
  בפרט, האובייקט ImprovedGenerator(g) יכול להריץ מתודות על g ולשנות את תוכנו ללא הגבלה.
- **<u>אין</u>** להניח כי הגנרטורים בשאלה סופיים.
- המתודה המיוחדת __iter__ מומשה עבורכם. אין לשנות את מימוש מתודה זו ואין לקרוא לה במהלך מימוש המתודות שבשאלה.
  להעשרה בלבד: בסעיף ב' נממש את מתודה __next__. כאשר מתודה זו ממומשת פייתון מאפשר לנו לבצע איטרציה על אובייקטים על ידי קריאה לה ולכן המימוש של מתודה __iter__ נראה כאילו "אינו עושה דבר".
- ניתן, ואף כדאי, להוסיף מתודות עזר למחלקה אשר אינן חלק מהדרישות של השאלה.

<u>סעיף א'</u>

ממשו את המתודה __init__. המתודה תקבל כקלט גנרטור g ותחזיר אובייקט מהמחלקה ImprovedGenerator.

הנחיות טכניות:

- באפשרותכם להגדיר שדות לאובייקט ללא הגבלה.

<u>סעיף ב'</u>

ממשו את המתודה __next__. המתודה תקבל אובייקט מהמחלקה ImprovedGenerator. המתודה תמצה את האיבר הבא של הגנרטור המשופר ותחזיר אותו כפלט. אם הגנרטור מוצה, המתודה תרים exception מסוג StopIteration.

מבנה הפונקציה (כפסאודו-קוד) צריך להיות כלהלן:

```
def __next__(self):
  if self is not empty:
    return next element in self
  else:
    raise StopIteration
```

בפרט, שימו לב:

- בניגוד לפונקצית גנרטור שמחזירה ערכים באמצעות המילה השמורה yield, את הפלט של המתודה יש להחזיר באמצעות המילה השמורה return.
- אם אין איבר נוסף להחזיר (כלומר, אם הגנרטור מוצה) יש להעלות שגיאה באמצעות המילה השמורה raise ולאחריה שגיאת StopIteration.

דוגמאות הרצה:

```
>>> g = (i for i in range(5))
>>> g2 = ImprovedGenerator(g)
>>> for i in range(6):
        print(next(g2))


0
1
2
3
4
Traceback (most recent call last):
  File "<pyshell#36>", line 2, in <module>
    print(next(g2))
  File "C:\...", line 30, in __next__
    raise StopIteration
StopIteration
```

<u>סעיף ג'</u>

ממשו את המתודה has_next. המתודה תקבל אובייקט מהמחלקה ImprovedGenerator ותחזיר True אם"ם ישנם איברים בגנרטור המשופר אשר לא מוצו.

דוגמאות הרצה:

```
>>> g = (i for i in range(5))
>>> g2 = ImprovedGenerator(g)
>>> g2.has_next()
True
>>> for i in range(4):
        print(next(g2))


0
1
2
3
>>> g2.has_next()
True
>>> print(next(g2))
4
>>> g2.has_next()
False
```

<u>סעיף ד'</u>

ממשו את המתודה peek. המתודה תקבל אובייקט מהמחלקה ImprovedGenerator אשר לא מוצה ותחזיר את האיבר הבא בגנרטור המשופר מבלי למצות אותו.

הנחיות: ניתן להניח כי הגנרטור לא מוצה.

דוגמאות הרצה:

```
>>> g = (i for i in range(5))
```

```
>>> g2 = ImprovedGenerator(g)
>>> next(g2)
0
>>> g2.peek()
1
>>> g2.peek()
1
>>> g2.peek()
1
>>> next(g2)
1
>>> next(g2)
2
>>> g2.peek()
3
>>> next(g2)
3
```

<u>סעיף ה'</u>

ממשו את המתודה product. המתודה תקבל שני אובייקטים מהמחלקה ImprovedGenerator ותחזיר אובייקט מהמחלקה ImprovedGenerator המהווה את מכפלת הגנרטורים המשופרים שבקלט.

כלומר, הפקודה g3 = g1.product(g2) תחזיר אובייקט מטיפוס ImprovedGenerator בו כל איבר הוא הזוג הבא של הגנרטורים המשופרים g1, g2.

הנחיות: ניתן להניח כי מספר האיברים שנותרו ב-g1 וב-g2 זהה (או ששניהם אינסופיים).

דוגמאות הרצה:

```
>>> g = (i for i in range(10))
>>> g2 = (i for i in range(10,20))
>>> X = ImprovedGenerator(g)
>>> Y = ImprovedGenerator(g2)
>>> Z = X.product(Y)
>>> Z.has_next()
True
>>> next(Z)
(0, 10)
>>> next(Z)
(1, 11)
>>> Z.peek()
(2, 12)
>>> Z.has_next()
True
>>> next(Z)
(2, 12)
```

<u>שאלה 2 – קוד האפמן</u>

א. מצאו את קוד האפמן האופטימלי עבור הקורפוס (corpus) הבא:
a: 1 b: 1 c: 2 d: 3 e: 5 f: 8 g: 13 h: 21

סדרת התדירויות הנ"ל מבוססת על 8 מספרי פיבונאצ'י הראשונים.

ב. הכלילו את תשובתכם מסעיף א' למציאת קוד האפמן אופטימלי כאשר התדירויות הן n מספרי פיבונאצ'י הראשונים. נמקו בקצרה, ללא צורך בהוכחה מפורטת, מדוע הכללה זו נכונה.

ג. נתון קובץ שמכיל תווים מתוך אלפבית בן 256 תווים. בנוסף נתון קורפוס עם תדירויות: $a_1 < a_2 < \cdots <$ a_n (כאשר $n = 256$) ומתקיים: $a_n < 2a_1$.

תהי a_1 תדירות התו p (התדירות המינימלית), ותהי a_n תדירות התו q (התדירות המקסימלית).
יהיו $C(a_1)$, $C(a_n)$ קודי ההאפמן שמתקבלים עבור התווים q, p בהתאמה.

מהו ההפרש בין $|C(a_1)|$ (מספר הביטים שדרושים כדי לקודד את התו p) לבין $|C(a_n)|$ (מספר הביטים שדרושים כדי לקודד את התו q)?

על תשובתכם להיות מנומקת!

<u>שאלה 3 – למפל זיו</u>

בשאלה זו נעסוק באלגוריתם למפל זיו כפי שנלמד בכיתה.

אחת הסיבות ליעילות הנמוכה של המימוש אותו למדנו היא הלולאה ב-maxmatch אשר בה המשתנה m רץ בין 1 ל-w. נממש עכשיו אלגוריתם יעיל יותר החוסך מבחינה פרקטית זמן אבל גוזל יותר זיכרון. הרעיון הוא לשמור טבלת hash הממפה שלשות של תווים לרשימה המכילה את **כל האינדקסים** בהם מתחילה מחרוזת עם שלשות תווים אלו בכדי שניתן יהיה לקצר את זמן הריצה. למשל עבור המחרוזת "abcabc" הטבלה תיראה כך:

{"abc": [0, 3], "bca": [1], "cab": [2]}

א. בשלד התרגיל עליכם להשלים את שתי הפונקציות maxmatch ו-LZW_compress בשורות המיועדות לכך בלבד. מומלץ לממש בצורה דומה למימוש המקורי של הפונקציות כפי שלמדנו בכיתה (כלומר, מספר השורות שיש לשנות או להוסיף הינו מינימלי). כמו כן, ניתן ורצוי להשתמש בפונקציית העזר add_triple_to_dict.

ב. מהי סיבוכיות המקום עבור triple_dict כפונקציה של אורך הטקסט N? שימו לב שאורך הטקסט בתווים N איננו חסום. עבור סעיף זה, יש להניח כי בכדי לשמור אינדקס של מחרוזת בגודל N יש צורך בlogN ביטים. הסבירו בקצרה את תשובתכם.

ג. נניח שמספר המופעים של כל שלשת תווים חסום על ידי קבוע c בתוך חלון שגודלו w. מהי סיבוכיות הזמן **הממוצעת** של maxmatch כפונקצייה של גודל החלון w ו/או אורך ההתאמה המקסימלית max_length?

<u>שאלה 4 – תיקון שגיאות</u>

להלן שיטה לאיתור שגיאות (הנקראת שיטת Berger): בהינתן 7 ביטים של אינפורמציה, נוסיף להם 3 ביטים שייצגו את מספר האפסים ב-7 הביטים הללו, בכתיב בינארי. דוגמאות:

כדי לשלוח את 1001100, נוסיף 100, משום שיש ארבעה אפסים. בסה"כ נשלח אם כן – 1001100<u>100</u>. כדי לשלוח את 1111111, נוסיף 000, משום שיש אפס אפסים. בסה"כ נשלח אם כן 1111111<u>000</u>.

יש לנמק בקצרה בכל אחד מהסעיפים הבאים.

א. מהו מרחק Hamming של הקוד?

ב. i. מהו מספר השגיאות שניתן לגלות?

ii. מהו מספר השגיאות שניתן לתקן?

ג. נניח שנפל מספר לא ידוע של שגיאות, אבל ידוע שהשגיאות כללו אך ורק אפסים (0) שהפכו לאחדים (1). איזו מהטענות הבאות נכונה? הצדיקו תשובתכם ע"י הסבר קצר או דוגמא קצרה.

a. במצב כזה לא נוכל לגלות אפילו שגיאה אחת.

b. במצב כזה נוכל לגלות שגיאה בודדת, אך לא יותר מכך.

c. במצב כזה נוכל לגלות עד 2 שגיאות, אך לא יותר מכך.

d. במצב כזה נוכל תמיד לגלות כל כמות של שגיאות.

<u>שאלה 5 – גנרטורים אינסופיים</u>

בשאלה זו נדון בפעולות על גנרטורים אינסופיים. בכל סעיף נתאר פעולה, ועליכם לענות האם ניתן לכתוב פונקציית גנרטור המממשת פעולה זו.

כזכור, פונקצית גנרטור (אינסופי) היא תקינה אם כל קריאה ל- next מחזירה בזמן סופי איבר נוסף או StopIteration Exception. אנו אומרים שפונקציית גנרטור "מייצרת" קבוצה אם כל איבר מהקבוצה מיוצר על ידי מספר סופי של קריאות ל- next. סדר ייצור האיברים אינו חשוב בשאלה זו.

בשאלה זו נניח שלפונקציה מותר להשתמש בזכרון בלתי מוגבל, כולל משתנים גלובליים אם נדרש. כמות הזכרון היא סופית עבור מספר סופי של קריאות next.

עליכם לנמק בקצרה: אם תשובתכם היא שניתן, הסבירו במילים את רעיון המימוש. אם תשובתכם היא שלא ניתן, הסבירו במילים מדוע לא. אין צורך במימוש בקוד במקרה הראשון או בהוכחה פורמלית במקרה השני, אך יש להסביר בפירוט את תשובתכם.

א. בהינתן שני גנרטורים אינסופיים המייצגים קבוצות של מספרים טבעיים (כלומר אין חזרות בכל גנרטור), הפעולה מחזירה גנרטור שמייצר את איחוד הקבוצות: כלומר הגנרטור המוחזר צריך לייצר כל מספר המיוצר על ידי לפחות אחד מהגנרטורים בקלט, אך ללא חזרות.

ב. בהינתן שני גנרטורים אינסופיים המייצגים קבוצות של מספרים טבעיים (כלומר אין חזרות בכל גנרטור), הפעולה מחזירה גנרטור שמייצר את חיתוך הקבוצות: כלומר הגנרטור המוחזר צריך לייצר כל מספר המיוצר על ידי שני הגנרטורים בקלט ,וללא חזרות.

ג. הפעולה (ללא קלט) מחזירה גנרטור שמייצר את כל המספרים הראשוניים הגדולים מ 100 ,ללא חזרות.

<u>שאלה 6 – קארפ-רבין</u>

מחרוזת t תיקרא סיבוב מעגלי של מחרוזת s אם שתיהן באותו אורך ו-t מורכבת מהסיפא של s המשורשר לרישא של s המסתיימת בתחילת סיפא זו. למשל, המחרוזת tocsintro היא סיבוב מעגלי של המחרוזת introtocs וכך גם המחרוזת sintrotoc.

נתונות שתי הפונקציות הבאות המופיעות גם בקובץ השלד (הן זהות לאלו מהשיעור):

```
def fingerprint(text, basis=2**16, r=2**32-3):
        """ used to compute karp-rabin fingerprint of the pattern
            employs Horner method (modulo r) """
        partial_sum = 0
        for ch in text:
                partial_sum =(partial_sum*basis + ord(ch)) % r
        return partial_sum

def text_fingerprint(text, m, basis=2**16, r=2**32-3):
        """ computes karp-rabin fingerprint of the text """
        f=[]
        b_power = pow(basis, m-1, r)
        list.append(f, fingerprint(text[0:m], basis, r))
        # f[0] equals first text fingerprint
        for s in range(1, len(text)-m+1):
                new_fingerprint = ( (f[s-1] - ord(text[s-1])*b_power)*basis
                        +ord(text[s+m-1]) ) % r
                # compute f[s], based on f[s-1]
                list.append(f,new_fingerprint)# append f[s] to existing f
        return f
```

בשאלה זו **<u>ניתן להתעלם</u>** ממצבי Positives-False נדירים – כלומר, מצבים בסבירות נמוכה בהם טביעות האצבע המחושבות לשתי תתי-מחרוזות הינן זהות על אף ששתי תתי-המחרוזות אינן זהות.

a. ממשו בקובץ השלד את הפונקציה is_rotated_1(s, t, basis=2**16, r=2**32-3) שתקבל שתי מחרוזות s, t ותחזיר True אם t היא סיבוב מעגלי של s ואחרת תחזיר False.

<u>דרישות והערות:</u>

1. הפונקציה צריכה להשתמש ב-O(1) זיכרון עזר. כלומר, אסור לפונקציה להקצות זיכרון בגודל שהוא לינארי באורכי מחרוזות הטקסט או יותר מכך. בפרט, אסור לפונקציה בשום שלב לפרוש את רצף ה-fingerprints המלא של אחת ממחרוזות הטקסט. לצורך שאלה זו, ניתן להחשיב אינדקסים (מספר שלם) בתור O(1) זיכרון.
2. ניתן להיעזר בפונקציה fingerprint הנתונה למעלה.

דוגמאות הרצה:

```
>>> is_rotated_1("amirrub", "rubamir")
True
>>> is_rotated_1("amirrub", "amirgil")
False
```

b. ממשו בקובץ השלד את הפונקציה is_rotated_2(s, t) שתקבל שתי מחרוזות s, t ותחזיר True אם t היא סיבוב מעגלי של s ואחרת תחזיר False.

<u>דרישות והערות:</u>

1. עליכם להשתמש בפונקציה text_fingerprint הנתונה למעלה.

אין מגבלה על סיבוכיות הזיכרון.

<u>שאלה 7 – מטריצות ותמונות (בונוס)</u>

בשאלה זו, ניצור מטריצות ונבנה תמונות ע״י ייצוגן כמטריצות, יש לשים לב כי כלל הפלטים בשאלה זו צריכים להיות אובייקטים מהמחלקה Matrix שנלמדה בשיעור על תמונות. בנוסף, בכדי לעבוד על שאלה זו, עליכם לדאוג שבתיקיה בה נמצא קובץ הקוד שלכם נמצא גם הקובץ matrix.py.

את כלל התמונות אשר אתם מייצרים בסעיף זה יש להכניס לPDF אותו אתם מגישים.

א. ניזכר בהגדרה של מטריצת Hadamard אשר הופיעה בתרגיל 4:

מטריצת Hadamard מסדר n (אותה נסמן מעתה ואילך על ידי had(n)) הינה מטריצה **בינארית** ריבועית המוגדרת באופן הרקורסיבי הבא:

- עבור n = 0, had(0) היא מטריצה מגודל 1×1 המכילה את האיבר אפס: $had(0) = (0)$
- עבור n > 0, had(n) היא מטריצה ריבועית מגודל $2^n \times 2^n$ אשר בנויה באופן הרקורסיבי הבא:

  אם had(n-1) היא מטריצת Hadamard מסדר n-1, אזי:

$$had(n) = \begin{pmatrix} had(n-1) & had(n-1) \\ had(n-1) & \overline{had(n-1)} \end{pmatrix}$$

השלימו בקובץ השלד את הפונקציה had אשר מקבלת כקלט מספר טבעי n ומחזירה אובייקט מסוג מטריצה, אשר מתאימה למטריצת Hadamard מסדר n.

השתמשו בפונקציה אשר כתבתם כדי לייצר מטריצת Hadamrd מסדר 8, ושמרו אותה כתמונה.

ב. נגדיר סוג מטריצות מעניין נוסף – מטריצת Disjointness:

מטריצת Disjointness מסדר n (אותה נסמן מעתה ואילך על ידי disj(n)) הינה מטריצה **בוליאנית** מגודל $2^n \times 2^n$ ומוגדרת כך:

בהנתן וקטור בוליאני v, נגדיר את הפונקציה $S(v)$ אשר מייצגת תת קבוצה של $\{0, \dots, n-1\}$ באופן הבא

$$i \in S(v) \iff v[i] = 1$$

לכל תא במטריצה, יהי r השורה של התא, וc העמודה של התא ויהיו br ו bc הייצוגים הבוליאנים שלהם בהתאמה.

בתא יהיה 1 אם $S(br) \cap S(bc) = \emptyset$, אחרת בתא יהיה 0.

השלימו בקובץ השלד את הפונקציה disj אשר מקבלת כקלט מספר טבעי n ומחזירה אובייקט מסוג מטריצה, אשר מתאימה למטריצת Disjointness מסדר n.

השתמשו בפונקציה אשר כתבתם כדי לייצר מטריצת Disjointness מסדר 8, ושמרו אותה כתמונה.

ג. בסעיף זה נרצה לייצר תמונה ע״י שילוב של תמונות קיימות. בקבצי התרגיל נתונות 10 תמונות של כלל הספרות (0 עד 9), התמונות הינן בגודל 20×20 ושמורות כמטריצות. עליכם לייצר תמונה (אשר תיוצג על ידי מטריצה) בגודל 30×300 , השתמשו בתמונות הקיימות של הספרות, והכניסו אותם לתמונה שאתם מייצרים, כך שבתוך התמונה תהיה כתובה תעודת הזהות שלכם. כלל הפיקסלים (הכניסות) אשר לא חלק מהספרות של תעודת הזהות שלכם צריכים להיות בצבע לבן.

בכדי לקרוא את התמונות, נתונה לכם הפונקציה load כחלק מהמחלקה Matrix.

את הקוד יש להשלים בפונקציה id_image אשר לא מקבלת קלט ומחזירה אובייקט מסוג מטריצה אשר מייצג את התמונה אשר ייצרתם, שמרו את התמונה אשר ייצרתם.

סוף