אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, חורף 2021

תרגיל בית מספר 6 - להגשה עד 17.1.21 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 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.

- 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.

<u>שאלה 1 – גנרטורים</u>

א. ממשו גנרטור בעל החתימה take_only(gen, predicate, n) שמקבל גנרטור gen, פונקציה predicate המקבלת ארגומנט יחד ומחזירה True/False (פונקציה כזאת נקראת פרדיקט) ומספר שלם אי-שלילי n. על הגנרטור לייצר את האיברים של gen לפי סדרם אם הם עונים על התנאי predicate. כלומר take_only מייצר תת-סדרה של איברי gen המכילה רק איברים x עבורם predicate(x)==True. אם n איברים רצופים אינם עונים לתנאי, take_only מפסיק לייצר איברים ומסיים את הריצה.

<u>שומו לב: אתם לא נדרשים לטפל ב- StopIteration, היא נזרקת עם סיום ריצת גנרטור.</u>

לדוגמא:

```
>>> list(take_only((i for i in range(30)), lambda x:x%3==1,5))
[1,4,7,10,13,16,19,22,25,28]
>>> list(take_only((i for i in range(30)), lambda x:x<10 or x%7==0,5))
[0,1,2,3,4,5,6,7,8,9,14]
>>> my_gen = take_only((i for i in range(30)), lambda x:x>10 or x==0,5))
>>> for g in my_gen:
>>>    print(g)
0
```

ב. ממשו גנרטור בעל החתימה blocks(gen,k) שמקבל גנרטור gen ומספר k ומחזיר רשימות של איברים עוקבים מתוך gen שאורכן k. למשל, שלושת האיברים הראשונים שיוחזרו ע״י blocks עם k=3 הם: $[a_0, a_1, a_2], [a_3, a_4, a_5], [a_6, a_7, a_8]$ כאשר ... a_0, a_1, a_2 הם איברי gen . אם gen מייצר מס׳ איברים סופי שאינו מתחלק ב-k, blocks יחזיר בלוק אחרון שאורכו קצר יותר מ-k. לדוגמא:

```
>>> list(blocks((i for i in range(10)),5))
[[0,1,2,3,4],[5,6,7,8,9]]
>>> list(blocks((i for i in range(10)),3))
[[0,1,2],[3,4,5],[6,7,8],[9]]
>>> list(blocks((i for i in range(9)),3))
[[0,1,2],[3,4,5],[6,7,8]]
```

<u>שימו לב:</u> בשני הסעיפים הגנרטורים שניתנים כקלט אינם בהכרח סופיים, ובמקרה כזה המרה ל- list כמו בחלק מהדוגמאות לעיל לא תחזיר פלט ותגרום לתקיעת התוכנית.

<u>שאלה 2 – קוד האפמן</u>

א. מצאו את קוד האפמן האופטימלי עבור הקורפוס (corpus) הבא:
a: 1 b: 1 c: 2 d: 3 e: 5 f: 8 g: 13 h: 21

סדרת התדירויות הנ"ל מבוססת על 8 מספרי פיבונאצ'י הראשונים.

ב. הכלילו את תשובתכם מסעיף א' למציאת קוד האפמן אופטימלי כאשר התדירויות הן n מספרי פיבונאצ'י הראשונים. נמקו בקצרה, ללא צורך בהוכחה מפורטת, מדוע הכללה זו נכונה.

ג. נתון קובץ שמכיל תווים מתוך אלפבית בן 256 תווים. בנוסף נתון קורפוס עם תדירויות: $a_1 < a_2 < \cdots < a_n$ (כאשר $n = 256$) ומתקיים: $a_n < 2a_1$.

תהי a_1 תדירות התו p (התדירות המינימלית), ותהי a_n תדירות התו q (התדירות המקסימלית).
יהיו $C(a_n),\ C(a_1)$ קודי ההאפמן שמתקבלים עבור התווים q, p בהתאמה.

מהו ההפרש בין $|C(a_1)|$ (מספר הביטים שדרושים כדי לקודד את התו p) לבין $|C(a_n)|$ (מספר הביטים שדרושים כדי לקודד את התו q)?

על תשובתכם להיות מנומקת!

ד. כיצד תשתנה תשובתכם לסעיף ג אם עכשיו n = 300? הסבירו בקצרה.

ה. כיצד תשתנה תשובתכם לסעיף ג אם נתון ש n = 272 ובנוסף שהתדירויות מקיימות את התנאים הבאים:

a. $a_{16} < 2a_1$

b. $a_{272} < 2a_{17}$

c. $16a_{16} < a_{17}$

הסבירו את תשובתכם ושרטטו סקיצה שממחישה את מבנה עץ האפמן עבור קורפוס זה.

המלצה: ברוב הסעיפים ניתן להריץ דוגמאות קוד כדי להשתכנע שהתשובה שלכם נכונה. אל תפספסו את ההזדמנות לוודא זאת.

<u>שאלה 3 – למפל זיו</u>

השאלה עוסקת בשינוי באלגוריתם למפל-זיו לדחיסת טקסט.

כזכור, הפונקציה LZW_compress מחזירה את ייצוג הביניים של דחיסת למפל-זיו של המחרוזת text. למשל:

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

בנוסף, ראינו בכיתה את הפונקציה:

```
def inter_to_bin(intermediate, W=2**12-1, L=2**5-1)
```

שבהינתן רשימה lst עם ייצוג ביניים של מחרוזת דחוסה, מחזירה מחרוזת של ביטים, המייצגת את רצף הביטים לאחר הדחיסה. נזכיר, שתו שלא נדחס ייוצג ע"י הביט 0 ואחריו 7 ביטים עבור התו עצמו (סה"כ 8 ביטים, כלומר גם בתרגיל זה אנחנו מניחים לשם פשטות כי אנחנו מטפלים רק בתווי ASCII), 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,L של שתי הפונקציות.

דוגמאות הרצה:

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

בעמוד הבא מופיעה הפונקציה **LZW_compress_new** שמציגה מימוש של אלגוריתם שונה במעט עבור דחיסת למפל זיו.

ענו בקובץ ה pdf על הסעיפים הבאים ביחס ל: **LZW_compress_new, LZW_compress.**

א. תנו דוגמא למחרוזת s המקיימת:

LZW_compress(s) = LZW_compress_new(s).

מה יהיה הפלט (ייצוג הביניים) שיתקבל בשתי ההרצות?

ב. טענה: קיימת מחרוזת **s** שמקיימת:

len(inter_to_bin(LZW_compress_new(s))) <
len(inter_to_bin(LZW_compress(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(LZW_compress_new(s))) >
len(inter_to_bin(LZW_compress(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.

```
def LZW_compress_new(text, start=0, W=2**12-1, L=2**5-1):
    n = len(text)
    if start >= n:
        return []

    #find the maximal length matching
    m,k = maxmatch(text, start, W, L)

    res1 = [text[start]] + \
          LZW_compress_new(text, start+1, W, L)
    res1_len = len(inter_to_bin(res1, W, L))

    if k < 3:
        return res1

    res2 = [[m,k]] + LZW_compress_new(text, start+k, W, L)
    res2_len = len(inter_to_bin(res2, W, L))

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

<u>שאלה 4 – קודים לתיקון שגיאות</u>

<u>חלק ראשון</u>

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 כל הסכומים מחושבים מודולו 2 (כלומר כל איברי השביעיה הם 0 או 1).

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. נמקו בקצרה מדוע אין מילות קוד במרחק קטן יותר.

ג. <u>טענה</u>: קיימת מילה $y \in \{0,1\}^7$, לא בהכרח מילת קוד, 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.

<u>חלק שני</u>

להלן פונקציית קידוד עבור קוד חדש בשם bad_coding , המקבלת רשימת ביטים x באורך כלשהו לא ידוע, 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=_____].

<u>שאלה 5– CYK</u>

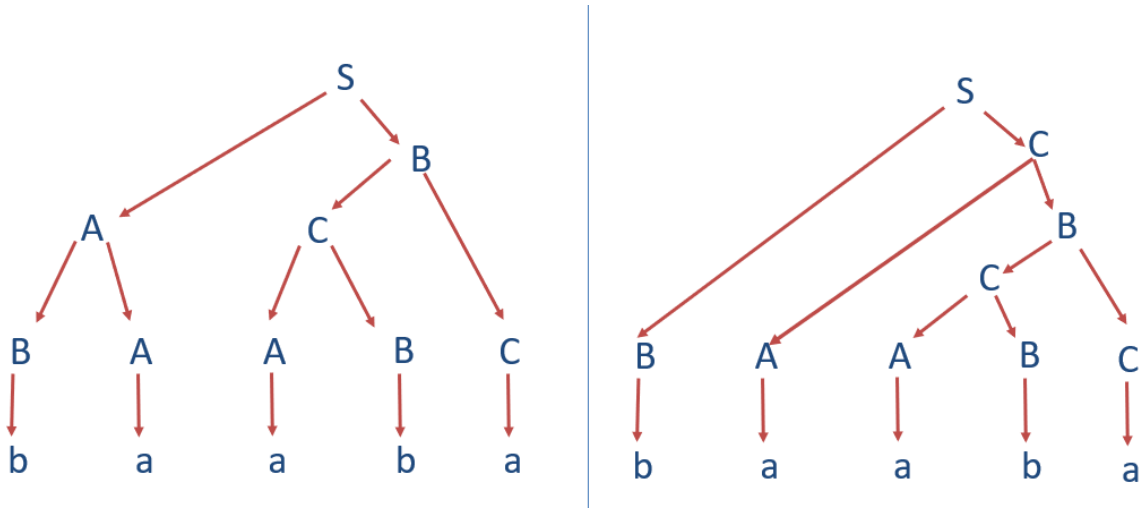
בשאלה זו נבחן את הקוד של אלגוריתם CYK שראיתם בהרצאה ובתרגול. להזכירכם, בהינתן דקדוק G בצורת CNF ומחרוזת st, אלגוריתם CYK מחזיר האם המחרוזת יכולה להגזר ע"י הדקדוק, או במילים אחרות, האם st בשפה של G: $st \in \mathcal{L}(G)$.

א. ראינו בתרגול שסיבוכיות הזמן של אלגוריתם CYK חסומה ע"י $O(n^3|R|)$ במקרה הממוצע ו- $O(n^3|R|^2)$ במקרה הגרוע ביותר, כאשר $n = |st|$ הוא אורך המחרוזת ו- $|R|$ הוא מספר חוקי הגזירה בדקדוק.בניתוח הסיבוכיות בתרגול הנחנו שמספר החלוקות הלא טריוויאליות בכל תא הוא O(n). חסם זה אינו הדוק (ישנם תאים בהם מספר החלוקות קטן משמעותית). נתחו מחדש את מספר החלוקות הלא טריוויאליות הכולל עבור כל התאים והסיקו כי סיבוכיות האלגוריתם במקרה הממוצע היא $\Theta(n^3|R|)$ ולא רק $O(n^3|R|)$.

ב. ראינו כי יש זוגות של (דקדוק, מחרוזת) עבורם יש יותר מעץ גזירה אחד שגוזר את המחרוזת. מצב זה נקרא ambiguity או עמימות.
בסעיף זה, נרצה לערוך שינוי קטן בCYK כך שבהינתן דקדוק ומחרוזת, האלגוריתם יחזיר את **העומק המינימלי** של עץ גזירה שגוזר את המחרוזת st, או 1- אם המחרוזת לא יכולה להיגזר ע"י הדקדוק.
האלגוריתם לאחר השינוי צריך להיות באותה סיבוכיות הזמן והמקום של האלגוריתם המקורי.
לדוגמה, עבור הדקדוק

$$S \to AB \mid BC$$
$$A \to BA \mid a$$
$$B \to CC \mid b$$
$$C \to AB \mid a$$

והמחרוזת $st = "baaba"$, שני עצי הגזירה הנ"ל הם חוקיים:



אך עומק עץ הגזירה השמאלי הוא 4 ועומק עץ הגזירה הימני הוא 5. במקרה זה, 4 הוא העומק המינימלי של עץ גזירה שגוזר את המחרוזת, ולכן עבור דקדוק ומחרוזת אלו האלגוריתם שלכם צריך להחזיר 4.

עבור אותו הדקדוק ומחרוזת $st = "baab"$, האלגוריתם צריך להחזיר 1- מכיוון שהמחרוזת אינה בשפה של הדקדוק.

בקובץ השלד תמצאו קוד חלקי לפונקציות CYK_d, fill_cell_d, fill_length_1_cells_d. השלימו את הקוד במקומות המסומנים כך שהפונקציה CYK_d תעבוד כנדרש.

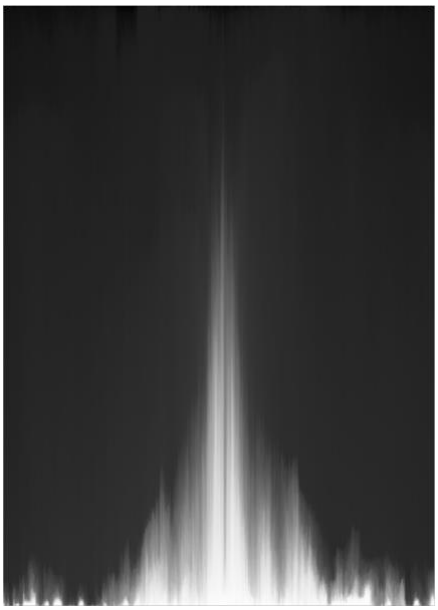
דוגמאות הרצה:

```
>>> rule_dict = {"S": {"AB", "BC"}, "A": {"BA", "a"}, "B": {"CC",
"b"}, "C": {"AB", "a"}}
>>>> CYK_d("baaba", rule_dict, "S")
4
>>>> CYK_d("baab", rule_dict, "S")
-1
```

<u>שאלה 6 (בונוס) – ייצוג ועיבוד תמונה דיגיטלית</u>

א. צוות הקורס כתב פונקציה מסתורית בשם what, המקבלת תמונה (אובייקט של PIL.Image, כפי שראינו בשיעור), ומחזירה עותק מעובד של התמונה. להלן הרצה על תמונת מגדל אייפל איתה עבדנו בשיעור, ותמונת הפלט. נסו להבין מה עושה הפונקציה what, וממשו אחת כזו בעצמכם (כלומר ממשו פונקציה שתיתן בדיוק אותו פלט).

```
>>> img = Image.open("./guess.bmp").convert('L')
>>> what(img).show()
```



ב. להלן כמה שינויים בפונקציה process_img (נעבור עליה בשיעור ביום רביעי).

```
def process_img(img, op):

    w,h = img.size
    mat = img.load()
    ~~new_img = img.copy()~~  #DELETED
    ~~new_mat = new_img.load()~~  #DELETED

    for x in range(w):
        for y in range(h):
            ~~new_mat[x,y] = op(mat, x, y)~~ #REPLACED WITH THE LINE
BELOW
           mat[x,y] = op(mat, x, y)

    ~~return new_img~~ #REPLACED WITH THE LINE BELOW
    return img
```

הריצו את הפקודה הבאה המשתמשת ב- process_img על תמונה כלשהי לבחירתכם, פעם אחת עם process_img לפני השינוי הנ"ל ופעם אחת אחרי, וצרפו את התמונות המקבלות. הסבירו בקצרה את התוצאות.

```
upside_down = process_img(img, lambda mat, x, y: mat[x,h-y-1])
```

<u>סוף</u>