מס' ת"ז_____ מס' מחברת_____

מבחן בקורס מבוא מורחב למדעי המחשב, CS1001.py

סמסטר א' 2014

מועד א, 2/2/2014

מרצים: פרופ' עמירם יהודאי, דר' דניאל דויטש

מתרגלים: מיכל קליינבורט, אמיר רובינשטיין

משך הבחינה: 3 שעות.

חומר עזר מותר: 4 עמודים בגודל A4 כ"א.

- במבחן 11 עמודים – בידקו שכולם בידיכם.
- יש לכתוב את כל התשובות בטופס הבחינה. המחברת תשמש כטיוטה בלבד ולא תיבדק.
- יש לענות על כל חמש השאלות. מספר הנקודות של כל שאלה הוא 20, אך זה 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
  - אין צורך להוסיף הערות (#comments) בגוף הקוד שאתם כותבים
- 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	ניקוד
1	
2	
3	
4	
5	
סה"כ	

בהצלחה !

<u>שאלה 1 (20 נק')</u>

השאלה עוסקת במיזוג m-ערוצי: מיזוג של $m\geq 2$ רשימות **ממוינות** (שנתונות כרשימה של רשימות) לרשימה אחת **ממוינת** המכילה את איברי כל הרשימות הנתונות. נניח כי אורך כל רשימה חסום ע"י n כלשהו.

בכל הסעיפים תוכלו לקרוא לפונקציה merge שראינו בכיתה, שממזגת שתי רשימות **ממוינות**:

```
>>> merge([1,2,2,3],[2,3,5])
[1, 2, 2, 2, 3, 3, 5]
```

א. השלימו את הגרסה הלא רקורסיבית הבאה, כך שתרוץ בזמן $O(n\cdot m^2)$. יש להשלים את 2 השורות המסומנות בלבד.

```
def multi_merge1(lst_of_lsts):
    m = len(lst_of_lsts)
    merged = []

    for _____________________________________________:

        _____________________________________________

    return merged
```

דוגמת הרצה:

```
>>> multi_merge1([[1,2,2,3],[2,3,5],[5]])
[1, 2, 2, 2, 3, 3, 5, 5]
```

ב. להלן גרסה רקורסיבית לפתרון הבעיה:

```
def multi_merge2(lst_of_lsts):
    m = len(lst_of_lsts)
    return multi_merge_rec(lst_of_lsts, 0, m-1)


def multi_merge_rec(lst_of_lsts, l, r):
    if l==r:
        return lst_of_lsts[l]
    L1 = multi_merge_rec(lst_of_lsts, l+1, r)
    return merge(lst_of_lsts[l],L1)
```

1. מהו עומק הרקורסיה שיוצרת multi_merge2? הקיפו בעיגול את התשובה הנכונה.

$O(m)$ $O(n)$ $O(n+m)$ $O(m\log n)$ $O(nm)$ $O(\log m)$

2. מהי סיבוכיות הזמן של multi_merge2? הקיפו בעיגול את התשובה הנכונה.

$O(m^2)$ $O(nm)$ $O(m\log n)$ $O(n\cdot m^2)$ $O(m+n)$ $O(n\cdot 2^m)$

ג. נכתוב כעת גרסה רקורסיבית נוספת. הקריאה הראשית מתוך `multi_merge2` לא משתנה. השלימו את הגרסה החדשה של הפונקציה `multi_merge_rec` במקום המיועד, כך שעומק הרקורסיה יהיה $O(\log m)$. אין לבצע slicing של רשימות.

```
def multi_merge_rec(lst_of_lsts, l, r): #another version
    if l==r:
        return lst_of_lsts[l]









    return _____
```

<u>שאלה 2 (20 נק')</u>

שאלה זאת עוסקת בתכונות של קודים. השאלה אינה מתייחסת לשימושים של הקוד (דחיסה, תיקון שגיאות, ...).

<u>תזכורת</u>: קוד C הוא מיפוי מאלפבית קלט Σ למחרוזות מעל האלפבית {0,1}. כלומר, הקידוד של כל תו ב Σ הוא מחרוזת בינארית. הקידוד של מחרוזת ב Σ הוא השרשור של קידודי התווים שמרכיבים את המחרוזת.

קוד C נקרא **חופשי רישא (prefix free)** אם לכל שני תווים שונים ב Σ, הקידוד של אחד מהם אינו רישא של הקידוד של השני.

קוד C נקרא **ניתן לשחזור (loseless)** אם בהינתן קידוד של מחרוזת, ניתן לזהות אותה באופן יחידי.

עבור כל אחד מהקודים בסעיפים א,ב,ג, סמנו את התשובה הנכונה מבין הבאות:

1. הקוד הוא חופשי רישא ואינו ניתן לשחזור
2. הקוד הוא חופשי רישא וניתן לשחזור
3. הקוד אינו חופשי רישא ואינו ניתן לשחזור
4. הקוד אינו חופשי רישא וניתן לשחזור

בנוסף, אם התשובה היא 1 או 3, עליכם להראות שתי מחרוזות מעל {a,b,c} שהקידוד שלהן זהה, ואת הקידוד שלהן. במקרים 2 ו- 4 אין צורך להוסיף דבר.

א. a → "010", b → "0110", c → "1"

התשובה היא: 1 / 2 / 3 / 4

ב. a → "0", b → "01", c → "111"

התשובה היא: 1 / 2 / 3 / 4

ג. a → "010", b → "001", c → "01"

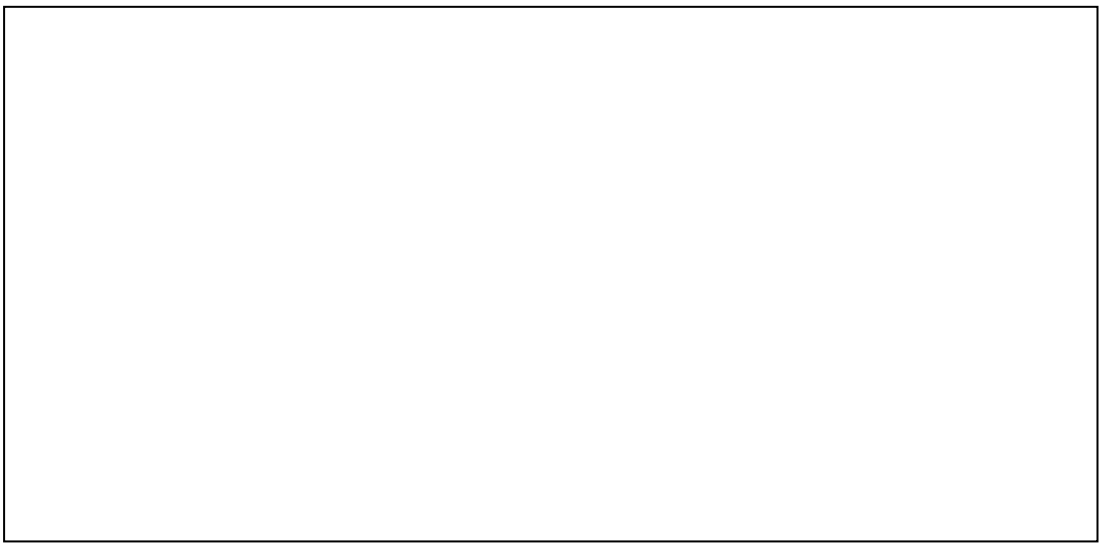
התשובה היא: 1 / 2 / 3 / 4

ד. נתון הקוד הבא, שניתן לשחזור:

a → "0", b → "01", c → "011"

עליכם להשלים את גוף לולאת ה- while בפונקציה decode שמקבלת מחרוזת מעל {0,1} שהיא קידוד חוקי, ומחזירה את המחרוזת מעל {a,b,c} שהיא המקור.

```
dict={"0":"a","01":"b","011":"c"} # ניתן להשתמש בזה, אבל לא חובה

def decode(b):
    result=[]
    n=len(b)
    p=0
    while p<n:
```



```
    return "".join(x for x in result)
```

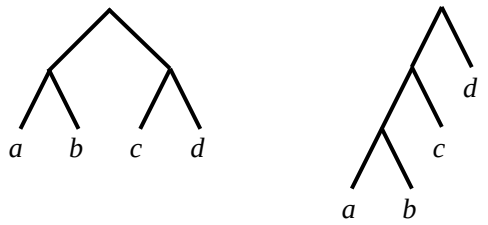
<u>שאלה 3 (20 נק')</u>

השאלה עוסקת בעצי האפמן.

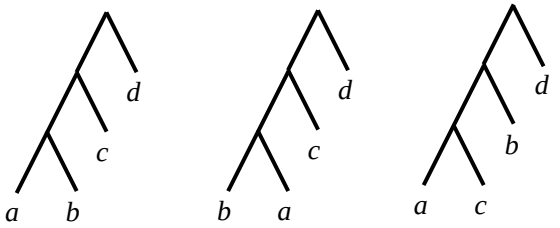
<u>הגדרה</u>: שני עצי האפמן ייקראו **אלטרנטיביים**, אם הם נוצרו מאותו קורפוס, אבל אוסף אורכי הקידודים של האותיות בשני העצים הוא שונה (ראו דוגמאות להלן).

שימו לב שהגדרה זו איננה מתייחסת למימוש הספציפי בפייתון לבניית עצי האפמן אותו ראינו בכיתה, אלא לאלגוריתם הכללי על פיו נבנה עץ האפמן על פי המשקלים, עם בחירה שרירותית במקרה של משקלים שווים ושל הסדר "ימין-שמאל".

למשל, עבור הקורפוס "*abcdd*" שני העצים הבאים אלטרנטיביים, כי אוסף אורכי הקידודים של הימני הוא 3,3,2,1 ואילו של השמאלי 2,2,2,2.



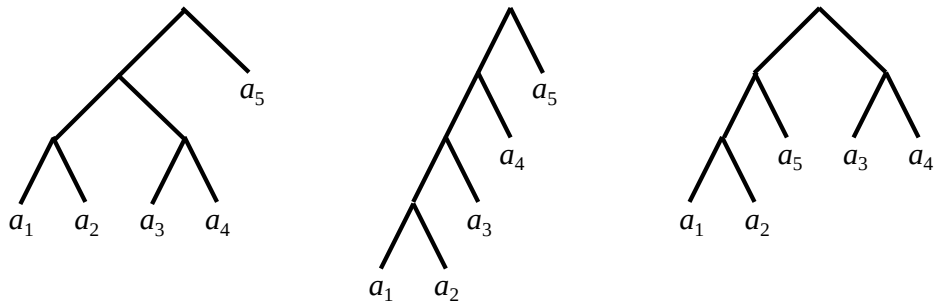
לעומת זאת בין שלושת העצים הבאים עבור אותו קורפוס אין אף שניים שהם אלטרנטיביים, כי בכולם יש אות אחת עם קידוד באורך 1, אות אחת עם קידוד באורך 2, ושתי אותיות עם קידוד באורך 3:



א. ציירו שני עצי האפמן אלטרנטיביים עבור הקורפוס "*abcdee*".

שני העצים האלטרנטיביים:

ב. להלן 3 עצי האפמן שונים, עבור האותיות $a_1,\ldots,a_5$. התדירויות של האותיות לא ידועות לכם.



<u>טענה</u>: ישנו קורפוס ש- 3 העצים הללו הם עצים אלטרנטיביים שלו.
הקיפו בעיגול האם הטענה הנ"ל נכונה או לא. אם לדעתכם היא נכונה, תנו את הקורפוס המתאים. אחרת, הסבירו במדויק מדוע הדבר לא ייתכן.

הטענה נכונה / לא נכונה (הקיפו בעיגול)

ג. מישל וברק התווכחו ביניהם לגבי יחידותם של עצי האפמן. מישל טענה שאם התדירויות של האותיות השונות לפיהן נבנה העץ הן <u>כולן שונות אחת מהשנייה</u>, אזי לא ייתכן שיש שניים או יותר עצים אלטרנטיביים. ברק טען שייתכנו מקרים כאלו בהם יש מספר עצים אלטרנטיביים שונים. החליטו מי צודק. אם ברק צודק, הציגו קורפוס שמדגים זאת. אם מישל צודקת, נמקו בקצרה.

מישל צודקת / ברק צודק (הקיפו בעיגול)

<u>שאלה 4 (20 נק')</u>

שאלה זו עוסקת בתמונות שמיוצגות ע"י מטריצה של ערכי רמת-אפור (grey level) לפיקסלים (כפי שראיתם בכיתה).

בכל התמונות בשאלה זו, כל שורה של המטריצה תכיל <u>משמאל לימין</u>:

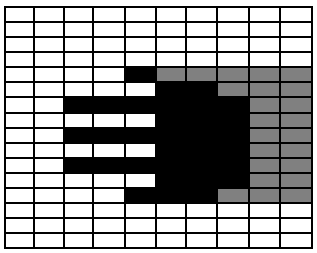
א. $i > 0$ פיקסלים לבנים

ב. $j \geq 0$ פיקסלים שחורים

ג. $k > 0$ פיקסלים אפורים (ערך 128) אם $j > 0$, אחרת: $k = 0$ פיקסלים אפורים.

כלומר אם נעבור משמאל לימין על הפיקסלים של שורה בודדת במטריצה נראה בהכרח פיקסלים לבנים, אחריהם ייתכן שיופיעו פיקסלים שחורים, ולבסוף יופיעו פיקסלים אפורים אם ורק אם היו פיקסלים שחורים בשורה.

דוגמא:



מטרתנו היא לספור באופן יעיל את מס' הפיקסלים השחורים בתמונה מסוג זה שניתנה כקלט.

נסמן את מספר השורות במטריצה ב - n ואת מספר העמודות ב - m.

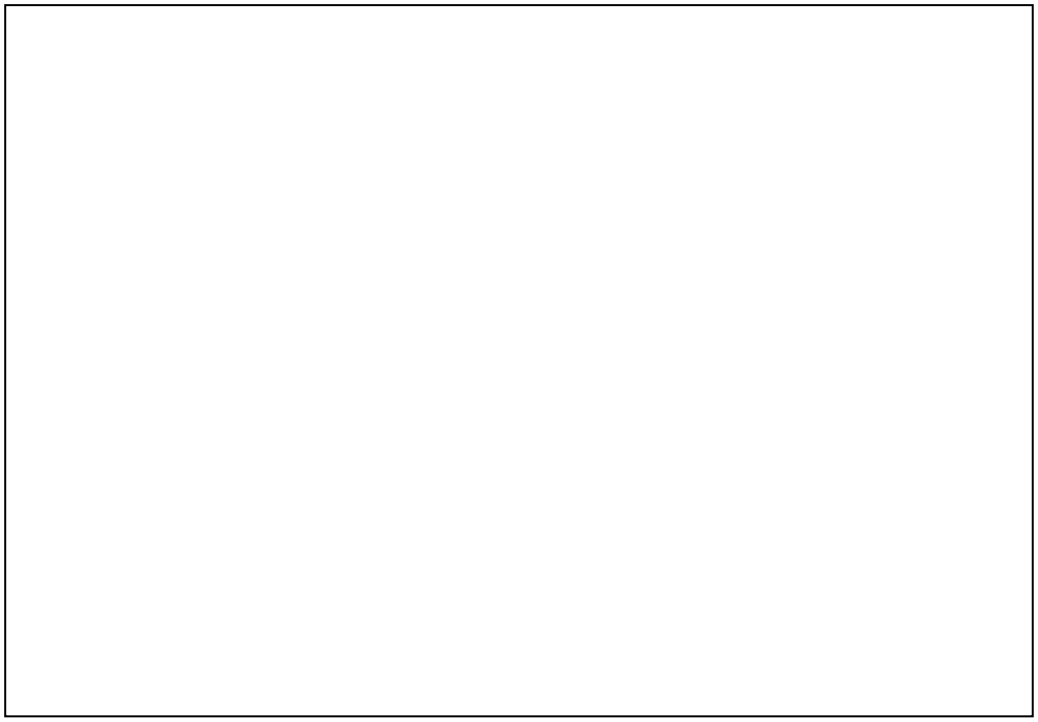
ניתן להניח כי הקלט שניתן לפונקציות תקין והתמונות מקיימות את התנאים הנ"ל.

א. השלימו בעמוד הבא את הפונקצייה `locate_leftmost_black(im, row_ind)` שמקבלת מטריצה `im` שמייצגת תמונה ואינדקס `row_ind` של שורה במטריצה, ומחזירה את האינדקס של הפיקסל השמאלי ביותר בשורה `row_ind` שצבעו שחור. אם אין פיקסלים שחורים בשורה על הפונקציה להחזיר None.

ניקוד מלא יינתן לפתרון יעיל ככל האפשר.

הערה: שימו לב, ישנם <u>שני</u> חלקים בפונקציה (הראשון הוא שורה בודדת) בהם חסרות שורות קוד אותן עליכם להשלים.

```
def locate_leftmost_black(im, row_ind):

    n,m = im.dim()
    white, black, grey = 255, 0, 128

    #if there are no black pixels in the row

    if _____________________________________________ :
        return None


    #otherwise, there is at least one black pixel
    #locate the index of the leftmost black pixel in the row

    leftmost_black = -1
```



```
    return leftmost_black
```

ב. גברת מוח כתבה את הפונקציה `locate_rightmost_black(im, row_ind)` שבהינתן מטריצה שמייצגת תמונה ואינדקס של שורה מחזירה את האינדקס של הפיקסל השחור הימני ביותר בשורה, או None אם אין פיקסלים שחורים בשורה.

השלימו בעמוד הבא את הפונקציה `count_black_pixels(im)` שמקבלת כקלט מטריצה שמייצגת תמונה מהסוג שהוזכר, ומחזירה את מספר הפיקסלים השחורים בתמונה.

עליכם להשתמש בשתי הפונקציות `locate_leftmost_black` ו-`locate_rightmost_black` ולתת פתרון יעיל ככל שתוכלו באמצעותן. הניחו כי שתי הפונקציות הללו רצות בסיבוכיות הטובה ביותר האפשרית עבורן.

```
def count_black_pixels(im):
    n,m = im.dim()
    blacks_count = 0
```

```
    return blacks_count
```

<u>שאלה 5 (20 נק')</u>

שאלה זאת עוסקת בפונקציות גנרטור.

<u>תזכורת</u>: קריאה לפונקצית גנרטור fgen מחזירה גנרטור (סוג של איטרטור), gen, וכל קריאה ל next(gen) מחזירה ערך כלשהו.

נאמר שפונקצית הגנרטור fgen **מייצרת** ערך x אם x מתקבל (מוחזר) אחרי מס' **סופי** של קריאות ל next(gen).

נאמר ש fgen **מייצרת** קבוצה (אולי אינסופית) של ערכים אם היא מייצרת כל ערך בקבוצה.

<u>בשאלה זו אין חשיבות לסדר ייצור הערכים ע"י הגנרטור</u>

א. רוני התבקש לכתוב פונקציית גנרטור שמייצרת את כל (אינסוף) הזוגות הסדורים של מספרים טבעיים (i,j), ללא חשיבות לסדר ייצורם וללא חזרות. שימו לב שהזוגות הם סדורים, כלומר (2,3) שונה מ- (3,2), והזוגות הנדרשים כוללים גם זוגות בהם i=j (למשל הזוג (2,2)).
הוא כתב את הקוד הבא:

```
def AllPairs():
 i=0
 while True:
    j=0
    while True:
          yield(i,j)
          j=j+1
    i=i+1
```

בקוד שלעיל ישנה בעיה. מהי? (סמנו בעמוד הבא את התשובה הנכונה ונמקו בקצרה).

1. קריאה ל- AllPairs() תגרום ללולאה אינסופית והתוכנית לא תעצור
2. קריאה ל- next על הגנרטור שמוחזר מ- AllPairs() תגרום ללולאה אינסופית והתוכנית לא תעצור
3. פונקצית הגנרטור מייצרת חלק מהזוגות הדרושים אך לא את כולם
4. זוגות מסוימים מיוצרים יותר מפעם אחת
5. אחרי מס' כלשהו של קריאות רצופות ל- next, קריאה נוספת ל- next תגרום ללולאה אינסופית והתוכנית לא תעצור

הסבר קצר:

_____

_____

_____

ב. השלימו את הקוד של פונקציית הגנרטור הבאה שמייצרת את כל הזוגות הסדורים של מספרים טבעיים (i,j) **המקיימים ש- j<i** .

```
def SomePairs():
 i=0
 while True:
```

ג. השלימו את הקוד של פונקציית הגנרטור RevGen. הקלט שלה הוא פונקציית גנרטור PairsGen שמייצרת מס' אינסופי של זוגות. RevGen מייצרת את כל הזוגות (i,j), שעבורם מתקיים שהזוג (j,i) מיוצר ע"י PairsGen.

```
def RevGen(PairsGen):
```

ד. כעת נתקן את הקוד של רוני כך ש- AllPairs() ייצר את כל הזוגות (i,j) כנדרש. השתמשו בפונקציות שכתבתם בסעיפים קודמים, וכן בשתי פונקציות עזר:

- פונקציית גנרטור בשם EqPairs שמייצרת את כל הזוגות של מספרים טבעיים מהצורה (i,i) (כלומר זוגות עבורם שני איברי הזוג שווים). אין צורך לממש פונקציה זו.

- פונקציית גנרטור בשם UnionGenerators. פונקציה זו מקבלת כקלט שני גנרטורים **אינסופיים** שמייצרים קבוצות איברים **זרות**. הפונקציה מייצרת את קבוצת האיחוד של קבוצות איברים אלה.

(1) השלימו את הקוד של UnionGenerators:

```
def UnionGenerators(gen1,gen2):
    while True:
```

(2) השלימו את הקוד של AllPairs תוך שימוש בפונקציות שכתבתם בסעיפים הקודמים ובפונקציות העזר.

```
def AllPairs():
```

סוף!