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

<u>תרגיל בית מספר 4 - להגשה עד 11 במאי (יום ג׳) בשעה 23:55</u>

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

<u>הגשה</u>:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- התשובות בקובץ ה pdf חייבות להיות מוקלדות **ולא** בכתב יד.
- 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 hw4_012345678.pdf ו- hw4_012345678.py.
- מכיוון שניתן להגיש את התרגיל בזוגות, עליכם בנוסף למלא את המשתנה SUBMISSION_IDS שבתחילת קובץ השלד. רק אחת הסטודנטיות בזוג צריכה להגיש את התרגיל במודל.

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

- 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

בהרצאה ראינו את אלגוריתם quicksort אשר משתמש הן ברקורסיה והן באקראיות. האקראיות, כזכור, הייתה בבחירת איבר הציר (pivot) שלפיו נחלק את הרשימה לשלוש רשימות (איברים שקטנים, זהים בגודלם וגדולים מהציר שבחרנו).

דיברנו גם על האפשרות לממש את האלגוריתם באופן דטרמיניסטי, כלומר, ללא שימוש באקראיות. ההצעה שלנו למימוש דטרמיניסטי הייתה פשוטה ביותר, איבר הציר יהיה האיבר הראשון ברשימה. להלן המימוש של פונקציה זו:

```python
def det_quicksort(lst):
  """ sort using deterministic pivot selection """
  if len(lst) <= 1:
    return lst
  else:
    pivot = lst[0]    # select first element from list
    smaller = [elem for elem in lst if elem < pivot]
    equal = [elem for elem in lst if elem == pivot]
    greater = [elem for elem in lst if elem > pivot]
    return det_quicksort(smaller) + equal + det_quicksort(greater) #two recursive calls
```

כזכור, זמן הריצה הטוב ביותר של quicksort (עם אקראיות) הוא $O(n\log n)$ כאשר n הוא אורך הרשימה, ואילו זמן הריצה הגרוע ביותר הוא $O(n^2)$. בסעיפים הקרובים נסתכל על הגרסא הדטרמיניסטית של האלגוריתם ונחשב את הסבירות שזמן הריצה יהיה הגרוע ביותר, כתלות באופן בו מסודרים איברי הרשימה.

סעיף א'

בהנתן n (טבעי, חיובי), נסמן ב-$p(n)$ את מספר הדרכים בהן ניתן לסדר את איברי הרשימה $\text{lst} = [1,2,\dots,n]$.
לדוגמא, עבור $n = 3$ ישנן שש דרכים לסדר את הרשימה:

$$[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]$$

ולכן $p(3) = 6$. מהו $p(n)$ ל-n כללי? הוכיחו את תשובתכם.

סעיף ב'

בהנתן n (טבעי, חיובי), נסמן ב-$w(n)$ את מספר הדרכים בהן ניתן לסדר את איברי הרשימה $\text{lst} = [1,2,\dots,n]$ כך שריצת det_quicksort על הרשימה תרוץ בזמן הארוך ביותר.
לדוגמא, עבור $n = 3$ ניתן לוודא כי הסידורים הבאים:

$$[1,2,3],[1,3,2],[3,1,2],[3,2,1]$$

יניבו את זמן הריצה הגרוע ביותר, ולכן $w(3) = 4$. מהו $w(n)$ ל-n כללי? הוכיחו את תשובתכם.

סעיף ג'

בהנתן שני הסעיפים הקודמים, חשבו את הגבול וציינו מהו:

$$\lim_{n\to\infty}\frac{w(n)}{p(n)}$$

מה ניתן להסיק מהגבול על הסיכוי לכך שזמן הריצה של האלגוריתם יהיה גרוע ביותר ככל ש-n גדל?

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

שאלה 2

הנחיה: הניחו לצורך ניתוח הסיבוכיות בשאלה זאת כי פעולות אריתמטיות לוקחות זמן קבוע.

תזכורת: פעולת ההכפלה של מטריצה A בגודל m על n במטריצה B בגודל n על p יוצרת מטריצה $C = A \cdot B$ בגודל m על p שהאיבר בשורה ה i והעמודה ה j שלה מחושב ע"י

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} \cdot B_{k,j}$$

מהנוסחה, ניתן לראות שחישוב איבר בודד לוקח כ n פעולות ולכן סך זמן החישוב הוא mnp.

בשאלה זאת נרצה לפתור את בעיית סדר ההכפלה המיטבי של רצף מטריצות. נבין את הבעיה ראשית דרך דוגמה.
נניח שנתונות לנו מטריצות $A, C \in \mathbb{R}^{100 \times 10}, B \in \mathbb{R}^{10 \times 100}$ ומטרתנו היא לחשב את המטריצה $D = A \cdot B \cdot C$.
כיוון שכפל מטריצות הינה פעולה אסוציאטיבית, ניתן לעשות זאת בשתי דרכים:

1. $D = (A \cdot B) \cdot C$ - נחשב קודם את $A \cdot B$ מה שייקח כ 10^5 זמן וייתן מטריצה בגודל 100 על 100. לאחר מכן נכפיל את התוצאה ב C מה שייקח עוד 10^5 פעולות.
2. $D = A \cdot (B \cdot C)$ - נחשב קודם את $B \cdot C$ מה שייקח כ 10^4 זמן וייתן מטריצה בגודל 10 על 10. לאחר מכן נכפיל את התוצאה ב C מה שייקח עוד 10^4 פעולות.

ניתן לראות שיש הבדל משמעותי בין השיטות ולכן באופן כללי נרצה לענות על השאלה הבאה: בהינתן רצף של n מטריצות $A_0, \dots, A_{n-1}$ אשר מקיים כי $A_i \in \mathbb{R}^{m_i \times m_{i+1}}$ מה הדרך הטובה ביותר למקם את הסוגריים כדי להקטין את זמן הריצה של המכפלה $B = A_0 \cdot \ldots \cdot A_{n-1}$.
שימו לב: התשובה לשאלה זאת אינה תלויה בערכי המטריצות אלא רק במימדים שלהן. בנוסף, כדי שפעולת הכפל תהיה מוגדרת היטב, מימדי המטריצות צריכים להתאים. כמסקנה, נוכל לתאר את הקלט של השאלה כרשימה $L = [m_0, \dots, m_n]$ באורך $n+1$ אשר הפירוש שלה הוא ש $A_i \in \mathbb{R}^{m_i \times m_{i+1}}$

סעיף א':

ממשו את הפונקציה best_mat_mult_time(L) בקובץ השלד. הפונקציה מקבלת רשימה $L = [m_0, \dots, m_n]$ באורך n+1 שמכילה את ממדי המטריצות המיועדות להכפלה. ומחזירה את זמן החישוב הטוב ביותר של כפל המטריצות. לדוגמא: המטריצות בתחילת השאלה מתוארות ע"י $L = [100{,}10{,}100{,}10]$ ולכן הרצה של הפונקציה על L תחזיר 20000.

הנחייה: על המימוש בסעיף זה להיות רקורסיבי **וללא** ממואיזציה

בקובץ ה pdf ציינו איזו טענה מהטענות הבאות היא נכונה והדוקה ביותר (מבין האפשרויות הנתונות) לגבי סיבוכיות זמן הריצה של הקוד שכתבתם, כפונקציה של n (אורך רשימת המימדים של המטריצות). נמקו והסבירו את תשובתכם.

1. הסיבוכיות היא קבועה
2. הסיבוכיות היא לינארית
3. הסיבוכיות היא פולינומיאלית, כלומר קיים קבוע $c > 0$ כך שסיבוכיות הזמן היא $O(n^c)$
4. הסיבוכיות היא אקספוננציאלית, כלומר קיים קבוע $c > 1$ כך שסיבוכיות הזמן של הפונקציה היא לפחות c^n.

סעיף ב׳:
ממשו את הפונקציה best_mat_mult_time_fast(L) אשר פועלת באופן דומה ל best_mat_mult_time(L) אך משתמשת בממואיזציה כדי להאיץ את זמן הריצה.

הנחייה: על המימוש להיות רקורסיבי ויעיל ככל הניתן (במונחי O).

בקובץ ה pdf ציינו איזו טענה מהטענות הבאות היא נכונה והדוקה ביותר (מבין האפשרויות הנתונות) לגבי סיבוכיות זמן הריצה של הקוד שכתבתם, כפונקציה של n (אורך רשימת המימדים של המטריצות). נמקו והסבירו את תשובתכם.

1. הסיבוכיות היא קבועה
2. הסיבוכיות היא לינארית
3. הסיבוכיות היא פולינומיאלית, כלומר קיים קבוע $c > 0$ כך שסיבוכיות הזמן היא $O(n^c)$
4. הסיבוכיות היא אקספוננציאלית, כלומר קיים קבוע $c > 1$ כך שסיבוכיות הזמן של הפונקציה היא לפחות c^n.

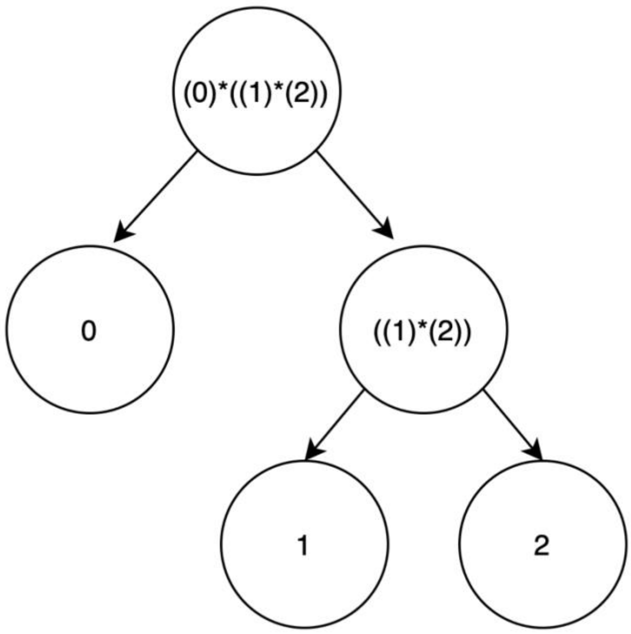
בונוס: מצאו את סיבוכיות הזמן ההדוקה של הקוד במונחי O. נמקו והסבירו את תשובתכם.

סעיף ג׳:
כעת נרצה גם להחזיר את סדר ההכפלה ולא רק את הזמן שייקח לבצע אותה. ממשו את הפונקציה best_mat_mult_order(L) אשר מחזירה את סדר ההכפלה ורצה באותה סיבוכיות זמן כמו בסעיף ב׳.

סדר ההכפלה ישמר בצורה של עץ בינארי אותו נממש בעזרת רשימות וניתן להמיר אותו למחרוזת בעזרת פונקציית העזר mult_order_to_str אשר נתונה לכם בקובץ השלד. באופן פורמאלי, נניח שהפלט של הפונקציה נשמר במשתנה בשם mat_o. אז הוא מקיים את התנאים הבאים באופן רקורסיבי:

- או ש mat_o הוא מספר שלם (אשר מייצג את האינדקס של המטריצה בשרשרת ההכפלות)
- או ש mat_o הוא רשימה באורך 2 (בדיוק) אשר כל איבר בה מקיים את התנאים של mat_o (כלומר הוא או שלם או רשימה באורך 2).

העלים של העץ יהיו למעשה האינדקסים של המטריצות כך שבדוגמא בתחילת התרגיל אשר בה $A_0, A_2 \in \mathbb{R}^{100\times10}, A_1 \in \mathbb{R}^{10\times100}$ נקבל את הפלט $mat_o = [0,[1,2]]$ אשר מתאר את העץ



בשורש העץ כתובה גם המחרוזת שתתקבל מהפעלה של mult_order_to_str על הפלט.

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

שאלה 3

בשאלה זו נרצה לעבוד עם מטריצות. מאחר שאין לנו בפייתון משתנה מטיפוס מטריצה, נשתמש ברשימות מקוננות על מנת לייצג מטריצות. הייצוג שלנו יהיה טבעי ביותר: בהנתן מטריצה M בגודל $n \times k$ (כלומר, מטריצה בת n שורות ו-k עמודות), נבנה רשימה מקוננת lst_m המכילה n תתי-רשימות, כל אחת באורך k.

כעת, את האיבר $M_{i,j}$ נמקם ב-$lst\_m[i][j]$. להלן דוגמא למטריצה בגודל 2×3:

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

והרשימה שתייצג את M היא הרשימה הבאה:

lst_m = [[1, 2, 3], [4, 5, 6]]

נגדיר עתה מטריצה **בינארית** מעניינת במיוחד – מטריצת Hadamard. למספר טבעי n מטריצת Hadamard מסדר n (אותה נסמן מעתה ואילך על ידי had(n)) מוגדרת באופן הרקורסיבי הבא:

- עבור n = 0, had(0) היא מטריצה מגודל 1×1 המכילה את האיבר אפס: $had(0) = (0)$
- עבור n > 0, had(n) היא מטריצה ריבועית מגודל $2^n \times 2^n$ אשר בנויה באופן הרקורסיבי הבא:
  אם had(n-1) היא מטריצת Hadamard מסדר n-1, אזי:

$$had(n) = \begin{pmatrix} had(n-1) & had(n-1) \\ had(n-1) & \overline{had(n-1)} \end{pmatrix}$$

כאשר עבור מטריצה בינארית M נסמן ב-$\overline{M}$ את המטריצה שבה מחליפים ערכי 0 ב-M ב-1 וערכי 1 ב-M ב-0.
כלומר, had(n) מורכבת מ-4 עותקים של had(n-1) (זוהי מטריצת בלוקים עם שני בלוקים עליונים של had(n-1) ושני בלוקים תחתונים של had(n-1)), כאשר בבלוק הימני-תחתון ערכי המטריצה מתהפכים (כלומר, ערכי 0 הופכים ל-1 וערכי 1 ל-0).

למטריצת Hadamard התכונה המרתקת הבאה: כל שתי שורות שונות במטריצה נבדלות בדיוק בחצי מהקואורדינטות שלהן (בדקו זאת!). תכונה זו שימושית במיוחד בתחום של תיקון שגיאות, אותו נפגוש בהמשך הקורס.

להלן שלוש המטריצות had(0), had(1), had(2) מיוצגות כרשימות מקוננות על פי הייצוג שקבענו לעיל:

had0 = [[0]]

had1 = [[0, 0], [0, 1]]

had2 = [[0, 0, 0, 0], [0, 1, 0, 1], [0, 0, 1, 1], [0, 1, 1, 0]]

שימו לב: בשאלה זו יש להתחשב בסיבוכיות זמן הריצה של פעולות אריתמטיות (כלומר, אין להניח כי פעולות אלו רצות זמן קבוע). <u>ניתן להניח כי החישוב של $pow(2,\ n)$ לוקח זמן $O(n)$ (ולא פחות מכך).</u>

<u>סעיף א'</u>

בקובץ ה*pdf* הוכיחו באינדוקציה כי לכל n מתקיימת התכונה הבאה: במטריצה *had(n)*, כל שורה מלבד השורה העליונה מכילה מספר זהה של אפסים ואחדות.

<u>סעיף ב'</u>

בקובץ השלד השלימו את מימוש הפונקציה $had\_local(n, i, j)$ אשר מקבלת כקלט שלושה מספרים שלמים $n \geq 0$ ו-$0 \leq i, j < 2^n$ ומחזירה את $had(n)_{i,j}$ (כלומר, את הכניסה בשורה ה-i והעמודה ה-j במטריצת $Hadamard$ מסדר n.

דוגמאות הרצה:

```
>>> had_local(2, 1, 2)
0
>>> had_local(2, 2, 2)
1
>>> had_local(0, 0, 0)
0
>>> had_local(1, 1, 1)
1
```

<u>הנחיות</u>: - על המימוש להיות רקורסיבי
- יש לממש את הפונקציה בצורה יעילה ככל הניתן
- בפרט, אין לייצר את כל המטריצה $had(n)$

<u>סעיף ג'</u>

בקובץ הpdf ציינו מהו זמן הריצה של הפונקציה שמימשתם בסעיף הקודם במונחים אסימפטוטיים, כתלות ב-n. נמקו את תשובתכם.

<u>סעיף ד'</u>

בקובץ השלד השלימו את מימוש פונקצית הלמבדא $had\_complete$. הפונקציה מקבלת כקלט שלם אי-שלילי $n \geq 0$ ומחזירה את מטריצת $Hadamard$ מסדר n.
עליכם להשתמש בפונקציה had_local.

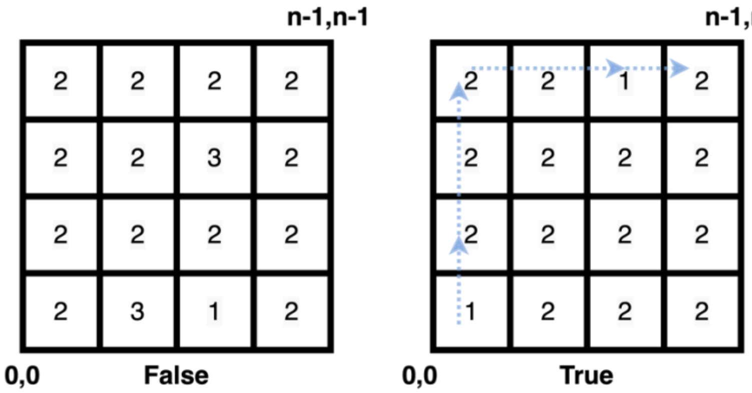
<u>שאלה 4</u>

בשאלה זאת נתון לוח דו-מימדי B בגודל n על n אשר מכיל מספרים שלמים אי-שליליים (כולל 0). בדומה לשאלה 3, 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 חוק ההתקדמות בלוח הוא:
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 בכיוון החיובי של אחד מצירי הלוח.

ממשו את הפונקציה grid_escape1(B) אשר מקבלת לוח B ומחזירה True אם ניתן להגיע מהמשבצת $\mathbf{0, 0}$ למשבצת $\boldsymbol{n-1, n-1}$ ו False אחרת. לדוגמא שני לוחות משחק והפלט עבורם:



שימו לב למיקום של נקודת ההתחלה והסוף בציור. בחירה זאת הינה שרירותית ולא משפיעה על הפתרון אך קריטית לצורך הבנת הדוגמא!

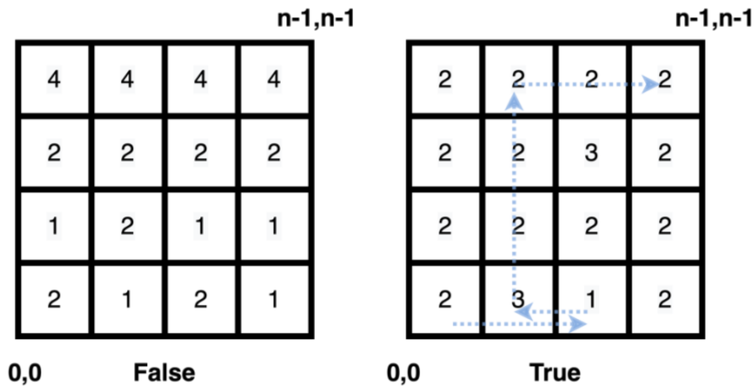
הנחיה: על הפתרון להיות רקורסיבי

<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 או השלילי 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 $\mathbf{0, 0}$ למשבצת $\boldsymbol{n-1, n-1}$ ו False אחרת. לדוגמא שני לוחות משחק והפלט עבורם:



הנחיה: על הפתרון להיות רקורסיבי

רמז: בסעיף זה יש סכנה להיכנס לרקורסיה אינסופית (כמו לולאה אינסופית). מומלץ להשתמש בזכרון עזר (בדומה לממואיזציה) בגודל הלוח כדי לסמן באילו תאים כבר ביקרנו במהלך הרקורסיה ולכן אין צורך לבדוק אותם שוב.

<u>שאלה 5</u>

בשאלה זאת שני סעיפים בלתי תלויים.

<u>סעיף א'</u>

בסעיף זה נפתור בעיה דומה ל subset sum אשר נקראת partition. השלימו את הפונקציה partition אשר מקבלת כקלט רשימה S באורך n של מספרים שלמים ומחזירה תת רשימה P שמקיימת

$$\sum_{s \in P} s = \sum_{s \in S \setminus P} s$$

ו $None$ אם לא קיימת תת-רשימה כזאת. כלומר, תת הרשימה P צריכה לקיים שסכום האיברים בה שווה לסכום האיברים ב S שלא נמצאים בה. שימו לב כי האיברים ב S אינם יחידים ולכן יכולים להופיע מספר פעמים.

הנחיה: על הפתרון להיות רקורסיבי.

לדוגמא, אם $S = [3,1,1,2,2,1]$ אז פלט של $[3,2]$ הוא תשובה תקינה כי סכום האיברים שווה לזה של הרשימה המשלימה $[1,1,1,2]$. לעומת זאת, אם $S = [1,1,1]$ אז לא ניתן לחלקה לשתי תתי-רשימות שסכומן שווה ולכן יוחזר $None$.

<u>סעיף ב'</u>

השלימו את הפונקציה n_to_k בקובץ השלד אשר מקבלת שני מספרים שלמים חיוביים n ו k ומחזירה את מספר הדרכים לחלק קבוצה של n איברים **לבדיוק** k קבוצות לא ריקות **ללא חזרות וללא חשיבות לסדר**. לדוגמא, עבור $n = 4, k = 2$ נראה שהתשובה היא 7. כדי לראות זאת ניתן לקחת כל קבוצה בגדול 4 כגון $S = \{a, b, c, d\}$ ולראות שמספר הדרכים לחלק אותה לבדיוק שתי תתי קבוצות לא ריקות הן:

$$\{\{a\},\{b,c,d\}\},\{\{b\},\{a,c,d\}\},\{\{c\},\{a,b,d\}\},\{\{d\},\{a,b,c\}\},\{\{a,b\},\{c,d\}\},\{\{a,c\},\{b,d\}\},\{\{a,d\},\{b,c\}\}$$

הנחיות:

1. על המימוש להיות רקורסיבי
2. השתמשו בממואיזציה כדי לקבל פתרון עם סיבוכיות זמן של $O(nk)$
3. ניתן להניח ש $k \leq n$

הסבירו בקובץ ה pdf מדוע הקוד שלכם מקיים את דרישת סיבוכיות הזמן.

שאלה 6

בשאלה זו נממש אלגוריתמים **לחישוב מרחק עריכה** בין מחרוזות. מרחק עריכה בין שתי מילים מוגדר כמספר השינויים המינימלי הדרוש כדי לעבור ממחרוזת אחת לשנייה, כאשר השינויים המותרים הם החלפת אות, הוספת אות והשמטת אות.

דוגמאות:

- מרחק העריכה בין "computer" ל "commuter" הוא 1 (ניתן להחליף את p ב m).
- גם מרחק העריכה בין "sport" ל "sort" הוא 1 (ניתן להשמיט את האות p)
- מרחק העריכה בין "kitten" ל- "sitting" הוא 3, באמצעות סדרת השינויים הבאה:
  - נחליף את k ב- s (ונגיע ל "sitten")
  - נחליף את e ב- i (ונגיע ל "sittin")
  - נוסיף g בסוף המחרוזת (ונגיע ל- "sitting")

שימו לב שייתכנו רצפים שונים של פעולות שמובילות ממחרוזת אחת לשניה, ואנו מתעניינים באורך המינימלי של רצף כזה. שימו לב גם שמרחק העריכה הוא סימטרי: אין זה משנה, מבחינת אורך הרצף, אם מתחילים מהמילה הראשונ

ה ומגיעים לשנייה או להפך.

סעיף א'

ממשו בקובץ השלד את הפונקציה distance(s1,s2) שמחשבת את מרחק העריכה בין שתי מחרוזות.

הנחיות:

- על הפונקציה להיות רקורסיבית
- אין להשתמש בלולאות for או while
- בסעיף זה אין להשתמש בממואיזציה.

דוגמאות הרצה:

```
>>> distance('computer','commuter')
1
>>> distance('sport','sort')
1
>>> distance('','ab')
2
>>> distance('kitten','sitting')


3
```

<u>סעיף ב'</u>

הניחו לצורך ניתוח הסיבוכיות בסעיף זה כי פעולות אריתמטיות לוקחות זמן קבוע. כמו כן, נניח שאורך כל מחרוזת הוא $\geq n$.

בקובץ ה pdf ציינו איזו טענה מהטענות הבאות היא נכונה והדוקה ביותר (מבין האפשרויות הנתונות) לגבי סיבוכיות זמן הריצה של הקוד שכתבתם, כפונקציה של n. נמקו והסבירו את תשובתכם.

5. הסיבוכיות היא קבועה
6. הסיבוכיות היא לינארית
7. הסיבוכיות היא ריבועית
8. הסיבוכיות היא אקספוננציאלית, כלומר קיים קבוע $c > 1$ כך שסיבוכיות הזמן של הפונקציה היא $O(c^n)$.

<u>סעיף ג'</u>

השלימו בקובץ השלד את הפונקציה distance_fast(s1,s2) שהיא גירסה עם ממאוזציה לפונקציה מסעיף א'.

הנחיות:

- הפונקציה distance_fast הינה פונקציית מעטפת. עליה לקרוא לפונקציה רקורסיבית בשם distance_mem.
- אין להשתמש בלולאות for או while

<u>סעיף ד'</u>

הניחו לצורך ניתוח הסיבוכיות בסעיף זה כי פעולות אריתמטיות לוקחות זמן קבוע. כמו כן, נניח שאורך כל מחרוזת הוא $\geq n$.

בקובץ ה pdf ציינו מה תהיה סיבוכיות הקוד לאחר הוספת ממואיזציה, כפונקציה של n? נמקו והסבירו.

<u>סוף</u>