אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2022

תרגיל בית מספר 4 - להגשה עד 08/05/2022 בשעה 23:59

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

<u>הנחיות לצורת ההגשה</u>:

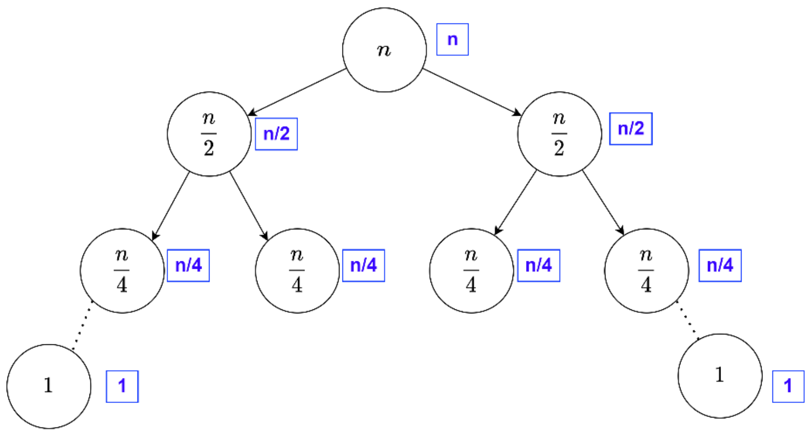
- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם hw4_012345678.pdf ו- hw4_012345678.py.
- עבור קובץ ה-pdf: מומלץ מאוד להקליד את התשובות בו. ניתן גם לכתוב את התשובות בכתב יד ברור ולסרוק אותן, אבל שימו לב שתשובות שיכתבו בכתב יד לא ברור לא יבדקו, וערעורים בנושא זה לא יתקבלו.
- עבור קובץ ה-py: השתמשו בקובץ השלד skeleton4.py כבסיס לקובץ אותו אתם מגישים.
- לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת py.

<u>הנחיות לפתרון</u>:

- הקפידו לענות על כל מה שנשאלתם.
- בכל שאלה, אלא אם מצוין אחרת באופן מפורש, ניתן להניח כי הקלט תקין.
- אין להשתמש בספריות חיצוניות פרט לספריות math, random, אלא אם נאמר במפורש אחרת.
- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים.
  להנחיה זו מטרה כפולה:
  i. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  ii. 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.
- נדרוש שכל הפונקציות שאנו ממשים תהיינה יעילות ככל הניתן. לדוגמה, אם ניתן לממש פתרון לבעיה בסיבוכיות $O(logn)$, ואתם מימשתם פתרון בסיבוכיות $\Theta(n)$, תקבלו ניקוד חלקי על הפתרון.
- בשאלות שבהן ישנה דרישה לניתוח סיבוכיות זמן הריצה, הכוונה היא לסיבוכיות זמן הריצה של המקרה הגרוע ביותר.

<u>הערות כלליות לתרגיל זה ולתרגילים הבאים:</u>

כאשר אתם מתבקשים לצייר עץ רקורסיה, מומלץ להיעזר בכלים דיגיטליים כמו draw.io כדי לצייר את העץ. עם זאת, ניתן לצייר את עצי הרקורסיה בכתב יד ולסרוק, כל עוד הציור ברור לחלוטין. **<u>ציור שאינו ברור לא ייבדק.</u>**
בציורכם הקפידו על הדברים הבאים: כל צומת בעץ מייצג קריאה רקורסיבית – בתוך הצומת כתבו את הקלט, או את אורך הקלט, המתאים לצומת זה, ולצד כל צומת כתבו את כמות העבודה שמתבצעת בצומת. לדוגמה, את עץ הרקורסיה עבור המקרה הטוב ביותר של Quicksort (כפי שראיתם בהרצאה) נצייר באופן הבא:



שאלה 1

בשאלה זו כל סעיף יעסוק בסוגיה הקשורה לקוד שראיתם בהרצאה או בתרגול. מומלץ לעבור על החומר הרלוונטי מההרצאה / תרגול לפני שאתם ניגשים לפתור את הסעיף.

סעיף א' - מקסימום של רשימה (תרגול 6)
ראיתם בתרגול מימוש רקורסיבי למציאת מקסימום של רשימה. במימוש זה השתמשנו במשתנים left, right כדי להימנע מפעולות slicing על הרשימה. לפניכם שלושה מימושים שונים לחישוב מקסימום של רשימה באופן רקורסיבי. הראשון דומה למימוש שראינו בכיתה, אך עם שימוש ב-slicing. השניים האחרים הם מימוש אחר לחישוב מקסימום של רשימה – עם ובלי slicing.
נסמן ב-n את אורך הרשימה L. נתחו את סיבוכיות זמן הריצה של שלוש הפונקציות כתלות באורך הרשימה n.

.i

```
def max_v1(L):
    if len(L) == 1:
       return L[0]

    mid = len(L) // 2
    first_half = max_v1(L[:mid])
    second_half = max_v1(L[mid:])

    return max(first_half, second_half)
```

.ii

```
def max_v2(L):
    if len(L) == 1:
       return L[0]

    without_left = max_v2(L[1:])

    return max(without_left, L[0])
```

.iii

```
def max_v3(L, left):
    if left == len(L)-1:
       return L[left]

    without_left = max_v3(L, left + 1)

    return max(without_left, L[left])
```

סעיף ב' – בעיית העודף (change) (תרגול 6-7)
בסעיף זה נרצה לממש וריאציה של בעיית העודף שראיתם בתרגול. במקום להחזיר את כמות הדרכים שניתן לפרוט את amount בעזרת המטבעות coins, נרצה להחזיר רשימה של כל הפריטות האפשריות.
ממשו את הפונקציה change_v2(amount, coins) אשר מקבלת מספר amount שלם אי שלילי ורשימה coins של מספרים שלמים חיוביים, ומחזירה רשימה של רשימות המייצגות את כל הפריטות החוקיות של amount בעזרת המטבעות coins. הסדר של הרשימה וכן הסדר הפנימי של תתי הרשימות אינו משנה, אבל יש להקפיד שכל פריטה חוקית מופיעה בדיוק פעם אחת.

דוגמת הרצה:

```
>>> change_v2(5, [1,2,3])
[[1,1,1,1,1],[1,2,2],[3,2],[1,3,1],[1,1,1,2]]

>>> change(amount, coins) == len(change_v2(5,[1,2,3])
True     # This should be True for any amount & coins
```

<u>סעיף ג' - המשחק זלול! (munch) (הרצאה 12)</u>

i. לפניכם וריאציה של הקוד שראיתם בכיתה שמחזיר True אם הלוח board במצב מנצח של משחק munch, או False אם הלוח במצב מפסיד. בקוד מספר שינויים המודגשים בצהוב.
<u>הסבירו מה יודפס</u> כאשר נריץ את הפקודה הבאה:

winnable([2,1,1], show=True, player=0)

עבור הרצה זו **<u>ציירו</u>** את עץ הרקורסיה במלואו (באופן דומה לציור שראיתם בהרצאה) **<u>וסמנו על העץ</u>** את הצמתים שבהם יתבצעו ההדפסות.

```
def winnable(board, show=False, player=0):
    if sum(board) == 0:
        return True

    m = len(board)

    for i in range(m):
        for j in range(board[i]):
            munched_board = board[0:i] + [min(board[k], j) for k in range(i, m)]

            if not winnable(munched_board, show, 1-player):
                if show and player == 0:
                    print("recommended move:", board, "-->", munched_board)
                return True

    return False
```

ii. בסעיף זה תוסיפו ממואיזציה לקוד שראיתם בכיתה (ולא לקוד מסעיף i). בקובץ השלד ממומשת עבורכם פונקציית המעטפת winnable_mem(board) המאתחלת מילון ריק ומבצעת קריאה לפונקציה winnable_mem_rec.
ממשו את הפונקציה הרקורסיבית winnable_mem_rec(board, d), כך שבהינתן רשימה board המייצגת לוח חוקי של מאנץ', פונקציית המעטפת תחזיר True אם הלוח במצב מנצח, או False אם הלוח במצב מפסיד. על הפונקציה להשתמש במילון d כדי לחסוך קריאות רקורסיביות חוזרות על קלטים שחושבו קודם לכן (כלומר, על הפונקציה להשתמש בממואיזציה). כמו כן, שימו לב <u>לא לבצע הדפסות</u> בשום שלב במימוש שלכם (בשונה מהקוד שראיתם בכיתה).
<u>תזכורת</u>: מפתחות במילון חייבים להיות אובייקטים מסוג immutable. בפרט, לא ניתן להשתמש באובייקט מטיפוס רשימה כמפתח במילון (נבין מדוע המגבלה הזו קיימת בהמשך הקורס). ניתן להמיר את הרשימה לאובייקט אחר שהוא immutable (לדוגמה tuple), ולהשתמש בו כמפתח במילון.

iii. עבור הפונקציה שמימשתם, ועבור כל אחד מן הקלטים [5]*8 , [5]*16, כתבו בקובץ ה-PDF בטבלה:
1) כמה חיפושים בדיוק נעשו במילון בסך הכל לאורך החישוב של הפונקציה?
(כל פקודה מהצורה "if key in d", "if d[key] == val" וכדומה, מהווה חיפוש במילון)
2) מתוך כל החיפושים – כמה חיפושים בדיוק הסתיימו בהצלחה? כלומר כמה פעמים החיפוש החזיר ערך שחושב קודם לכן?

<u>הערה</u>: נסו לערוך את הקוד שכתבתם בסעיף הקודם כדי לחשב ערכים אלה. אין להגיש את הקוד הערוך, אלא רק את הקוד שמימשתם בסעיף הקודם.

שאלה 2

גרף $G = (V, E)$ מוגדר על ידי קבוצה של n **קודקודים**, נסמנה $V = \{0,1, \ldots, n-1\}$, וקבוצת **קשתות** $E \subseteq V \times V$ של זוגות קודקודים. בהינתן גרף $G = (V, E)$ עם n קודקודים, **מטריצת השכנויות A** של הגרף G היא מטריצה בגודל $n \times n$ אשר מקיימת:

$$A_{ij} = \begin{cases} 1\ if\ (i,j) \in E \\ 0\ if\ (i,j) \notin E \end{cases}$$

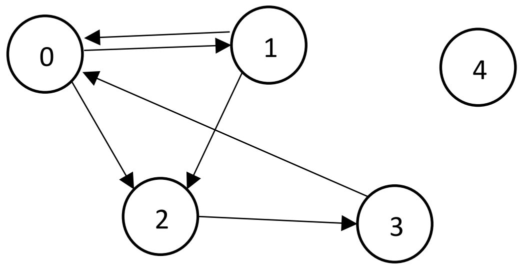
במילים, לכל זוג קודקודים $i, j \in V$ מתקיים $A_{ij} = 1$ אמ"ם יש קשת בגרף מ-i ל-j.
לדוגמה, עבור הקודקודים $V = \{0,1,2,3,4\}$ והקשתות $E = \{(0,1), (1,0), (1,2), (0,2), (2,3), (3,0)\}$:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

נייצג את המטריצה בפייתון על ידי רשימה של רשימות, כך שכל תת רשימה תייצג שורה במטריצה. לדוגמה, את המטריצה הנ"ל נייצג בפייתון על ידי הרשימה:

$$A = [[0,1,1,0,0], [1,0,1,0,0], [0,0,0,1,0], [1,0,0,0,0], [0,0,0,0,0]]$$

דרך נוחה לייצג גרף באופן ויזואלי היא באמצעות ציור הקודקודים כמעגלים, וציור הקשתות כחצים בין המעגלים. למשל את הדוגמה הנ"ל ניתן לצייר באופן הבא:



נאמר שקיים בגרף **מסלול באורך k** בין קודקוד s לקודקוד t, אם קיימת סדרה של קשתות $e_1, \ldots, e_k \in E$ מ-s ל-t.
באופן יותר פורמלי, אם נסמן $e_i = (v_i, u_i)$, אז לכל $1 \le i < k$ צריך להתקיים ש- $u_i = v_{i+1}$, וכן $v_1 = s$ ו-$u_k = t$.
באופן שקול, ניתן לחשוב על מסלול כעל סדרת קודקודים $s, v_1, \ldots, v_{k-1}, t$ כך שבין כל זוג עוקב של קודקודים יש קשת בגרף.
נגדיר מסלול באורך $k = 0$ באופן טבעי להיות מסלול ריק (כלומר, יש מסלול באורך 0 בין s ל-t אמ"ם $s = t$), ומסלול באורך $k = 1$ להיות קשת בגרף (כלומר יש מסלול באורך 1 מ-s ל-t אמ"ם הקשת (s,t) בגרף).

<u>הערות</u>:

- ההגדרה לעיל היא של **גרף מכוון**. מקרה פרטי של גרף מכוון הוא **גרף לא מכוון** שבו לכל קשת $(u,v) \in E$, גם $(v,u) \in E$. בשאלה זו נעסוק במקרה הכללי יותר, כלומר בגרפים מכוונים.
- לאורך כל השאלה נניח כי בגרפים **אין קשתות עצמות**, כלומר לכל i מתקיים ש-$(i,i) \notin E$.
- לאורך השאלה נסמן ב-n את מספר הקודקודים בגרף, וב-k אורך של מסלול בגרף. ניתן להניח כי $k \le n$.

<u>סעיף א'</u>

ממשו את הפונקציה $legal\_path(A, vertices)$ המקבלת מטריצת שכנויות A של גרף כלשהו, ורשימה של קודקודים $vertices$ של קודקודים בגרף, ומחזירה True אם רשימת הקודקודים מהווה מסלול בגרף, ו- False אחרת. ניתן להניח כי הגרף אינו ריק.

דוגמאות הרצה (A היא מטריצת השכנויות של הגרף המתואר מעלה):

```
>>> A = [[0,1,1,0,0], [1,0,1,0,0], [0,0,0,1,0], [1,0,0,0,0],[0,0,0,0,0]]
>>> legal_path(A, [0, 1, 2, 3])
True
>>> legal_path(A, [0, 1, 2, 3, 0, 1])
True
>>> legal_path(A, [0, 1, 2, 3, 4])
False
```

<u>סעיף ב׳</u>

מיכל רצתה לממש פונקציה אשר בודקת האם קיים מסלול באורך k בין שני קודקודים s,t בגרף. לשם כך, היא מימשה את הפונקציה הרקורסיבית הבאה המקבלת מטריצת שכנויות A, שני קודקודים s ו-t, ומספר k, אשר מחזירה True אמ״ם קיים מסלול באורך k מ-s ל-t:

```
def path_v1(A, s, t, k):
    if k == 0:
        return s == t

    for i in range(len(A)):
        if A[s][i] == 1:
            if path_v1(A, i, t, k-1):
                return True
    return False
```

i. ציירו את עץ הרקורסיה המתאים עבור ההרצה הבאה (מכיוון ש-A ו-t אינם משתנים לאורך הריצה של הפונקציה, רשמו בכל צומת בעץ את הערכים המתאימים של s ו-k בלבד):

```
>>> A = [[0,1,1,0,0], [1,0,1,0,0], [0,0,0,1,0], [1,0,0,0,0],[0,0,0,0,0]]
>>> path_v1(A, 0, 4, 3)
```

ii. תנו דוגמה לקלט שעבורו זמן הריצה של הפונקציה הוא לפחות אקספוננציאלי במספר הקודקודים n. כלומר, לכל n מצאו קלט כך שזמן הריצה של הפונקציה הוא לפחות $c\cdot 2^n$ עבור c קבוע כלשהו. **<u>הסבירו</u>** את תשובתכם.
<u>הערה</u>: ניתן למצוא דוגמה יחסית פשוטה כך שזמן הריצה של הפונקציה הוא לפחות $(n-2)^{n-2}$.

<u>סעיף ג׳</u>

אלחנן ניסה לממש את הפונקציה באופן שונה:

```
def path_v2(A, s, t, k):
    if k == 0:
        return s == t

    # ADD YOUR CODE HERE #

    for i in range(len(A)):
        mid = k // 2
        if path_v2(A, s, i, mid) and path_v2(A, i, t, k - mid):
            return True
    return False
```

i. מיכל הבחינה שהמימוש של אלחנן לא משתמש כלל במטריצת השכנויות של הגרף כדי לבדוק קיומן של קשתות, ולכן לא יתכן שהוא נכון. תנו דוגמה לקלט שעבורו הפונקציה הנ״ל מחזירה פלט לא נכון.

ii. תקנו את הפונקציה בקובץ השלד, על ידי **הוספת קוד בחלק המסומן** בין תנאי ה-if שבראשית הפונקציה לבין לולאת ה-for, וללא מחיקת הקוד הקיים. (שימו לב שלסעיף זה אין בדיקה ב-test שבקובץ השלד כדי לא לחשוף את התשובה ל-i. הקפידו לוודא את נכונות הפתרון שלכם!)

iii. נסמן ב-$f(n)$ את זמן הריצה של הפונקציה במקרה הגרוע על קלט בגודל n.
נאמר שזמן הריצה הוא **סופר-פולינומיאלי** ב-n אם לכל קבוע c, קיים n_0 כך שלכל $n>n_0$ מתקיים $f(n)>n^c$.
הגדרה שקולה היא שלא קיים קבוע c כך ש-$f(n)=O(n^c)$.
לדוגמה, הפונקציה 2^n היא סופר-פולינומיאלית ב-n, כמו גם $n^{\log(n)}$.
תנו דוגמה לקלט שעבורו זמן הריצה של הפונקציה הוא סופר-פולינומיאלי במספר הקודקודים n. **<u>הסבירו</u>** את תשובתכם.
<u>הערה</u>: ניתן למצוא דוגמה יחסית פשוטה כך שזמן הריצה של הפונקציה הוא לפחות $(n-1)^{\log(n-1)}$. שימו לב שזו אכן פונקציה סופר-פולינומיאלית.

<u>סעיף ד'</u>

ננסה כעת לפתור בעיה מעט שונה: בהינתן A מטריצת שכנויות, וקודקודים s,t, האם קיים מסלול **באורך כלשהו** בין s ל-t?
לפניכם מימוש <u>לא נכון</u> לבעיה זו:

```
def path_v3 (A, s, t):
    if s == t:
        return True

    for i in range(len(A)):
        if A[s][i] == 1:
            if path_v3(A, i, t):
                return True
    return False
```

i. לפניכם 4 טענות על הפונקציה path_v3:

a. קיים קלט (A,s,t) כך <u>שיש</u> מסלול בין s ל-t, אך הפונקציה תחזיר False.
b. קיים קלט (A,s,t) כך <u>שיש</u> מסלול בין s ל-t, אך הפונקציה לא תסיים לרוץ, או שתיזרק שגיאת זמן ריצה.
c. קיים קלט (A,s,t) כך <u>שאין</u> מסלול בין s ל-t, אך הפונקציה תחזיר True.
d. קיים קלט (A,s,t) כך <u>שאין</u> מסלול בין s ל-t, אך הפונקציה לא תסיים לרוץ, או שתיזרק שגיאת זמן ריצה.

בקובץ השלד מופיעים ארבעה משתנים בשם "path_v3_X", עבור $X \in \{a,b,c,d\}$, אחד לכל אחת מארבע הטענות הנ"ל. לכל אחד מארבעת המשתנים:

- אם הטענה <u>נכונה</u>: השלימו את ההשמה של המשתנה ל-tuple באורך 3 מהצורה (A,s,t) שמייצג קלט שהפונקציה path_v3 מצפה לקבל (יש לבחור קלט שבו $n = len(A)$ הוא לכל היותר 10), ואשר מוכיח את נכונות הטענה.
לדוגמה, מלאו path_v3_c = ([[0,0],[0,0]], 0, 1) אם אתם טוענים שבגרף המיוצג על ידי [[0,0],[0,0]] אין מסלול מ-0 ל-1, אך הפונקציה תחזיר True על קלט זה.
- אם הטענה <u>לא נכונה</u>: השלימו את ההשמה של המשתנה לערך None.
לדוגמה: path_v3_X = None

ii. תקנו את הקוד על ידי **הוספת קוד בלבד** (ניתן להוסיף את הקוד בכל מקום, אבל אין למחוק קטעי קוד קיימים), כך שסיבוכיות זמן הריצה שלו במקרה הגרוע תהיה $O(n^2)$, כאשר n הוא מספר הקודקודים בגרף.
<u>הסבירו</u> מדוע המימוש שלכם עומד בדרישת הסיבוכיות.

שאלה 3

הנחיות לכל הסעיפים בשאלה זו:

- על הפונקציה שאתם מממשים להיות רקורסיבית, או להיות פונקציית מעטפת שקוראת לפונקציה רקורסיבית.
- ניתן להניח שפעולות אריתמטיות לוקחות זמן קבוע.

בהינתן רשימה L של מספרים שלמים וחיוביים, ומספר שלם s, נאמר ש**ניתן ליצור את s מ-L** אם ניתן להגיע ל-s על ידי חיבור וחיסור של איברי L.

סעיף א'

בסעיף זה, נבדוק האם ניתן ליצור את s מ-L תחת ההגבלה שאנו משתמשים בכל איבר ב-L **בדיוק פעם אחת.**
לדוגמה, אם $L = [5,2,3]$ ו- $s = 6$ אז ניתן ליצור את s מ-L תחת ההגבלה, מכיוון ש-

$$5 - 2 + 3 = 6$$

כמו כן ניתן ליצור מ-L את $s = -10$ מכיוון ש-

$$-5 - 2 - 3 = -10$$

לעומת זאת, לא ניתן ליצור מ-L את $s = 9$ או את $s = 7$ תחת ההגבלה.
ממשו את הפונקציה can_create_once(s, L) אשר מחזירה True אם אפשר ליצור את s מ-L, ו-False אחרת, תחת הגבלה זו. על הפונקציה להיות מסיבוכיות זמן $O(2^n)$ כאשר n הוא אורך הרשימה L. **הסבירו** מדוע הפונקציה שמימשתם עומדת בדרישת הסיבוכיות.

סעיף ב'

בסעיף זה נממש פונקציה דומה לזו מסעיף א', אבל הפעם נרשה להשתמש בכל איבר ב-L **לכל היותר פעמיים.** לדוגמה, אם $L = [5,2,3]$ אז הפעם ניתן ליצור את $s = 9$ תחת הגבלה זו, מכיוון ש-

$$5 + 2 + 2 = 9$$

כמו כן ניתן ליצור את $s = 9$ גם בדרך הבאה

$$5 - 2 + 3 + 3 = 9$$

ממשו את הפונקציה can_create_twice(s, L) אשר מחזירה True אם אפשר ליצור את s מ-L, ו-False אחרת, תחת הגבלה זו. על הפונקציה להיות מסיבוכיות זמן $O(5^n)$ כאשר n הוא אורך הרשימה L. **הסבירו** מדוע הפונקציה שמימשתם עומדת בדרישת הסיבוכיות.

סעיף ג'

מומלץ לקרוא על הפונקציה המובנית eval שיכולה לסייע לכם בסעיף זה.
בהינתן רשימה L של מספרים שלמים חיוביים וביניהם המחרוזות '+', '*', ו-'-', נחשוב על הרשימה כעל ביטוי מתמטי של חיבור, חיסור וכפל בין מספרים. לדוגמה, הרשימה $L = [6\ ,\ '-',\ 4\ ,\ '*'\ ,\ 2\ ,\ '+'\ ,\ 3]$ תייצג את הביטוי:

$$6 - 4 \times 2 + 3$$

בהינתן רשימה L כזו, ומספר שלם s, נרצה למצוא האם יש דרך למקם סוגריים על הביטוי המתמטי שמייצג L כך שערך הביטוי בהתאם לחוקי הקדימות של הסוגריים יהיה שווה ל-s. לדוגמה, עבור ה-L הנ"ל ו-$s = 10$:

$$(6 - 4) \times (2 + 3) = 10$$

כמו כן ניתן להגיע ל $s = 1$ על ידי:

$$\big(6 - (4 \times 2)\big) + 3 = 1$$

ממשו את הפונקציה valid_braces_placement(s, L) המקבלת מספר שלם s ורשימה L כמתואר[1], ומחזירה True אם קיימת השמת סוגריים מתאימה, ואחרת מחזירה False. לשם פשטות הניתוח, נסמן ב-n את כמות המספרים ברשימה L (כלומר, מספר איברי הרשימה לא כולל המחרוזות). על הפונקציה להיות מסיבוכיות זמן $O(n!)$. **הסבירו** מדוע הפונקציה שמימשתם עומדת בדרישת הסיבוכיות.

רמז: בניתוח הסיבוכיות ניתן להיעזר בעובדה שלכל k טבעי קבוע, הטור $\sum_{i=1}^{\infty} \frac{i^k}{i!}$ מתכנס למספר קבוע כלשהו C_k.

בפרט, לכל n טבעי הסכום הסופי $\sum_{i=0}^{n} \frac{i^k}{i!}$ קטן ממש מ-C_k.
שימו לב שלא כל פתרון של השאלה מצריך שימוש בעובדה הזו בניתוח הסיבוכיות.

הערה: לכל k המספר C_k הוא כפולה שלמה של e (בפרט $C_1 = e$). סדרת המספרים השלמים $\left\{\frac{C_k}{e}\right\}_{k=1}^{\infty}$ נקראת סדרת מספרי בל, והיא בעלת משמעות קומבינטורית, ושימושית בתורת ההסתברות ובענפים שונים במתמטיקה. מוזמנים לקרוא עוד כאן וכאן.

[1] באופן יותר מדויק, מתקיים שבאינדקסים הזוגיים ברשימה (מתחילים מ-0) תמיד יהיו מספרים שלמים חיוביים, באינדקסים האי זוגיים יהיו אחת המחרוזות '+', '*', או '-', והרשימה תהיה באורך אי זוגי גדול או שווה ל-1.

<u>שאלה 4</u>

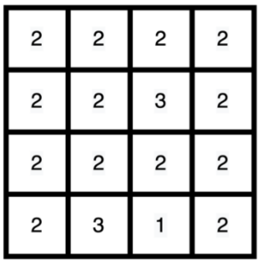
בשאלה זאת נתון לוח דו-ממדי B בגודל n על n אשר מכיל מספרים שלמים אי-שליליים (כולל 0). בדומה למטריצה, הלוח מיוצג ע"י רשימות מקוננות. מטרתנו היא להכריע האם ניתן להגיע מהמשבצת $(0,0)$ למשבצת $(n-1, n-1)$ לפי חוקי המשחק אשר יתוארו בכל סעיף של השאלה.

<u>סעיף א'</u>

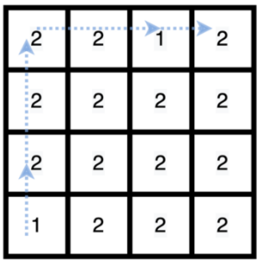
בסעיף זה חוקי ההתקדמות בלוח הם:

- אם אנחנו במשבצת ה (i,j) אז אנחנו יכולים להתקדם "קדימה" או ל $(i+B[i][j], j)$ או ל $(i,\ j+B[i][j])$. כלומר, אנחנו יכולים לזוז בדיוק $B[i][j]$ משבצות <u>בכיוון החיובי</u> של אחד מצירי הלוח.
- אסור לצאת מגבולות הלוח כלומר צריך להישאר בתחום 0 עד $n-1$ בכל ציר.

ממשו את הפונקציה grid_escape1(B) אשר מקבלת לוח B ומחזירה True אם ניתן להגיע מהמשבצת $(0,0)$ למשבצת $(n-1, n-1)$ ו False אחרת. לדוגמא שני לוחות משחק והפלט עבורם (שימו לב שהנקודה $(0,0)$ נמצאת בפינה השמאלית-תחתונה והנקודה $(n-1, n-1)$ נמצאת בפינה הימנית-עליונה:



False



True

בציור השמאלי ניתן להשתכנע שאכן אין מסלול חוקי מ-$(0,0)$ ל-$(3,3)$: אנחנו מתחילים ב-$(0,0)$, ועלינו ללכת $B[0][0] = 2$ צעדים. נניח שבחרנו ללכת שני צעדים למעלה, ל-$(0,2)$. כעת שוב עלינו ללכת $B[0][2] = 2$ צעדים – לא ניתן ללכת למעלה, לכן נלך ימינה ל-$(2,2)$. כעת עלינו ללכת $B[2][2] = 3$ צעדים – אבל אין אף כיוון חוקי ללכת אליו. באופן דומה ניתן להיווכח שכל מסלול אחר מסתיים ללא מוצא, ולכן על הפונקציה להחזיר False.

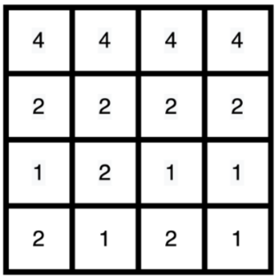
הנחיה: על הפתרון להיות רקורסיבי

<u>סעיף ב'</u>

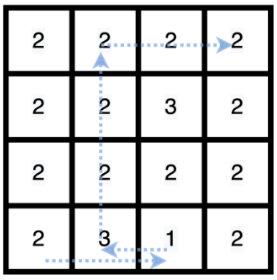
בסעיף זה מותר גם לזוז "אחורה" כך שחוקי ההתקדמות בלוח הם:

- אם אנחנו במשבצת ה i,j אז אנחנו יכולים להתקדם או ל $(i+B[i][j], j)$ או ל $(i,\ j+B[i][j])$ או ל $(i-B[i][j], j)$ או ל $(i,\ j-B[i][j])$. כלומר, אנחנו יכולים לזוז בדיוק $B[i][j]$ משבצות בכיוון החיובי <u>או השלילי</u> של אחד מצירי הלוח.
- אסור לצאת מגבולות הלוח כלומר צריך להישאר בתחום 0 עד $n-1$ בכל ציר.

ממשו את הפונקציה grid_escape2(B) אשר מקבלת לוח B ומחזירה True אם ניתן להגיע מהמשבצת $(0,0)$ למשבצת $(n-1, n-1)$ ו False אחרת. לדוגמא שני לוחות משחק והפלט עבורם:



False



True

הנחיה: על הפתרון להיות רקורסיבי

רמז: בסעיף זה יש סכנה להיכנס לרקורסיה אינסופית (כמו לולאה אינסופית). מומלץ להשתמש בזיכרון עזר (בדומה לממואיזציה) בגודל הלוח כדי לסמן באילו תאים כבר ביקרנו במהלך הרקורסיה ולהשתמש במידע זה כדי להימנע מרקורסיה אינסופית.

שאלה 5

בחלק זה נדון בסיבוכיות של פעולות חיבור, כפל והעלאה בחזקה של מספרים בכתיב בינארי.

חיבור וכפל של מספרים בינאריים יכול להתבצע בצורה דומה מאוד לזו של מספרים עשרוניים. להלן המחשה של אלגוריתם החיבור והכפל של שני מספרים בינאריים A, B:

```
  1 1 1 1    (carried digits)

      1 1 1 1 (A)
+   1 1 0 1 0 (B)
-------------
= 1 0 1 0 0 1
```

להלן המחשה של אלגוריתם ההכפלה $A \times B$:

```
          1 0 1    (A)
      × 1 0 1 0    (B)
      ---------
          0 0 0    ← Corresponds to a zero in B
  +     1 0 1      ← Corresponds to a one in B
  +   0 0 0
  +  1 0 1
  ---------------
  =  1 1 0 0 1 0
```

בדוגמה הנ״ל יש 19 פעולות. 12 עבור הכפל והשאר עבור החיבור.

בהרצאה נלמד אלגוריתם iterated squaring להעלאה בחזקה של מספרים שלמים חיוביים (ללא מודולו). הקוד מופיע בהמשך.

נסמן ב-n את מספר הביטים ב-a וב-m את מספר הביטים ב-b. עליכם לנתח את סיבוכיות זמן הריצה של הפעולה a^b בשיטה זו כתלות ב-m ו/או n ולתת חסם הדוק ככל האפשר במונחי $O(\cdot)$.

שימו לב שהמספרים המוכפלים גדלים עם התקדמות האלגוריתם, ויש לקחת זאת בחשבון. כמו כן, בניתוח נתייחס רק לפעולות הכפל המופיעות באלגוריתם. אמנם באלגוריתם ישנן פעולות נוספות מלבד כפל, אך לפעולות אלו סיבוכיות זמן מסדר גודל זניח לעומת פעולות הכפל. למשל, ההשוואה $b > 0$ רצה בזמן $O(1)$ (לא נכנסנו לעומק של ייצוג מספרים שלמים שליליים, אבל ראינו ברפרוף את שיטת המשלים ל- 2 ושם ברור שמספיק לקרוא את הביט השמאלי כדי להכריע האם מספר שלם הוא חיובי או שלילי). הפעולה b % 2 דורשת בדיקת הביט הימני של b בלבד ולכן רצה בזמן $O(1)$. פעולת החילוק $b//2$ כוללת העתקה של b ללא הביט הימני ביותר למקום חדש בזיכרון, פעולה שגם כן רצה בזמן ליניארי בגודל של b. על ההסבר להיות תמציתי ומדויק. מומלץ לחשוב על המקרה ה״גרוע״ מבחינת מספר פעולות הכפל ולתאר כיצד הגעתם לחסם עבור פעולות אלו.

```
def power2(a,b):
    """ Computes a**b using iterated squaring
        Assume a,b are integers, b>=0 """
    result = 1
    while b>0:        # b has more digits
        if b%2 == 1: # b is odd
            result = result*a
        a = a*a
        b = b//2      # discard b's LSB
    return result
```

סוף