אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, חורף 2016-2017

תרגיל בית מספר 6 - להגשה עד 29 בינואר (יום ראשון) בשעה 23:55

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

שימו לב: שאלה 5 אינה להגשה!!!

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.

- בכל השאלות ניתן להניח כי הקלט לתכנית / לפונקציות הינו תקין, אלא אם צוין במפורש אחרת.

- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.

שאלה 1 – 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 השלמה היא בגודל n שורות על m עמודות, ומתקיים k<= min(n,m). התמונה וכן החלון ייוצגו באמצעות המחלקה Matrix שראינו בקורס. כל פיקסל מייצג ערך אפור בין 0 (שחור) ל- 255 (לבן).

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 והשוואה בין מספרים שלמים רצות בזמן קבוע O(1) (כלומר ללא תלות בגודל המספר).

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 kxk, אשר הפינה השמאלית העליונה שלו ממוקמת mat[i][j]. 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 **מטה** בפיקסל אחד. גם כאן הפונקציה מניחה כי החלון שלמטה אכן קיים (כלומר שלא הגענו לגבול התחתון של התמונה).

ג. עתה נממש את הפונקציה has_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. כאמור, מותרות חפיפות בין תת-תמונות.

הנחיות: (1) מותר שהפונקציה תחזיר תשובה שגויה, אם לשני "חלונות" שונים יש אותה טביעת אצבע. (2) חישוב טביעות האצבע ייעשה ע"י הפונקציות מהסעיפים הקודמים. (3) המקרה הגרוע ביותר מבחינת סיבוכיות הריצה הוא כאשר התמונה אינה מכילה תת-תמונה חוזרת (מדוע?). במקרה זה סיבוכיות הזמן הדרושה לחישוב כל טביעות האצבע תהיה O(mnk), ואילו סיבוכיות הזמן הדרושה לכלל הבדיקות האם יש טביעות אצבע חוזרות תהיה O(mn) בממוצע (חישבו באיזה מבנה נתונים של פייתון יש לאחסן את טביעות האצבע כדי לעמוד בדרישה האחרונה).

ד. לפונקציה fingerprint שהופיעה בתחילת השאלה ישנו חיסרון בולט, ביחס לבעיה אותה אנו מנסים לפתור בשאלה זו. כתוצאה מכך, has_repeating_subfigure עלולה להחזיר True למרות שאין תת-תמונה ריבועית בגודל kxk שמופיעה יותר מפעם אחת (כזכור, במקרה כזה התקבל false positive).

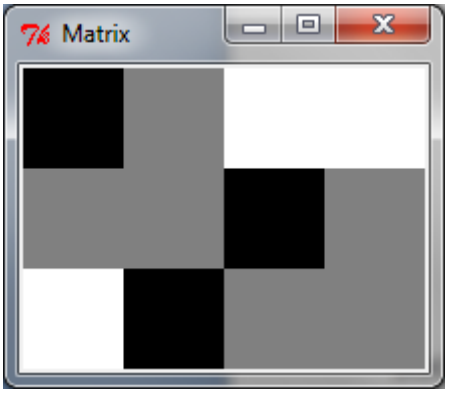
תנו דוגמה למטריצה שמייצגת תמונה בגודל 4x4 שבה יתקבל false positive. את הדוגמה תנו כערך שמוחזר מהפונקציה problematic_matrix() (בקובץ השלד), זאת משום שנרצה לבדוק את הפתרון לסעיף זה באופן אוטומטי. למשל, אם לדעתכם מטריצת האפסים בגודל 4x4 מהווה דוגמה מתאימה אז השלימו את problematic_matrix כך:

```
def problematic_matrix():
    im = Matrix(4,4)
    for i in range(4):
        for j in range(4):
            im[i,j] = 0
    return im
```

דוגמאות הרצה – בעמוד הבא (לא לפספס!).

דוגמאות הרצה (שחור – 0, לבן – 255, אפור - 128):

```
>>> 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
>>> has_repeating_subfigure(im, k)
True
>>> has_repeating_subfigure(im, 3)
False # there is no repeating subfigure of size 3x3
```



שאלה 2 - האפמן

שימו לב שעל מנת להריץ את הפונקציות בסעיפים הבאים עליכם לדאוג שבתיקיה (folder) בה נמצא קובץ הקוד שלכם נמצא גם הקובץ huffman.py . בנוסף יהיה עליכם להוסיף את הפקודה הבאה לקובץ השלד:

```
from huffman import *
```

א. השלימו בקובץ השלד את הפונקציה weighted_length(D, W), אשר מקבלת שתי רשימות שוות אורך: D שמכילה את הקידודים של תווי האי"ב (כמחרוזות בינאריות), ו- W, שמכילה את המשקלים של תווים אלו (מספר המופעים שלהם בקורפוס כלשהו). שימו לב שהערך באינדקס i ברשימה D מתאים לקידוד של התו ה-i באי"ב. כמו כן, הערך באינדקס i ברשימה W מתאים למספר המופעים של אותו התו (התו ה-i באי"ב) בקורפוס ממנו נלמדו התדירויות. הפונקציה תחזיר את האורך הממושקל של הקידוד, כפי שהוגדר בשקף 21 בהרצאה.
למשל (עבור קורפוס בעל 3 תווים שונים):

```
>>> weighted_length(["0", "11", "10"], [5, 1, 3])
13
```

ב. כיתבו פונקציה optimal(D, W) שמקבלת אותו קלט כמו בסעיף הקודם, ומחזירה True אם הקידוד D ביחס למשקלים W הוא קידוד אופטימלי, כלומר לא קיים קידוד אחר 'D עבורו האורך הממושקל קטן יותר (ביחס לאותם משקלים).
למשל:

```
>>> optimal(["0", "11", "10"], [5, 1, 3])
True
```

ג. להלן שני קידודים עבור האי"ב {"a", "b", "c", "d"}:

a. $D_1 = \{0, 100, 101, 11\}$

b. $D_2 = \{00, 01, 10, 11\}$

תנו דוגמה לקורפוס שמכיל את כל אחד מארבעת תווי האי"ב, שביחס אליו לשני הקידודים הללו אותו אורך ממושקל. את הדוגמה תנו כערך שמוחזר מהפונקציה corpus(), זאת משום שנרצה לבדוק את הפתרון לסעיף זה באופן אוטומטי. למשל, אם לדעתכם קורפוס מתאים הוא "aabcd" השלימו את corpus כך:

```
def corpus():
    return "aabcd"
```

שאלה 3 – למפל זיו

השאלה עוסקת בשינוי באלגוריתם למפל-זיו לדחיסת טקסט.

תזכורת: הפונקציה lz77_compress2 (שלמדנו בכיתה) מחזירה את ייצוג הביניים של דחיסת למפל-זיו של המחרוזת text. למשל:

```
>>> lz77_compress2("abcdabc")
['a', 'b', 'c', 'd', [4, 3]]
>>> lz77_compress2("ababab")
['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 max_length, w של שתי הפונקציות.

דוגמאות הרצה:

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

1. בשלב הראשון עליכם לממש את הגירסה המעט שונה של אלגוריתם למפל זיו , שתופעל ע״י הפונקציה lz77_compress2_no_future . בגירסה זו בכל שלב בו תחושב ההתאמה המקסימלית, תווים שנמצאים באינדקס הנוכחי ואילך לא יילקחו בחשבון בחישוב ההתאמה (ראו הסבר נוסף עם דוגמת ההרצה). הפונקציה lz77_compress2_no_future שנמצאת בקובץ השלד ואין לשנותה (!) קוראת לפונקציה maxmatch_new שמופיעה גם היא בקובץ השלד ואותה עליכם לממש. maxmatch_new מקבלת את אותם הפרמטרים שמקבלת maxmatch.

דוגמת הרצה:

```
>>> s = "aaaaaa"
>>> lz77_compress2_no_future(s)
['a', 'a', 'a', [3, 3]]
```

שימו לב שהפלט המתקבל מהגירסה הרגילה של למפל-זיו הינו:

```
>>> lz77_compress2(s)
['a', [1, 5]]
```

כלומר בגירסה הרגילה של למפל זיו לאחר שכתבנו את התו s[0] , בשלב שבו מטפלת באינדקס 1, נמצא התאמה בין המחרוזת s[1:5], לבין המחרוזת s[0:4]. שימו לב ש s[0:4] אינה מסתיימת לפני אינדקס 1 (כלומר היא חורגת ״אל העתיד״). בשקף 11 בהרצאה 20 יש טענה דומה עבור מחרוזת כללית.

בגירסה החדשה שתממשו (ותופעל ע"י lz77_compress2_no_future), תווים שנמצאים באינדקס הנוכחי ואילך לא יילקחו בחשבון בחישוב ההתאמה המקסימלית ולכן תתקבל התוצאה [['a','a','a', [3,3]].

2. טענה: קיימת מחרוזת s לא ריקה שמקיימת

```
len(inter_to_bin(lz77_compress2_no_future(s))) ==
len(inter_to_bin(lz77_compress2(s)))
```

תנו דוגמא למחרוזת s אם לדעתכם הטענה נכונה. את הדוגמה תנו כערך שמוחזר מהפונקציה lz_Q1(), זאת משום שנרצה לבדוק את הפתרון לסעיף זה באופן אוטומטי. אם לדעתכם דוגמא מתאימה היא המחרוזת "aabcd" אז השלימו את lz_Q1 כך:

```
def lz_Q1():
    return "aabcd"
```

אם לעומת זאת הטענה אינה נכונה לדעתכם, אז ממשו את lz_Q1 באופן הבא:

```
def lz_Q1():
    return None
```

3. טענה: קיימת מחרוזת s לא ריקה שמקיימת

```
len(inter_to_bin(lz77_compress2_no_future(s))) >
len(inter_to_bin(lz77_compress2(s)))
```

תנו דוגמא למחרוזת s אם לדעתכם הטענה נכונה. את הדוגמה תנו כערך שמוחזר מהפונקציה lz_Q2(), זאת משום שנרצה לבדוק את הפתרון לסעיף זה באופן אוטומטי. אם לדעתכם דוגמא מתאימה היא המחרוזת "aabcd" אז השלימו את lz_Q2 כך:

```
def lz_Q2():
    return "aabcd"
```

אם לעומת זאת הטענה אינה נכונה לדעתכם, אז ממשו את lz_Q2 באופן הבא:

```
def lz_Q2():
    return None
```

4. טענה: קיימת מחרוזת s לא ריקה שמקיימת

```
len(inter_to_bin(lz77_compress2_no_future(s))) <
len(inter_to_bin(lz77_compress2(s)))
```

תנו דוגמא למחרוזת s אם לדעתכם הטענה נכונה. את הדוגמה תנו כערך שמוחזר מהפונקציה lz_Q3(), זאת משום שנרצה לבדוק את הפתרון לסעיף זה באופן אוטומטי. אם לדעתכם דוגמא מתאימה היא המחרוזת "aabcd" אז השלימו את lz_Q3 כך:

```
def lz_Q3():
    return "aabcd"
```

אם לעומת זאת הטענה אינה נכונה לדעתכם, אז ממשו את lz_Q3 באופן הבא:

```
def lz_Q3():
    return None
```

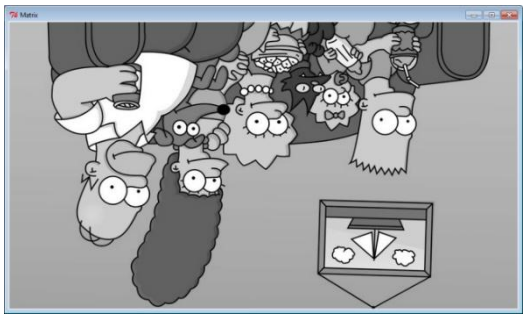
שאלה 4- תמונות

שאלה זו עוסקת בעיבוד תמונה.

שימו לב שעל מנת להריץ את הפונקציות בשני הסעיפים הבאים עליכם לדאוג שבתיקיה (folder) בה נמצא קובץ הקוד שלכם נמצא גם הקובץ matrix.py . בנוסף יהיה עליכם להוסיף את הפקודה הבאה לקובץ השלד:

```
from matrix import *
```

1. השלימו בקובץ השלד את שלוש השורות החסרות בפונקציה upside_down, שמקבלת מטריצה שמייצגת תמונה ומחזירה מטריצה חדשה שמייצגת את התמונה שמתקבלת ע״י שיקוף התמונה על הציר האופקי.
לדוגמא עבור התמונה: לאחר הפעלת הפונקציה תתקבל התמונה:



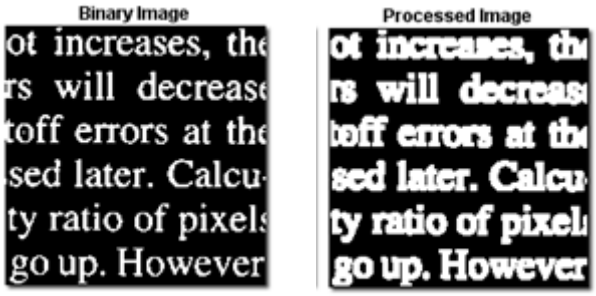
2. **סגמנטציה** של תמונה היא חלוקתה למקטעים (סגמנטים) של פיקסלים, כאשר המקטעים זרים ומכסים את כל התמונה. לפיקסלים באותו סגמנט יש בד״כ תכונות משותפות. מטרת הסגמנטציה היא לפשט את הייצוג של התמונה לאוסף של אובייקטים בעלי משמעות, כדי להקל על עיבוד התמונה לצרכים שונים (כגון מציאת גבולות של אובייקטים בתמונה,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

3. **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*חציון* 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):



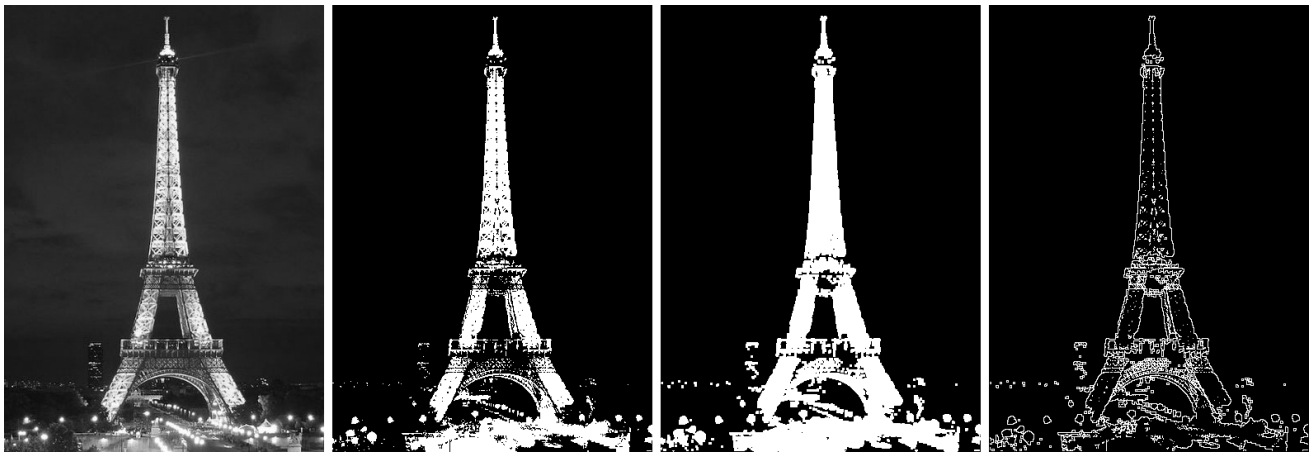
Dilation

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:

Dilation Original Edges

דוגמה נוספת בעמוד הבא.

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) :



A B C D

א. 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.

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

ג. 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
```

שאלה 5 – קודים לתיקון שגיאות - לא להגשה

חלק ראשון

הקוד לתיקון טעויות המתואר כאן מעתיק 3 ביטים של אינפורמציה למילות קוד בנות 7 ביטים, על פי הסכמה הבאה:

$(x_1, x_2, x_3) \rightarrow (x_1, x_2, x_3, x_1+x_2, x_1+x_3, x_2+x_3, x_1+x_2+x_3)$

כאשר הסכומים מחושבים מודולו 2.

א. בטבלה הבאה, השלימו בכל שורה את מילת הקוד המתקבלת מ- 3 הביטים הרשומים בה.

(x_1, x_2, x_3)	$(x_1, x_2, x_3, x_1+x_2, x_1+x_3, x_2+x_3, x_1+x_2+x_3)$
(0 , 0 , 0)	
(0 , 0 , 1)	
(0 , 1 , 1)	
(1 , 1 , 1)	

ב. מהו המרחק המינימלי, d, של הקוד? רשמו שתי מילות קוד שונות w_1, w_2, שהמרחק ביניהן הוא d.

ג. <u>טענה</u>: קיימת מילה $y \in \{0,1\}^7$ כך שיש שתי מילות קוד שונות w_1, w_2, המקיימות: המרחק של שתיהן מ- y שווה, ומרחק זה הוא המרחק המינימלי מ- y למילת קוד כלשהי.

החליטו אם הטענה הנ"ל נכונה. אם לדעתכם הטענה נכונה – תנו דוגמה ל- y, w_1, w_2 כאלו. אחרת – הסבירו מדוע לא.

חלק שני

להלן פונקציית קידוד עבור קוד חדש בשם bad_coding , המקבלת רשימת ביטים x ומוציאה רשימת ביטים.

```
def bad_coding(x):
    z = (x[0]+x[1]) % 2
    return (x+[z])*4
```

תזכורת: קוד $C : \{0,1\}^k \rightarrow \{0,1\}^n$ עם מרחק מינימלי d נקרא קוד מטיפוס $[n, k, d]$.

האורך של x יסומן כרגיל ב- |x|.

השלימו את המשפט הבא: bad_coding הוא קוד מטיפוס [n=____ , k=_____ , d=_____].

סוף