מס' ת"ז_____ מס' מחברת_____

מבחן בקורס מבוא מורחב למדעי המחשב, CS1001.py
עם תיקונים קלים, בעקבות הבהרות שניתנו בזמן הבחינה

סמסטר ב' 2014

מועד א, 11/7/2014

מרצים: פרופ' בני שור, אמיר רובינשטיין

מתרגלים: מיכל קליינבורט, יעל ברן

משך הבחינה: 3 שעות.

חומר עזר מותר: 2 דפי עזר (דו צדדיים) בגודל A4 כ"א.

- במבחן 14 עמודים – בידקו שכולם בידיכם. בנוסף, בסוף הבחינה ישנו דף נוסף ריק, לשימוש במקרה חירום בלבד.
- יש לכתוב את כל התשובות בטופס הבחינה. המחברת תשמש כטיוטה בלבד ולא תיבדק.
- יש לענות על כל ארבע השאלות.
- בכל השאלות, אלא אם נכתב במפורש אחרת:
  - אם עליכם לכתוב פונקציה, אין צורך לבדוק בה את תקינות הקלט שלה
  - מותר להסתמך על סעיפים קודמים, גם אם לא עניתם עליהם
  - ניתן לצטט טענות שנטענו בהרצאה או בשיעורי התרגול
- אנו ממליצים לא "להיתקע" על אף שאלה, אלא להמשיך לשאלות אחרות ולחזור לשאלה אח"כ.
- בכל סעיף בשאלות הפתוחות ניתן לכתוב "איני יודע/ת" ולא לכתוב שום טקסט נוסף. במקרה זה יינתן 20% מציון הסעיף.
- יש לכתוב את כל התשובות במקום המוקצב ובכתב קריא. חריגות משמעותיות מהמקום המוקצב, או תשובות הכתובות בכתב קטן מדי, לא ייקראו ; תשובות שדורשות מאמצים רבים להבנתן עלולות לגרור הורדת ציון.

טבלת ציונים: (לשימוש הבודקים)

שאלה	ערך	ניקוד
1	35	
2	20	
3	25	
4	20	
סה"כ	100	

בהצלחה !

שאלה 1 (35 נק')

בשאלה זו נגדיר מחלקה חדשה בשם Polynomial שתייצג פולינום במשתנה אחד. להזכירכם, פולינום במשתנה אחד מדרגה n ניתן לכתוב באופן הבא: $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ כאשר $n \geq 0$ מספר טבעי. המקדמים a_i לכל $0 \leq i < n$ הם שלמים אי שליליים (כלומר $a_i \geq 0$). בנוסף, מתקיים $a_n > 0$, למעט בפולינום האפס $p(x) = 0$.

המתודות __init__ ו-__repr__ כבר ממומשות עבורכם. המקדמים של הפולינום נשמרים בשדה **coeffs** שהינו רשימה שבה האיבר במקום ה-i מייצג את המקדם של x^i. לצורך ניתוח סיבוכיות הניחו כי חיבור וכפל של שני מספרים כלשהם רץ בזמן קבוע O(1).

א. (20 נק') השלימו בעמוד הבא את מימוש המתודות הבאות. בכל מתודה מצויינת מהי סיבוכיות זמן הריצה הדרושה - לקבלת ניקוד מלא יש לעמוד בדרישות סיבוכיות אלו. ניתן להניח כי הקלט תקין. היעזרו בדוגמאות ההרצה שמופיעות בהמשך.

- **evaluate** – מקבלת כקלט מספר x (שלם או ממשי) ומחזירה את הערך של הפולינום עבור ה-x הנתון. סיבוכיות זמן דרושה O(n).
  <u>הערה מחייבת</u>: במתודה זו אין להשתמש כלל בפעולת העלאה בחזקה (pow או **).

- **derivative** – מחזירה פולינום <u>חדש</u> שהוא הנגזרת של הפולינום המקורי. סיבוכיות זמן דרושה O(n). פעולת הנגזרת של פולינום מוגדרת כדלקמן:

$$derivative(a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0) = n \cdot a_n x^{n-1} + (n-1) \cdot a_{n-1}x^{n-2} + \cdots + a_1$$

- **__add__** – מקבלת פולינום נוסף כארגומנט, ומחזירה פולינום <u>חדש</u> שמייצג את סכום שני הפולינומים. אם דרגת הפולינום הקיים (self) היא n ודרגת הפולינום הנוסף היא m אז סיבוכיות הזמן הדרושה היא O(m+n).

- **__mul__** – מקבלת פולינום נוסף כארגומנט ומחזירה פולינום <u>חדש</u> שמייצג את מכפלת שני הפולינומים. אם דרגת הפולינום הקיים (self) היא n ודרגת הפולינום הנוסף היא m אז סיבוכיות הזמן הדרושה היא O(mn).

דוגמת הרצה (המשך בעמוד הבא):

```
>>> q = Polynomial([0, 0, 0, 6])
>>> q
(6*x**3)
>>> p = Polynomial([3, 4, 1])
>>> p
(3*x**0) + (4*x**1) + (1*x**2)
>>> p.evaluate(10)
143
```

```
>>> dp = p.derivative()
>>> dp
(4*x**0) + (2*x**1)
>>> ddp = p.derivative().derivative()
>>> ddp
(2*x**0)
>>> r = p + q
>>> r
(3*x**0) + (4*x**1) + (1*x**2) + (6*x**3)
>>> w = Polynomial([1, 0 ,10])
>>> z = p*w
>>> z
(3*x**0) + (4*x**1) + (31*x**2) + (40*x**3) + (10*x**4)
```

השלימו את המתודות כנדרש:

```
class Polynomial():
    def __init__(self, coeffs_lst):
        self.coeffs = coeffs_lst

    def __repr__(self):
        terms = ["("+str(self.coeffs[k])+"*x**" + \
                 str(k)+")" \
                 for k in range(len(self.coeffs)) \
                 if self.coeffs[k]!=0]
        return " + ".join(terms)
```

```
    def evaluate(self, x):
       result = 0









       return result
```

```
    def derivative(self):









```

```
    def __add__(self, other):
```

```
    def __mul__(self, other):
```

ב. (5 נק') השלימו את מימוש המתודה find_root ששייכת למחלקה Polynomial. על המתודה להשתמש בשיטת ניוטון רפסון (NR) למציאת קירוב לשורש של פונקציה. חתימת הפונקציה NR, כפי שנלמדה בכיתה, הינה:

```
NR(func, deriv, epsilon=10**(-8), n=100, x0=None)
```

להלן דוגמת הרצה (מהכיתה) של הפונקציה NR:

```
>>> NR(lambda x: x**2+2*x-7, lambda x:2*x+2)
-3.828427124747116
```

```
def find_root(self):
    return NR(_____ , _____ )
```

ג. (10 נק') פולינום **דליל** הינו פולינום שבו מספר המקדמים a_i השונים מ-0 (נסמנו ב- k) קטן מאוד ביחס ל- n ($k \ll n$). לדוגמא: עבור $p(x) = 3x^{1000000} + 20x$ מתקיים k=2, n=1000000.

הציעו ייצוג חלופי למחלקה Polynomial, שיאפשר מימוש המתודה derivative בסיבוכיות זמן O(k). כלומר עבור פולינום דליל המתודה תהיה יעילה יותר אסימפטוטית מאשר בסעיף א'. האתחול ב- __init__ ירוץ ב- O(n) זמן. השלימו את המימושים החלופיים ל- __init__ ול- derivative. הקלט שמקבלת __init__ הוא עדיין רשימת מקדמים כמו בסעיף א'.

```
Class Polynomial():
```

```
def __init__(self, coeffs_lst):
```

```
  def derivative(self):
```

שאלה 2 (20 נק')

בשאלה זו נעסוק בחיפוש כל המופעים של תמונה קטנה בתוך תמונה גדולה ממנה. נכנה את התמונה הקטנה "חלון", ולשם פשטות נניח כי החלון ריבועי, כלומר מספר השורות בו שווה למספר העמודות (ויסומן ב- k). התמונה וכן החלון ייוצגו באמצעות המחלקה Matrix שראינו בקורס. כל פיקסל מייצג ערך אפור בין 0 (שחור) ל- 255 (לבן).

פתרון אפשרי מתבסס על הרעיון של אלגוריתם Karp-Rabin, בו השתמשנו על מנת לחפש מחרוזת תבנית בתוך מחרוזת טקסט: מחשבים מעין טביעת אצבע של ה"חלון" הנתון, וכן של כל החלונות בגודל k x k אשר מוכלים בתמונה הגדולה. מדווחים על התאמה היכן שטביעות האצבע שוות.

נממש כעת חלק מהפונקציות הדרושות לפתרון הבעיה.

לשם פשטות ניתוח הסיבוכיות, בכל הסעיפים נניח כי פעולות חיבור וחיסור רצות בזמן קבוע O(1).

נגדיר פונקציה fingerprint, אשר בהינתן מטריצה ריבועית k x k מחזירה מספר המייצג "טביעת אצבע" שלה:

```
def fingerprint(mat):
   assert isinstance(mat,Matrix)
   k,makesure = mat.dim()
   assert k == makesure

   return sum(mat[i,j] for i in range(k) for j in range(k))
```

<u>תזכורת (1)</u>: המתודה dim של המחלקה Matrix מחזירה tuple שהאיבר הראשון בו הוא מספר השורות והשני מספר העמודות.

<u>תזכורת (2)</u>: עבור אובייקט mat של המחלקה Matrix האיבר mat[i,j] מייצג את הפיקסל בשורה ה- i ובעמודה ה- j. בנוסף, הפיקסל באינדקס [0,0] ממוקם בפינה השמאלית-עליונה של התמונה.

לצורך הפתרון נזדקק לפונקציה move_right אשר מקבלת (בסדר זה) תמונה (כלומר אובייקט מסוג Matrix), אינדקסי שורה ועמודה של פיקסל בתוכה, גודל חלון (k) ואת טביעת האצבע של החלון בגודל המתאים, אשר הפינה השמאלית העליונה שלו ממוקמת באינדקסים הללו. הפונקציה מחזירה את טביעת האצבע של החלון אשר מתקבל על ידי הזזת החלון המקורי **ימינה** בפיקסל אחד. ניתן להניח כי החלון מימין אכן קיים (כלומר שלא הגענו לגבול הימני של התמונה).

לדוגמה, לאחר רצף הפקודות

```
fp = fingerprint(mat[0:k,0:k])
right_fp = move_right(mat,0,0,k,fp)
```

מתקיים

```
right_fp == fingerprint(mat[0:k,1:k+1])
```

א. (7 נק') השלימו את מימוש הפונקציה move_right, בסיבוכיות זמן ריצה O(k):

```
def move_right(mat, i, j, k, fp):
```

ב. (8 נק') הניחו כי מומשה עבורכם גם הפונקציה move_down. לפונקציה זו יש חתימה זהה לזו של move_right:

```
move_down(mat, i, j, k, fp)
```

וההבדל בין שתי הפונקציות הוא ש move_down מחזירה את טביעת האצבע של החלון אשר מתקבל על ידי הזזת החלון המקורי **<u>מטה</u>** בפיקסל אחד. הפונקציה מניחה כי החלון שלמטה אכן קיים (כלומר שלא הגענו לגבול התחתון של התמונה).

עתה נממש את הפונקציה image_fingerprints, אשר מקבלת תמונה mat וגודל חלון k. הפונקציה מחזירה את טביעות האצבע של כל החלונות בגודל k x k אשר מוכלים בתמונה. לשם כך יוצרת הפונקציה רשימה של רשימות בגודל המתאים, ומאחסנת באינדקס ה $[i][j]$ את טביעת האצבע של החלון, אשר פינתו השמאלית-עליונה ממוקמת בשורה ה i בעמודה ה j של התמונה mat.

השלימו את מימוש הפונקציה, עבור תמונה בגודל n x m וגודל חלון k, בסיבוכיות זמן O(mnk):

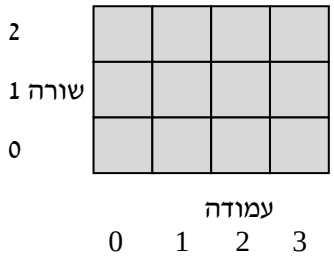
```
def image_fingerprints(mat, k):
   n,m = mat.dim()
   if n<k or m<k:
       return []
   fps = [[0]*(m-k+1) for i in range(n-k+1)]

   first = _____
   fps[0][0] = first
   i = 0
   while True:
       current = first
       for j in range(0,m-k):

           current = _____

           fps[_____][_____] = current
       if (i==n-k):

           _____

       first = _____
       fps[i+1][0] = first
       i += 1
   return fps
```

ג. (5 נק') ציינו את חסרונה העיקרי של הפונקציה fingerprint שהופיעה בתחילת השאלה, ביחס לבעייה אותה אנו מנסים לפתור בשאלה זו. תארו במילים שיפור אפשרי לפונקציה, שיסייע להתגבר על חסרון זה.

שאלה 3 (25 נק')

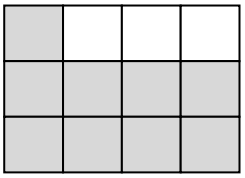
בשיעור (הרצאה 14) ובשיעורי הבית (מטלה 4) הוצג המשחק זלול! (Munch!), משחק לוח לשני שחקנים אשר גורעים (זוללים) בזה אחר זה קוביות מתוך טבלת שוקולד על פי חוקים מוסכמים מראש. בשאלה זו נדון בגרסא "מרוככת" של המשחק. בגרסא זו, כל שחקן חייב לבחור בתורו קוביית שוקולד קיימת (שלא נאכלה עדין) הנמצאת או בשורה התחתונה או בעמודה השמאלית של טבלת השוקולד. זה מבטיח כי הטבלה הנותרת לאחר הצעד היא מלבנית (אין בה "מדרגות"). הקובייה בשורה התחתונה ובעמודה השמאלית היא, כרגיל, מורעלת, והשחקן שאוכל אותה מפסיד.

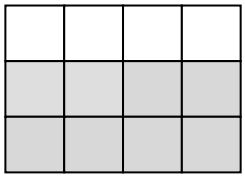
זה מאפשר לייצג קונפיגורציה נתונה באופן קומפקטי ע"י מספר השורות ומספר העמודות. לדוגמה, הייצוג הקומפקטי של מצב המשחק ההתחלתי של טבלה בגודל m=4, n=3 הוא [3,4] וכך נראה הלוח:



[3,4]

במשחק ה"מרוכך" הטבלה מימין, למשל, אינה המשך חוקי של [3,4], אך הטבלה משמאל מהווה המשך חוקי של אותה קונפיגורציה.





בשאלה זו נממש פונקציה win(n,m) אשר מקבלת מצב לוח חוקי ברגע נתון של המשחק (קונפיגורציה, ע"פ המינוח למעלה), ובודקת האם עבור השחקן אשר תורו לשחק כעת קיימת אסטרטגית ניצחון מקונפיגורציה זו (כלומר האם ינצח אם ישחק היטב מכאן והלאה). הפונקציה תחזיר True או False: האם קיימת או לא קיימת אסטרטגית ניצחון לשחקן הנוכחי, בהתאמה. לדוגמה, עבור לוח שנותרה בו רק הקוביה האחרונה, הפונקציה תחזיר False (כי השחקן שתורו לשחק חייב לבלוע את הקוביה המורעלת הזו). מצב הלוח הנוכחי מועבר לפונקציה בייצוג קומפקטי, ע"י זוג הטבעיים החיוביים n,m.

סעיף א (10 נק')

השלימו את הקוד הרקורסיבי הבא לחישוב הפונקציה win(n,m) (יש למלא רק בשורות המסומנות):

```
def win(n,m):
      assert n>0 and m>0
       if n==m==1:
            return False
      for i in range(  ____________________ ):

           if   ____________________ :

                return  ____________________

      for j in range( ____________________ ):

            if  ____________________ :

                 return ____________________

      return  ____________________
```

כעת הסבירו במילים כיצד פועל הקוד שכתבתם:

_____

_____

_____

_____

_____

_____

_____

_____

_____

סעיף ב (10 נק')

ציירו את עץ הרקורסיה המתקבל מהרצת הקוד שהשלמתם בסעיף א', עבור טבלת השוקולד בגודל m=3, n=2 (שורש העץ (2,3) כבר מופיע). בכל צומת, ציינו רק את מספר השורות ומספר העמודות, (n,m). סמנו ע"י W את הקונפיגורציות המנצחות, וע"י L את המפסידות. ציינו גם כמה פעמים מופיעה הקונפיגורציה (1,1) בעץ שלכם.

(2,3)

סעיף ג (5 נק')

יעל טוענת כי עבור כל קונפיגורצית לוח ריבועית (כלומר כזו בה מתקיים m=n) לא קיימת אסטרטגית נצחון לשחקן שתורו לשחק כעת.

מי מהשניים צודק ? או שמא שניהם טועים ? ספקו הסבר תמציתי ומשכנע.

הדרכה: מומלץ ראשית לבדוק את המקרה n=m=3 על מנת לקבל אינטואיציה לבעיה. לניקוד מלא יזכה נימוק אשר תקף למקרה n=m הכללי.

_____

_____

_____

_____

_____

_____

שאלה 4 (20 נק')

השאלה עוסקת בגרסה של קוד אינדקס לאיתור ותיקון שגיאות אותה ראינו בכיתה.

תזכורת: אנו רוצים לשלוח הודעה $\text{msg} = b_1 b_2 \ldots b_k$, כאשר $b_i \in \{0,1\}$ ו- $k = 2^m - 1$ עבור m טבעי כלשהו. נסתכל על הביטים ה"דולקים" ($b_i = 1$, עבור $1 \le i \le k$), ונחשב xor בין האינדקסים שלהם (בכתיב בינארי). נסמן את תוצאת החישוב ב- ec. אם אין בכלל ביטים "דולקים" אז ec=000. נוסיף ל- msg פעמיים את ec (נסמנם ec1 ו- ec2), ונוסיף עוד ביט אחד של זוגיות (parity bit) בסוף (נסמנו p). את השדר כולו נסמן trans.

דוגמה, עבור $m=3$:

```
msg       0110110          (k=2^3-1)
indices   1234567

      2=  010 xor
      3=  011 xor
      5=  101 xor
      6=  110
     ec=  010

trans     0110110 010 010 0
           msg    ec1 ec2 p
```

כזכור, המרחק של הקוד הנ"ל הוא $d=4$.

בכל אחד מהסעיפים א', ב', ג' מתואר מצב בו נפלו 3 שגיאות בשדר (0 שהפך ל- 1 או להיפך). עליכם להחליט האם המצב המתואר אפשרי או לא. אם לדעתכם כן, רישמו בטבלה דוגמה לשדר חוקי מתאים (לפני השגיאות), וסמנו ב- X את מיקומי 3 הביטים בהם נפלו השגיאות. אם לא – רישמו "לא ייתכן" על הטבלה והסבירו. בכל הסעיפים $m=3$.

א. (4 נק') נפלו 3 שגיאות בשדר, ומילת הקוד הקרובה ביותר היא במרחק 1.

trans														
errors (X)														

הסבר קצר, אם המצב לא ייתכן: _____

_____

_____

ב. (4 נק') נפלו 3 שגיאות בשדר, ומילת הקוד הקרובה ביותר היא במרחק 2.

trans														
errors (X)														

הסבר קצר, אם המצב לא ייתכן: _____

_____

_____

ג. (4 נק') נפלו 3 שגיאות בשדר, ומילת הקוד הקרובה ביותר היא במרחק 3.

trans														
errors (X)														

הסבר קצר, אם המצב לא ייתכן: _____

_____

_____

ד. (8 נק') השלימו את הפונקציה decode, המקבלת שדר, שייתכן שנפלו בו שגיאות, בהתאם להנחיות הבאות:

- הניחו כי נפלו לכל היותר 2 שגיאות בשדר.
- אם לא נפלו כלל שגיאות בשדר, תוחזר msg.
- אם נפלה בשדר שגיאה אחת בלבד, תוחזר ההודעה המקורית לאחר תיקון.
- אם נפלו 2 שגיאות, תודפס ההודעה "Error detected" ויוחזר None.

דוגמאות הרצה:

```
>>> decode("01101100100100", 3) #no errors
'0110110'
>>> decode("11101100100100", 3) #1 error
'0110110'
>>> decode("10101100100100", 3) #2 errors
Error detected
```

לנוחיותכם, הניחו כי קיימות שלוש הפונקציות הבאות:

- EC(msg), אשר בהינתן הודעה msg מחזירה את ה- ec שלה כמחרוזת. לדוגמה:

```
>>> EC("0110110")
'010'
```

- xor(bin1, bin2) אשר בהינתן שתי מחרוזות בינאריות מחזירה את תוצאת ה- xor בין הביטים שלהן. לדוגמה:

```
>>> xor("010", "111")
'101'
```

- parity(bin) אשר בהינתן מחרוזת בינארית bin מחזירה את ביט הזוגיות שלה (כמחרוזת). לדוגמה:

```
>>> parity("0110110010010")
'0'
```

```
def decode(trans, m):
    assert len(trans) == 2**m+2*m
    msg = trans[:2**m-1]
    ec1 = trans[_____________________________]
    ec2 = trans[_____________________________]
    p = trans[-1]

    # 0 errors
    if ____________________________________________________:
        return msg

    # 1 error
    elif parity(trans)== '1':

        if ____________________________________________________:
            return msg
        else:
            ec = EC(msg)
            err = ____________________________________________
            location = err-1 #indices in Python start with 0
            return trans[:location] + xor(trans[location], "1") + \
                   trans[location+1:2**m-1]
    #2 errors
    else:
        print("Error detected")
        return None
```

סוף!

דף נוסף למקרה הצורך