מס' ת"ז_____ מס' מחברת_____

מבחן בקורס מבוא מורחב למדעי המחשב, CS1001.py

עם שינויים קלים בהתאם להבהרות / תיקונים במהלך הבחינה

סמסטר א' 2014-5

מועד ב, 13/3/2015

מרצים: פרופ' עמירם יהודאי, דר' דניאל דויטש

מתרגלים: מיכל קליינבורט, אמיר רובינשטיין

משך הבחינה: 3 שעות.

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

- במבחן 12 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 חמש השאלות. מספר הנקודות לא משקף בהכרח את רמת הקושי או את הזמן הנדרש לפתרון השאלה.
- 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.

בהצלחה !

טבלת ציונים: (לשימוש הבודקים)

שאלה	ערך	ניקוד
1	25	
2	20	
3	20	
4	15	
5	20	
סה"כ	100	

מס' ת"ז_____ מס' מחברת____

שאלה 1 (25 נק')

בשאלה זו נפתור את בעיית "תת המחרוזת המשותפת" של שתי מחרוזות.

בהינתן שתי מחרוזות s1 ו- s2, ומספר שלם k, נרצה לקבל תת-מחרוזת כלשהי (רצופה) באורך k בדיוק, שמופיעה בשתי המחרוזות, אם יש כזו. אם יש יותר מאחת, נחזיר אחת מהן שרירותית. למשל, עבור s1 = "abcde" ו- s2 = "acd", עבור k=2 תת המחרוזת 'cd' משותפת לשתי המחרוזות, אבל אין תת מחרוזת משותפת באורך 3.

נסמן n = |s1| ו- m = |s2|. בשאלה זו נניח כי k זניח לעומת אורכי המחרוזות, ולכן ניתן להחשיבו כקבוע לצורכי ניתוח סיבוכיות.

א. להלן פתרון נאיבי לבעיה:

```
def common_substring_naive(s1,s2,k):
    """ find common substring of s1 and s2 of length k """
    for i in range(len(s1)-k+1):
        for j in range(len(s2)-k+1):
            if s1[i:i+k] == s2[j:j+k]:
                return s1[i:i+k]
    return None
```

מהי סיבוכיות זמן הריצה של הפתרון הנ"ל, כתלות ב- n ו- m? תנו תשובה במונחי סדר גודל O(…), הדוקה ככל שתוכלו:

O(_____)

ב. תארו במילים אלגוריתם נוסף לבעיה, שרץ בסיבוכיות זמן O(m+n) בממוצע. הסבירו באיזה מבנה נתונים מוכן (built-in) של פייתון הייתם משתמשים ומדוע. אין לחרוג מהמקום המוקצה, ויש לתאר את האלגוריתם באופן ברור וחד משמעי.

כעת נרצה למצוא תת מחרוזת משותפת <u>ארוכה ביותר</u> של המחרוזות s1 ו- s2.

ג. הוצע הפתרון הבא: לבדוק ערכי k הולכים וגדלים: k=1, 2, 3,..., עד הפעם הראשונה שבה לא נמצאה תת מחרוזת משותפת באורך k.

האם הטענה הבאה נכונה או לא? הסבירו:

אם עבור 'k מסויים נמצאה תת מחרוזת משותפת ל- s1 ול- s2, אבל עבור k'+1 לא נמצאה כזו, אז תת המחרוזת המשותפת הארוכה ביותר של המחרוזות היא באורך 'k.

ד. להלן פתרון אחר למציאת תת מחרוזת משותפת הארוכה ביותר, העושה שימוש במנגנון של חיפוש בינארי על הערכים של k:

```
def longest_common_substring(s1, s2):
    k = 1
    while common_substring_naive(s1,s2,k) != None:
        print(k, "found")
        k *= 2

    print(k, "not found")

    longest = k//2
    low = k//2+1
    high = k-1
    while low <= high:
        mid = (low+high)//2
        res = common_substring_naive(s1,s2,mid)
        if res == None:
            print(mid, "not found")
            high = mid-1
        else:
            print(mid, "found")
            low = mid+1
            longest = mid

    return common_substring_naive(s1,s2,longest)
```

שימו לב כי הפונקציה מדפיסה שורות מהצורה "x found" ו- "x not found" (עבור ערך כלשהו x) בשלבים שונים של ביצועה.

מה תדפיס הפונקציה עבור שתי מחרוזות קלט, שתת המחרוזת המשותפת הארוכה ביותר שלהן היא באורך 25? כתבו את כל השורות (מהצורה "x found" ו- "x not found") שיודפסו, לפי סדר הדפסתן:

שאלה 2 (20 נק')

נתונה רשימה של מספרים שלמים אי שליליים L. הרשימה מייצגת נקודה במרחב רב מימדי, כלומר איברי הרשימה הם הקואורדינטות של הנקודה. למשל [0,0,0] מייצגת את ראשית הצירים במרחב תלת-מימדי.

נרצה לחשב בכמה דרכים ניתן להגיע מראשית הצירים (הנקודה המיוצגת ע"י רשימה של אפסים שאורכה כאורך של L) לנקודה המיוצגת ע"י L, כאשר בכל צעד מותר לזוז יחידה אחת קדימה על אחד הצירים (כלומר להוסיף 1 לאחד האיברים ברשימה).

למשל, עבור L=[1,2], כלומר הנקודה (1,2) במישור, יש שלוש דרכים כאלו:

[0,0] → [1,0] →[1,1] →[1,2]

[0,0] → [0,1] →[1,1] →[1,2]

[0,0] → [0,1] →[0,2] →[1,2]

א. השלימו את הפונקציה <u>הרקורסיבית</u> cnt_paths, שתחזיר את מספר הדרכים הנ"ל.

```
def cnt_paths(L):

   if _____________________________:
      return 1
```

א. נממש את cnt_paths_mem, גירסה משופרת של הפונקציה מסעיף א', תוך שימוש ב- memoization (מלבד זה לא יהיה הבדל באופן הפעולה של הפונקציות). הפונקציה cnt_paths_mem מומשה עבורכם ועליכם להשלים את הפונקציה cnt_paths_rec, שנקראת מתוך cnt_paths_mem.

```
def cnt_paths_mem(L):
    d = {}
    return cnt_paths_rec(L,d)


def cnt_paths_rec(L,d):
```

שאלה 3 (20 נק')

א. ממשו את פונקציית הגנרטור **filter** שמקבלת גנרטור **g** ופונקציה בוליאנית **cond** (פונקציה שמחזירה ערך אמת **True** או **False**). הפונקציה **filter** מחזירה גנרטור שיוצר את כל האברים ש **g** יוצר שגם מקיימים את התנאי **cond**. כלומר כל איבר **x** שנוצר על ידי **g**, שמקיים שהקריאה **cond(x)** מחזירה **True**. לדוגמא, בהינתן גנרטור בשם **naturals** שמייצר את כל המספרים הטבעיים החל ב- 1, ופונקציה **even** שמוגדרת כך:

```
def even(x):
    return x%2==0
```

נקבל

```
>>> g1 = iter([3,4,5,6,7])
>>> e1 = filter(g1,even)
>>> list(e1) #converts an iterator to a list
[4, 6]
>>> g2 = naturals()
>>> e2 = filter(g2,even)
>>> [next(e2) for i in range(10)]
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
def filter(g,cond):
```

ב. היזכרו בשיטת החצייה לחישוב שורש של פונקציה שראינו בכיתה. זהו תהליך של חיפוש בינרי שפועל על פונקציה רציפה **f**, בקטע **[a,b]** (עבור **a<b**) כאשר ידוע שערכי הפונקציה בקצות התחום, **f(a)** ו **f(b)** הם בעלי סימנים מנוגדים, ולכן קיימת בתחום נקודה בה ערך הפונקציה הוא 0.

בסעיף זה נכתוב פונקציה **bisectIter** שמקבלת כפרמטרים פונקציה **f**, ושני מספרים ממשיים **a** ו **b**, ומחזירה גנרטור של זוגות. כל זוג כזה מייצג קטע סגור, שבתוכו מצוי שורש של הפונקציה. הזוגות הללו מתקבלים בתהליך של חיפוש בינארי, באופן הבא:
בכל שלב מחושבת נקודת האמצע **m** של הקטע הנוכחי **[a,b]**. אם ערך הפונקציה בנקודה **m** הוא 0, מיוצר ע"י הגנרטור קטע (מיוצג ע"י רשימה באורך 2 שאיבריה הם גבולות התחום) שכולל רק את הנקודה הזאת, **[m,m]**, והפונקציה מסתיימת. אחרת מיוצר ע"י הגנרטור הקטע הבא לחיפוש, שגודלו מחצית מקודמו, והתהליך ממשיך.

יובהר שהגנרטור יכול להיות סופי (אם בשלב מסוים נמצאת נקודה שבה ערך הפונקציה הוא 0), או אינסופי.
לדוגמא, בהינתן הפונקציה

```
def g1(x):
    return x**2+x-6
```

נקבל

```
>>> it = bisectIter(g1,0.0,8.0)
>>> list(it)    #converts an iterator to a list
[(0.0, 8.0), (0.0, 4.0), (2.0, 2.0)]

>>> it = bisectIter(g1,0.0,5.0)
>>> [next(it) for i in range(6)]
[(0.0, 5.0), (0.0, 2.5), (1.25, 2.5), (1.875, 2.5), (1.875,
2.1875), (1.875, 2.03125)]
```

השלימו את הפונקציה **bisectIter**:

```
def bisectIter(f,a,b):
    assert a<b and f(a)*f(b) <= 0
    # also assume f is continuous
    mid = (a+b)/2
    yield (a,b)
    while f(mid) != 0.0:
```

ג. בסעיף זה נממש פונקציה **bisect** שמחשבת שורש של פונקציה בשיטת החצייה בדיוק רצוי. הפונקציה מקבלת את אותם הפרמטרים כמו **bisectIter** ובנוסף, פרמטר אופציונלי **epsilon** שמייצג "תנאי עצירה" - כלומר כשגודל התחום שמיוצר קטן מ- **epsilon**, הגנרטור יפסיק לייצר איברים.
הפונקציה משתמשת בפונקציות **bisectIter** ו **filter** . יש להשלים שורה אחת בודדת בלבד.
לדוגמה:

```
>>> bisect(g1,0,5)
1.9999980926513672
>>> bisect(g1,0,5,epsilon=0.0000001)
1.9999999552965164
```

```
def bisect(f,a,b,epsilon=0.00001):
  return next(filter(bisectIter(f,a,b) ,
                     _____)[0]
```

שאלה 4 (15 נק')

היזכרו ב- **class Polynomial** לייצוג פולינומים שראיתם בתרגיל הבית. להלן שתיים מהפונקציות שלו:

```
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]
        terms = "".join(terms)
        if terms == "":
            return "0"
        else:
            return terms[3:] #discard leftmost '+'
```

ניתן להתייחס לפעולות הכפל והחיבור של הפולינום כפעולות "סימבוליות", שאינן בהכרח פעולת החיבור והכפל האריתמטיות שאנו מכירים. למשל, ניתן להגדיר שהאפקט של פעולת "חיבור" הוא חישוב מקסימום בין שני הארגומנטים, בעוד האפקט של פעולת "כפל" הוא למעשה חיבור ("רגיל").
לדוגמא, תחת הגדרה זו, הערך של הפולינום Polynomial([0,2,0,1]) עבור x=4 הוא

$$\max(0, 2+4, 0+4+4, 1+4+4+4) = 13$$

נניח קיומו של **class** חדש בשם **arithmetics** שמכיל שתי פעולות (בנוסף ל- **_init_** ו- **_repr_** ריקות), ששמותיהן **plus** ו- **mult**. כל אחת מהן מקבלת שני מספרים כקלט וממשת אפקט קונקרטי כלשהו של פעולת החיבור\כפל הסימבולית בהתאמה . למשל:

```
class arithmetics():
 ....
   def plus(self,x,y):
     return max(x,y)

   def mult(self,x,y):
     return x+y
```

ניתן להניח שהמימוש של הפעולות מקיים את כל חוקי סדר הפעולות המוכרות ביחס לחיבור וכפל "רגילים" (אין חשיבות לסדר בין שתי פעולות חיבור או שתי פעולות כפל עוקבות, פעולת כפל קודמת לחיבור, וכן הלאה).

א. ממשו פונקציה בשם **evaluate** של המחלקה **Polynomial**, שמקבלת כקלטים (בנוסף ל- **self**):
- ערך **x** להצבה במשתנה הפולינום
- אובייקט **arithobj** מסוג **arithmetics**

ומחשבת את הערך של הפולינום ביחס לערך המשתנה וביחס ל"אפקט" של פעולות החיבור והכפל בפולינום כפי שמוכתבות ע"י אובייקט ה- **arithmetics** שהועבר.

```
class Polynomial():
  def evaluate(self, x, arithobj):
```

ב. נניח שכל קריאה לפעולות של **arithobj** וכל פעולה אריתמטית דורשת $O(1)$ זמן. מהי סיבוכיות זמן הריצה (זמן גרוע ביותר) של הקוד שכתבתם ביחס לאורך הקלט n (אורך רשימת המקדמים בפולינום)? הסבירו בקצרה.

שאלה 5 (20 נק')

השאלה עוסקת בשינוי באלגוריתם למפל-זיו לדחיסת טקסט.
כזכור, הפונקציה lz_compress2 (מופיעה לנוחיותכם בסוף השאלה) מחזירה את ייצוג הביניים של דחיסת למפל-זיו של המחרוזת text. למשל:

```
>>> lz77_compress2("abcdabc")
['a', 'b', 'c', 'd', [4, 3]]
>>> lz77_compress2("abab")
['a', 'b', 'a', 'b']
>>> lz77_compress2("abababab")
['a', 'b', [2, 4]]
```

בנוסף, ראינו בכיתה את הפונקציה:

```
def inter_to_bin(lst, w=2**12-1, max_length=2**5-1)
```

שבהינתן רשימה lst שמייצגת ייצוג ביניים של מחרוזת דחוסה, מחזירה מחרוזת של ביטים, המייצגת את המחרוזת הבינארית הדחוסה. נזכיר, שתו שלא נדחס ייוצג ע"י הביט 0 ואחריו 7 ביטים עבור התו עצמו (סה"כ 8 ביטים), ואילו מקטע שנדחס ייוצג ע"י הביט 1 ואחריו 12 ביטים עבור ההיסט אחורה, ו- 5 ביטים עבור אורך המקטע שנדחס (סה"כ 18 ביטים). שימו לב שבחישוב זה לקחנו בחשבון את ערכי ברירת המחדל של הפרמטרים w,max_length של שתי הפונקציות.

דוגמאות הרצה:

```
>>> inter_to_bin(lz77_compress2("abcdabc"))
'01100001011000100110001101100100100000000010000011'
>>> len(inter_to_bin(lz77_compress2("abcdabc")))
50      # 4*8 + 18
>>> inter_to_bin(lz77_compress2("abab"))
'01100001011000100110000101100010'
>>> len(inter_to_bin(lz77_compress2("abab")))
32      # 4*8
```

להלן מימוש של אלגוריתם שונה במעט עבור דחיסת למפל זיו:

```
def lz77_compress_new(text, start=0, w=2**12-1, max_length=2**5-1):
    n = len(text)
    if start >= n:
        return []

    #find the maximal length matching
    m,k = maxmatch(text, start, w, max_length)

    res1 = [text[start]] + \
           lz77_compress_new(text, start+1, w, max_length)
    res1_len = len(inter_to_bin(res1, w, max_length))

    if k < 3:
        return res1

    res2 = [[m,k]] + lz77_compress_new(text, start+k, w, max_length)
    res2_len = len(inter_to_bin(res2, w, max_length))

    if (res2_len < res1_len):
        return res2
    return res1
```

א. נתונה המחרוזת **s = "ababcabcd"**.

השלימו בשורה הבאה את הפלט (ייצוג הביניים) שיתקבל מהרצת **lz77_compress2(s):**

[_____]

השלימו בשורה הבאה את הפלט (ייצוג הביניים) שיתקבל מהרצת **lz77_compress_new(s,start=0):**

[_____]

ב. טענה: קיימת מחרוזת **s** שמקיימת:

len(inter_to_bin(lz77_compress_new(s,start=0))) <

len(inter_to_bin(lz77_compress2(s)))

תנו דוגמא למחרוזת **s** כזו בצירוף שני ייצוגי הביניים המתקבלים ע"י הפעלת כל אחת מהפונקציות הנ"ל, או הסבירו מדוע אין מחרוזת **s** כזו.

ג. טענה: קיימת מחרוזת **s** שמקיימת:

len(inter_to_bin(lz77_compress_new(s,start=0))) >

len(inter_to_bin(lz77_compress2(s)))

תנו דוגמא למחרוזת **s** כזו בצירוף שני ייצוגי הביניים המתקבלים ע"י הפעלת כל אחת מהפונקציות הנ"ל, או הסבירו מדוע אין מחרוזת **s** כזו.

להלן תזכורת לפונקציה **lz77_compress2** המקורית שראינו בכיתה:

```
def lz77_compress2(text, w=2**12-1, max_length=2**5-1):
   """LZ77 compression of an ascii text. Produces
      a list comprising of either ascii character
      or by a pair [m,k] where m is an offset and
      k is a match (both are non negative integers)"""
   result = []
   out_string = ""
   n = len(text)
   p = 0
   while p<n:
      if ord(text[p])>=128: continue
      m,k = maxmatch(text,p,w,max_length)
      if k<3:   # modified from k<2
         result.append(text[p]) #  a single char
         p+=1
      else:
         result.append([m,k])   # three or more chars in match
         p+=k
   return result  # produces a list composed of chars and pairs
```

סוף!

דף נוסף למקרה הצורך