אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, חורף 2015

תרגיל בית מספר 5 (אחרון!) - להגשה עד 21 בינואר (יום רביעי) 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 skeleton4.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 012345678.pdf ו- 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.

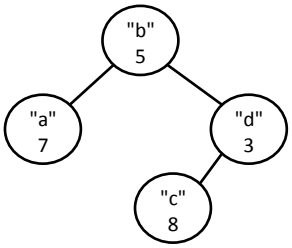
שאלה 1 – עצים בינאריים, רקורסיה ו- OOP

בהרצאה ראינו מימוש למבנה הנתונים "עץ חיפוש בינארי". במימוש שראינו, צומת בעץ יוצג ע"י אובייקט מהמחלקה Tree_node, ואילו העץ עצמו **לא** יוצג בגישת OOP.

בשאלה זו נניח כי הערכים (val) של הצמתים הם מספרים ממשיים כלשהם (חיוביים או שליליים) ונניח כי המפתחות (key) הם מחרוזות.

א. נגדיר <u>מסלול כבד ביותר</u>: רצף צמתים שמתחיל בשורש העץ ומגיע עד עלה כלשהו, שסכום הערכים (val) לאורכו הוא מקסימלי. את הסכום המקסימלי הנ"ל נכנה <u>משקל העץ</u>.

לדוגמה, משקל העץ הבא הוא 16, שמתאים למסלול הכבד ביותר "b"→"d"→"c" (הערך העליון בכל צומת הוא המפתח key, והתחתון הוא הערך val):



ממשו בקובץ השלד פונקציה רקורסיבית tree_weight שמקבלת Tree_node שמייצג את שורש העץ ומחזירה את משקל העץ.

ב. ממשו בקובץ השלד את הפונקציה tree_heavy_path שמקבלת Tree_node שמייצג את שורש העץ ומחזירה מסלול כבד ביותר בעץ. המסלול יוחזר כרשימה (list של פייתון) ובה רצף הערכים (val) של הצמתים. אם יש יותר ממסלול כבד ביותר יחיד, יוחזר אחד מהם, לבחירתכם.

<u>רמז</u>: tree_heavy_path תקרא תחילה ל- tree_weight מהסעיף הקודם עם שינוי קל: במהלך פעילותה תעדכן הפונקציה tree_weight בכל צומת בעץ שדה נוסף, ובו מידע שישמש לאחר מכן לחישוב המסלול הדרוש. יהיה עליכם להוסיף אם כן שדה למחלקה Tree_node.

<u>הנחיות והבהרות לכלל השאלה</u>:

- בסעיף א' הגישו את tree_weight לאחר השינוי הנדרש לצורך סעיף ב'. השינוי לא צריך להשפיע על הפלט של weight – ודאו שאכן זה כך (אחרת הטסטים האוטומטיים של סעיף א' עלולים להיכשל).
- המחלקה Tree_node שראיתם בכיתה מופיעה בקובץ השלד. אין לשנות מחלקה זו, למעט תוספת השדה המוזכרת בסעיף ב'.
- הפונקציות השונות לעצי חיפוש בינאריים מופיעות בקובץ השלד. שימו לב שלכל הפונקציות של עצים הוספנו את התחילית "tree_". אין לשנות כלל פונקציות אלה.

דוגמאות הרצה:

```
>>> t = None
>>> t = tree_insert(t,"b",5) #the first time we change t from None to
                             a "real" Node
>>> tree_insert(t,"d",3)
>>> tree_insert(t,"a",7)
>>> tree_insert(t,"c",8)
>>> tree_weight(t)
16
>>> tree_heavy_path(t)
 [5, 3, 8]
```

שאלה 2 – גנרטורים

תזכורת: 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.

שימו לב: בשאלה זו אין חשיבות לסדר ייצור הערכים ע״י הגנרטור.

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
```

ציינו בקובץ ה pdf מה הבעיה בקוד שכתב רוני ונמקו בקצרה.

ב. ממשו בקובץ השלד את פונקציית הגנרטור SomePairs שמייצרת את כל הזוגות הסדורים של מספרים טבעיים (i,j) **המקיימים ש: i < j**.

ג. ממשו בקובץ השלד את פונקציית הגנרטור RevGen שמקבלת כקלט פונקציית גנרטור PairsGen שמייצרת מס׳ סופי או אינסופי של זוגות. RevGen מייצרת את כל הזוגות (i,j) שעבורם מתקיים שהזוג (j,i) מיוצר ע״י PairsGen. אם מספר הזוגות שמייצרת PairsGen הוא סופי אז תתרחש שגיאת StopIteration.

דוגמת הרצה:

```
>>> def f():
        return ((i, 2*i) for i in range(4)) #generator expression
>>> {(x,y) for (x,y) in RevGen(f)}
{(6, 3), (4, 2), (0, 0), (2, 1)}  #The order of pairs does not matter!
>>> g = RevGen(f)
#the order of pairs in the following commands does not matter!
>>> next(g)
(0, 0)
>>> next(g)
(2, 1)
>>> next(g)
(4, 2)
>>> next(g)
(6, 3)
>>> next(g)
Traceback (most recent call last):
...
StopIteration
```

ד. ממשו בקובץ השלד פונקציית גנרטור בשם EqPairs שמייצרת את כל הזוגות של מספרים טבעיים מהצורה (i,i) (כלומר זוגות עבורם שני איברי הזוג שווים).

ה. ממשו בקובץ השלד פונקציית גנרטור בשם UnionGenerators שמקבלת כקלט שני גנרטורים אינסופיים (ולא פונקציות גנרטור) שמייצרים קבוצות זרות. הפונקציה מייצרת את קבוצת האיחוד של קבוצות איברים אלה.

ו. ממשו בקובץ השלד את הקוד של AllPairs תוך שימוש בפונקציות שכתבתם בסעיפים הקודמים. לשם הבהרה, לאחר הפקודה g = AllPairs(), ניתן יהיה לבצע next(g) אינסוף פעמים, כך שכל זוג (i,j) יתקבל מתישהו (אחרי מספר סופי של פקודות next).

שאלה 3 – קארפ-רבין

בשאלה זו נרצה לענות על השאלה, האם תמונה נתונה מכילה תת-תמונה ריבועית בגודל נתון שחוזרת על עצמה יותר מפעם אחת, כאשר שני מופעים של תת תמונה יכולים לחפוף אחד את השני באופן חלקי. נכנה את תת-התמונה "חלון", ונניח שגודלו k x k פיקסלים. התמונה וכן החלון ייוצגו באמצעות המחלקה Matrix שראינו בקורס. כל פיקסל מייצג ערך אפור בין 0 (שחור) ל- 255 (לבן). הניחו שמתקיים k<= min(n,m) עבור תמונה בעלת n שורות ו m עמודות.

פתרון יעיל אפשרי מתבסס על הרעיון של אלגוריתם Karp-Rabin, בו השתמשנו על מנת לחפש מחרוזת תבנית בתוך מחרוזת טקסט: מחשבים מעין טביעת אצבע של כל החלונות בגודל k x k אשר מוכלים בתמונה הגדולה. מדווחים על חזרה אם נמצאו שתי טביעות אצבע שוות.

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

נגדיר אם כן פונקציה fingerprint, אשר בהינתן מטריצה ריבועית k x k מחזירה מספר, שנקרא לו "טביעת אצבע" של המטריצה:

```
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))
```

לצורך פתרון יעיל, נזדקק לפונקציה move_right אשר מקבלת (בסדר זה) תמונה mat (כלומר אובייקט מסוג Matrix), אינדקסי שורה i ועמודה j של פיקסל בתוכה, גודל חלון k, ואת טביעת האצבע fp 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 **ימינה** בפיקסל אחד. הפונקציה תניח 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. השלימו בקובץ השלד את מימוש הפונקציה move_right, בסיבוכיות זמן ריצה O(k).

ב. השלימו בקובץ השלד את מימוש הפונקציה move_down, בסיבוכיות זמן ריצה O(k). ההבדל בין פונקציה זו לזו מסעיף א' הוא ש move_down מחזירה את טביעת האצבע של החלון אשר מתקבל על ידי הזזת החלון המקורי **<u>מטה</u>** בפיקסל אחד. גם כאן הפונקציה מניחה כי החלון שלמטה אכן קיים (כלומר שלא הגענו לגבול התחתון של התמונה).

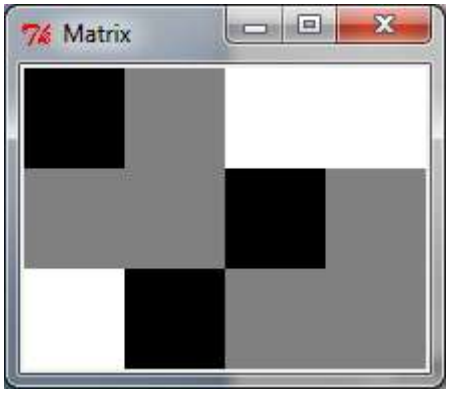
ג. עתה נממש את הפונקציה image_fingerprints, בעזרת הפונקציות מהסעיפים הקודמים. אשר מקבלת תמונה mat וגודל חלון k. הפונקציה מחזירה את טביעות האצבע של כל החלונות בגודל k x k אשר מוכלים בתמונה. לשם כך יוצרת הפונקציה רשימה של רשימות (מעין טבלה) בגודל המתאים, ומאחסנת באינדקס ה $[j][i]$ את טביעת האצבע של החלון, אשר פינתו השמאלית-עליונה ממוקמת בשורה ה i בעמודה ה j של התמונה mat. השלימו בקובץ השלד את מימוש הפונקציה, עבור תמונה בגודל n x m וגודל חלון k, בסיבוכיות זמן O(mnk).

ד. לסיום, ממשו בקובץ השלד את הפונקציה repeating_subfigure, שמקבלת מטריצה mat שמייצגת תמונה, וגודל צלע k של חלון ריבועי. הפונקציה תחזיר True אם יש בתמונה תת-תמונה ריבועית בגודל kxk שמופיעה בה יותר מפעם אחת, אחרת False.

<u>הנחיות</u>: (1) הפונקציה עלולה להחזיר תשובה שגויה בסיכוי מסוים, אם לשני "חלונות" שונים יש אותה טביעת אצבע. (2) חישוב טביעות האצבע ייעשה ע"י image_fingerprints. (3) בדיקה האם יש טביעות אצבע חוזרות תיעשה בסיבוכיות זמן O(mn) <u>בממוצע</u> (האם פתרון נאיבי שבו פשוט בודקים כל זוג מתאים כאן?).

ה. ציינו את חסרונה העיקרי של הפונקציה fingerprint שהופיעה בתחילת השאלה, ביחס לבעיה אותה אנו מנסים לפתור בשאלה זו. תארו במילים שיפור אפשרי לפונקציה, שיסייע להתגבר על חסרון זה.

דוגמאות הרצה:




```
>>> im = Matrix.load("./sample.bitmap")
>>> im.display(zoom = 50)
>>> k=2
>>> fingerprint(im[:k,:k])
384
>>> fingerprint(im[1:k+1,1:k+1])
256
>>> move_right(im, 0, 0, k, 384)
511
>>> move_down(im, 0, 1, k, 511)
256
>>> image_fingerprints(im, k)
[[384, 511, 638], [511, 256, 384]]
>>> repeating_subfigure(im, k)
True
>>> repeating_subfigure(im, 3)
False # there is no repeating subfigure of size 3x3
```

שאלה 4 – עיבוד תמונה

הערה: בשאלה זו הקוד שלכם צריך לרוץ כאשר הקובץ matrix.py מההרצאות / תרגולים נמצא באותה ספריה בה נמצא קובץ הפתרון. שימו לב שקובץ השלד כבר מכיל את השורה **from matrix import *** שורה זו מייבאת את כל הקוד מהקובץ matrix.py, כך שניתן להשתמש בו כאילו הופיע בקובץ השלד עצמו.

א. **סגמנטציה** של תמונה היא חלוקתה למקטעים (סגמנטים) של פיקסלים, כאשר המקטעים זרים ומכסים את כל התמונה. לפיקסלים באותו סגמנט יש בד"כ תכונות משותפות. מטרת הסגמנטציה היא לפשט את הייצוג של התמונה לאוסף של אובייקטים בעלי משמעות, כדי להקל על עיבוד התמונה לצרכים שונים (כגון מציאת גבולות של אובייקטים בתמונה, ספירת עצמים וכו').

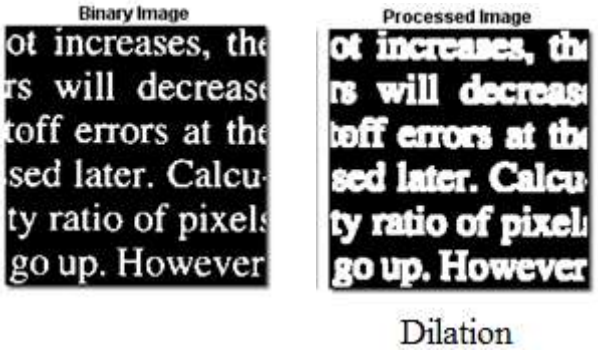
סגמנטציה **בינארית** של תמונת גוני אפור מחלקת אותה לשני מקטעים – שחור ולבן. אחת השיטות הפשוטות לביצוע סגמנטציה בינארית היא שיטת ה- **Thresholding**: מחליטים על סף (threshold) מסוים, וכל פיקסל בעל ערך גבוה או שווה לסף הופך ללבן (255), בעוד שכל יתר הפיקסלים הופכים לשחור (0).

ממשו בקובץ השלד פונקציה segment(im, th) המקבלת מטריצה im (אובייקט מהמחלקה Matrix שמימשנו בקורס), וערך סף $0 \le th < 256$, ומחזירה מטריצה חדשה, שהיא תוצאה ביצוע thresholding על im עם הסף th.

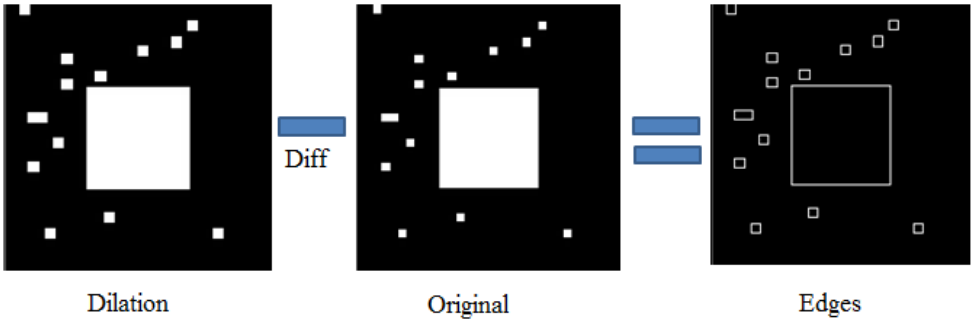
לדוגמה, להלן תמונה המיוצגת ע"י מטריצה im, ומימינה תוצאה הסגמנטציה שלה עם th=129, כלומר התמונה שתוצג בעת ביצוע הפקודה segment(im,129).display():

ב. **גבול (edge)** בתמונה הוא איזור בו יש שינוי חד בבהירות הפיקסלים. זיהוי גבולות (edge detection) הוא תהליך זיהוי של אזורים כאלו בתמונה. שיטה פשוטה לזיהוי גבולות בתמונה בינארית (שחור-לבן) עושה שימוש באופרטור שנקרא dilation (הרחבה): עבור פיקסל i,j, אם אחד משכניו לבן, הפיקסל i,j יהפוך ללבן. "שכנים" הם פיקסלים במרחק k מהפיקסל הנוכחי (כלומר ריבוע במימדים 2k+1 על 2k+1 שמרכזו הפיקסל הנוכחי). שימו לב כי dilation הינו סוג של אופרטור לוקאלי, המשנה פיקסלים בתמונה בהשפעת שכניהם, זאת בדומה לאופרטורים הלוקאליים *ממוצע* ו*מציון* אותם ראיתם בכיתה (ששימשו לצורך ניקוי רעש מסוגים שונים). בנוסף, כפי שיובהר בהמשך, השימוש ב- dilation לצורך זיהוי גבולות מניח כי העצמים בתמונה הם בהירים, ואילו הרקע הוא כהה.

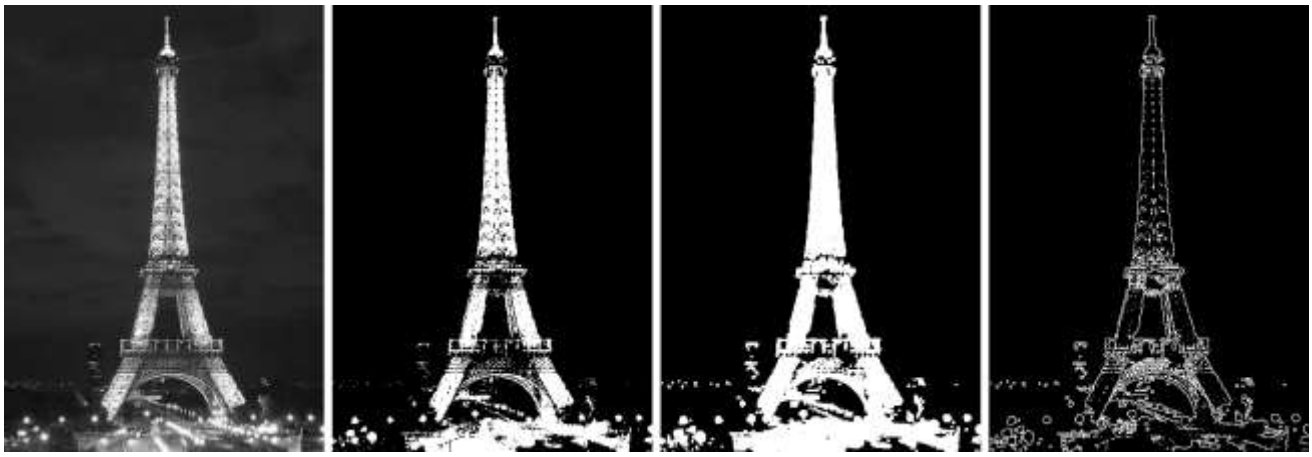
לדוגמה, התמונה משמאל היא תמונה בינארית, ומימינה תוצאת הפעלת האופרטור dilation עליה (התמונות מתוך http://micro.magnet.fsu.edu/primer/java/digitalimaging/russ/erosiondilation/index.html):



זיהוי גבולות בתמונה נעשה ע״י הפעלת dilation על התמונה, ואז חישוב ההפרש בין כל פיקסל לאחר ה- dilation לבין הפיקסל באותו מיקום בתמונה המקורית. לדוגמה:



דוגמה נוספת (A– תמונת גווני אפור, B – הפעלת סגמנטציה בינארית עם סף ששווה 100, C – הפעלת dilation על B עם k=1, D– תוצאה חישוב ההפרש C-B):



- ממשו פונקציה dilate(im, k) שמקבלת מטריצה im (אובייקט מהמחלקה Matrix), וגודל סביבה k (בדומה לשאר האופרטורים הלוקאליים שנלמדו) ומחזירה את התמונה לאחר dilation. למשל, עבור המטריצה B הנ״ל, התמונה C תתקבל ע״י dilate(B,1).display().

הנחיות:

- יש להשתמש בפונקציה local_operator שנלמדה בכיתה ומופיעה בקובץ השלד (יחד עם copy ו items שנלמדו בהרצאה). אין לשנות שלוש פונקציות אלו.
- local_operator לא מטפלת בפיקסלים שנמצאים במרחק k מקצוות התמונה. אין צורך לשנות זאת.
- ניתן לממש את dilate בשורה אחת בודדת.

- ממשו את הפונקציה diff(A,B) שמקבלת שתי מטריצות (באותו גודל, אין צורך לוודא זאת) ומחזירה את ההפרש שלהן (פיקסל-פיקסל).

- ממשו את edges(im, k, thr) שמקבלת מטריצה im, גודל סביבה k, וערך סף thr, ומחזירה מטריצה חדשה ובה הגבולות בתמונה A, בהתאם לשיטה הנ"ל. למשל, עבור התמונה A בדוגמה שלעיל, ערך סף 100 ו- k=1 תוחזר התמונה D.

דוגמאות הרצה:

```
>>> m1 = Matrix(4,4,0)
>>> m1[0,0] = 20
>>> m1[1,0] = 60
>>> m2 = segment(m1,10)
>>> m2.rows
[[255, 0, 0, 0], [255, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]:
>>> m3 = dilate(m2,1)
>>> m3.rows
[[255, 0, 0, 0], [255, 255, 0, 0], [0, 255, 0, 0], [0, 0, 0, 0]]:
>>> m4 = diff(m3,m2)
>>> m4.rows
[[0, 0, 0, 0], [0, 255, 0, 0], [0, 255, 0, 0], [0, 0, 0, 0]]:
>>> edges(m1,1,10) == m4
True
```

ג. הריצו את הפונקציה edges לזיהוי גבולות על תמונה לבחירתכם (השתמשו ב- image2bitmap כדי להמירה תמונה "רגילה" לפורמט bitmap שניתן לייצוע ע"י אובייקט מטיפוס Matrix שנלמד בכיתה). צרפו לקובץ ה- pdf את התמונה המקורית ואת תוצאת זיהוי הגבולות עליה. ציינו מה היה הסף בו השתמשתם לסגמנטציה ומדוע בחרתם בסף זה.

שאלה 5 – hash וטקסטים

בשאלה זו נמצא את 10 המילים הנפוצות ביותר בספר "הרפתקאות תום סוייר" מאת מארק טוויין.
את הספר ניתן למצוא בגרסה אלקטרונית באתר של פרוייקט גוטנברג:

http://www.gutenberg.org/cache/epub/74/pg74.txt

בקובץ השלד תמצאו שתי פונקציות שמומשו עבורכם (ואין לשנות אותן):

- הפונקציה download(url) שמקבלת מחרוזת המייצגת כתובת url, ומחזירה את הטקסט שמכיל עמוד האינטרנט בכתובת זו. כדי לקבל את הסיפור הנ"ל כטקסט עליכם לכתוב:

```
>>> tom = download ("http://www.gutenberg.org/cache/epub/74/pg74.txt")
```

- הפונקציה clean(text) ש"מנקה" מחרוזת text שהיא מקבלת באופן הבא: מסירה כל תו שאיננו אות באנגלית, רווח או תווי ירידת שורה, וכן הופכת אותיות גדולות באנגלית לקטנות. למשל:

```
>>> clean("a\nBc"+chr(3)+"d")
'a\nbc d'
```

בשלב ראשון נרצה לחשב את תדירויות כל אחת מהמילים בטקסט.

א. להלן פונקציה שמקבלת רשימה (list) של מילים, ומחזירה את תדירויות המילים (כרשימות מהצורה [word, cnt] כאשר cnt הוא מספר הפעמים ש- word מופיעה בקלט):

```
def count_words_naive(words):
    count_list=[]
    words_set = set(words) #set of different words
(no repetition)
    for word in words_set:
        count_list += [ [word, words.count(word)] ]
    return count_list
```

את הפונקציה הזו יש להפעיל אם כן כך:

```
>>> words = clean(tom).split()
>>> count_words_naive(words)
```

הריצו את הפונקציה. תנו הערכה לזמן שייקח לה לסיים. מהי סיבוכיות הפונקציה כתלות ב- n=len(words)? הפרידו למקרה הגרוע ולמקרה הטוב (ציינו מהו כל אחד).

ב. נייעל את המימוש, ע"י שימוש **במילון של פייתון** (dict), שכזכור ממומש כטבלת hash. השלימו בקובץ השלד את המימוש של count_words. הפונקציה מקבלת רשימת מילים, ומחזירה מילון של פייתון המכיל את המילים כמפתחות, כאשר הערך של כל מילה הוא כמות המופעים שלה. השורה הראשונה והאחרונה בפונקציה כבר כתובות עבורכם, ואין לשנות אותן. דוגמת הרצה (שימו לב שלסדר האיברים במילון אין חשיבות):

```
>>> count_words (["ab", "cd", "cd", "ef", "cd", "ab"])
{'ab': 2, 'ef': 1, 'cd': 3}
```

לצורך מציאת 10 המילים הנפוצות, נרצה כעת למיין את אוסף התדירויות שקיבלנו. נשתמש בפלט של הפונקציה count_words מסעיף ב'.

ג. השלימו את הפונקציה sort_by_cnt, כך ש**תכיל שורה אחת בלבד ובה פקודת return**. הפונקציה מקבלת מילון שמוחזר מהפונקציה count_words, ומחזירה רשימה של tuples מהצורה (word,cnt), כאשר word הופיעה בטקסט cnt פעמים, והרשימה ממוינת לפי cnt מגדול לקטן. דוגמת הרצה:

```
>>> sort_by_cnt(count_words (["ab", "cd", "cd", "ef", "cd", "ab"]))
[('cd', 3), ('ab', 2), ('ef', 1)]
```

<u>הערה</u>: אם יש מילים באותה תדירות, הסדר ביניהן לא משנה.

<u>עזרה והנחיה</u>: השתמשו בפונקציה sorted, ובפונקציה items של המחלקה dict של פייתון.

ד. ציינו בקובץ התשובות (pdf) מהן 10 המילים הנפוצות. אפשר פשוט לצרף את הפלט של הפקודות הבאות:

```
>>> tom = download("http://www.gutenberg.org/cache/epub/74/pg74.txt")
>>> words = clean(tom).split()
>>> word_cnt = count_words(words)
>>> word_cnt_sorted = sort_by_cnt(word_cnt)
>>> word_cnt_sorted[:10]
```

עזרה נוספת בבדיקת תקינות:

```
>>> len([cnt for cnt in word_cnt.values() if cnt==1])
3640 #number of words that appear only once
```

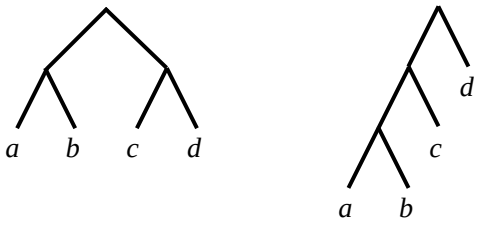
שאלה 6 – דחיסת האפמן

השאלה עוסקת בעצי האפמן.

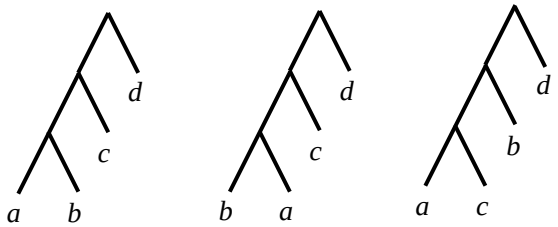
הגדרה: שני עצי האפמן ייקראו **אלטרנטיביים**, אם הם נוצרו מאותו קורפוס, אבל אוסף אורכי הקידודים של האותיות בשני העצים הוא שונה (ראו דוגמאות להלן).

שימו לב שהגדרה זו איננה מתייחסת למימוש הספציפי בפייתון לבניית עצי האפמן אותו ראינו בכיתה, אלא לאלגוריתם הכללי על פיו נבנה עץ האפמן מתוך 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.

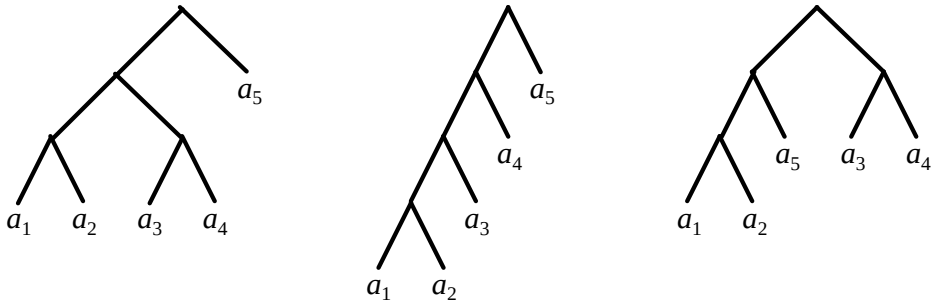


לעומת זאת בין שלושת העצים הבאים עבור אותו קורפוס אין אף שניים שהם אלטרנטיביים, כי בכולם יש אות אחת עם קידוד באורך 1, אות אחת עם קידוד באורך 2, ושתי אותיות עם קידוד באורך 3:



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

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.
האם הטענה נכונה או לא? אם לדעתכם היא נכונה, תנו את הקורפוס המתאים. אחרת, הסבירו במדויק מדוע הדבר לא ייתכן.

ג. מיכל ואמיר התווכחו ביניהם לגבי יחידותם של עצי האפמן. מיכל 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. אמיר טען שייתכנו מקרים כאלו בהם יש מספר עצים אלטרנטיביים שונים.
החליטו מי צודק. אם אמיר צודק, הציגו קורפוס שמדגים זאת. אם מיכל צודקת, נמקו בקצרה.

<u>שאלה 7 – זיו למפל – בונוס 15 נקודות (כי יילמד רק כשבוע וחצי לפני ההגשה)</u>

א. כזכור, באלגוריתם Lempel-Ziv דוחסים חזרות באורך לפחות 3 (מתעלמים מחזרות באורך 1,2 משום שדחיסתן אינה משתלמת). אם נסמן ב-L את אורך החזרה המינימלי שהאלגוריתם דוחס, אז L=3.
האם תיתכן מחרוזת שדחיסתה עם L=4 תהיה יעילה יותר מאשר עם L=3? אם לדעתכם כן, רשמו את המחרוזת ואת ייצוג הביניים* של הדחיסה, עבור L=3 ועבור L=4. אם לדעתכם לא, הסבירו מדוע.
* דוגמה לייצוג ביניים: ייצוג הביניים של המחרוזת 'abcabcdededе' הוא ['a','b','c',(3,3),'d','e',(2,4)]

ב.

i. לפניכם מוצג קוד עבור הפונקציה genString(n) שמייצרת מחרוזת באורך n מתוך התפלגות ידועה של שכיחות אותיות (הנתונה ע"י המחרוזת freq בקוד).

```
def genString(n):
    freq = 'a'*25+'bcdefghijklmnopqrstuvwxyz'
    randLetters = [random.choice(freq) for i in range(n)]
    return ''.join(randLetters)
```

תהי s=genString(100000). איזו דחיסה צפויה לתת יחס דחיסה טוב יותר עבור s: Huffman או Lempel-Ziv? הסבירו את תשובתכם.
הבהרה: קידוד Huffman כאן ישתמש ב-s הן בתור corpus והן בתור text.

ii. נחליף את המחרוזת freq במחרוזת הבאה: freq = 'a'*2500+'bcdefghijklmnopqrstuvwxyz'
האם לדעתכם התשובה תשתנה? הסבירו.

ג. נניח שעבור טקסט באורך n, מאפשרים לאורך החזרה המקסימלי באלגוריתם Lempel-Ziv להיות n-1 (במקום 31 כפי שמופיע בערכי ברירת המחדל של האלגוריתם שהוצג בהרצאה). שאר פרטי האלגוריתם ללא שינוי. רוצים לדחוס באופן זה את המחרוזת '01010101…' באורך n.
כיצד נראה ייצוג הביניים של הדחיסה? מהו יחס הדחיסה (=מספר הביטים במחרוזת הדחוסה חלקי מספר הביטים במחרוזת ללא דחיסה) כתלות ב- n? תנו תשובה בסדר גודל במונחים של O(…)

ד. נניח, כי בנוסף לשינוי שתואר בסעיף ב', מכניסים את שני השינויים הבאים באלגוריתם:
- אין אפשרות לחפיפה בין סגמנטים חוזרים. במילים אחרות, החזרה לא תוכל לעבור את הנקודה הנוכחית בטקסט. למשל, ייצוג הביניים של המחרוזת 'abcabcabc' יהיה ['a','b','c',(3,3),(3,3)] במקום ייצוג הביניים באלגוריתם הרגיל - ['a','b','c',(3,6)].
- גודל החלון יוגבל ל- n-1 (במקום 4096 כפי שמופיע בערכי ברירת המחדל של האלגוריתם שהוצג בהרצאה).

כיצד יראה כעת ייצוג הביניים של המחרוזת '01010101…' באורך n? מהו יחס הדחיסה כתלות ב- n? הניחו לשם פשטות כי n הוא חזקה שלמה של 2, ותנו תשובה בסדר גודל ((O(…). הסבירו את תשובתכם.

טיפ: אתם יכולים לבדוק את התשובות לכל הסעיפים ע"י הרצות...

<u>סוף</u>