תרגיל בית 5

נושאים: אלגוריתמים, רקורסיה

1. (50%) ראינו בהרצאה ששיטות ההצפנה המודרניות משתמשות במספרים ראשוניים גדולים מאוד. בתרגיל זה נלמד איך לבדוק באמצעות אלגוריתם Miller-Tabin אם מספר גדול מאוד הוא ראשוני. את התוכנית עליכם לכתוב בקובץ check_primes.py.

בתרגיל זה תתרגלו מימוש אלגוריתם הנתון ב pseudocode. **שימו לב! אף שהתרגיל נראה ארוך וקשה, זהו למעשה תרגיל קל. כל שעליכם לעשות הוא לפעול לפי ההוראות.**

א. עבור מספר שלם חיובי נתון n נגדיר את **הדרגה הזוגית** (even degree) של n בתור המעריך של החזקה הגבוהה ביותר של 2 שמחלקת אותו. נסמן את זה ב even(n). למשל, even(12)=2 (כי 2^2 מחלק את 12, אבל 2^3 לא מחלק את 12). even(64)=6, even(2)=1, even(k)=0 לכל מספר אי-זוגי k.

עבור מספר שלם חיובי נתון n נגדיר את **החלק האי-זוגי** (odd part) של n בתור המספר האי-זוגי הגבוה ביותר שמחלק אותו. נסמן את זה ב odd(n). למשל, odd(12)=3 , odd(64)=1, odd(k)=k לכל מספר אי-זוגי k, ו- odd(k)=1 לכל מספר k שהוא חזקה של 2.

עליכם לכתוב פונקציה get_even_odd_parts שמקבלת מספר שלם חיובי n ומחזירה זוג מספרים: even(n) ו odd(n).

רמז: שימו לב שאם s=even(n) ו t=odd(n) אז $n=2^s t$.

ב. בסעיף זה נממש פונקציה is_probably_prime() לקביעה האם מספר נתון הוא ראשוני:

<u>ארגומנטים</u>:

n = מספר שלם חיובי (שאותו רוצים לבדוק)

num_iterations = מספר שלם חיובי (בין 5 ל 10)

<u>ערך מוחזר</u>:

Ture – אם יש סיכוי גבוה ש n ראשוני

False – אם בטוח ש n לא ראשוני

<u>אופן פעולת הפונקציה</u>:

i. הפונקציה תחשב את s=even(n-1) ואת t=odd(n-1) באמצעות הפונקציה מסעיף א'.

ii. הפונקציה תריץ num_iterations פעמים את הקריאה is_suspected_prime(n,t,s) (ר' סעיף ג').

iii. אם **<u>בכל</u>** הפעמים הוחזר True, אז is_probably_prime תחזיר True. אם באחת הפעמים הוחזר False – הפונקציה תחזיר False.

ג. בסעיף זה עליכם לממש את האלגוריתם הבא בפונקציה is_suspected_prime:

<u>ארגומנטים</u>:

n = מספר שלם חיובי

t = מספר שלם חיובי

s = מספר שלם חיובי

<u>ערך מוחזר</u>:

True – אם יתכן ש n ראשוני

False – אם בטוח ש n לא ראשוני.

<u>אופן פעולת הפונקציה</u>: (עליכם "לתרגם" את האלגוריתם הבא לשפת פייתון):

```
choose a random positive integer a between 2 and n-1
let d= a^t mod n
if d==1 or d==n-1 then return True
for i=1 to s – 1
    d= d^2 mod n
    if d == n – 1
        then return True
return False
```

את החזקה a^t mod n יש לחשב בעזרת הפונקציה modular_power שלמדנו בהרצאה.

ד. כיתבו תוכנית check_primes.py שקוראת מספרים מקובץ input_ex1.txt, כל מספר בשורה נפרדת. הפונקציה תבדוק לכל מספר אם הוא ראשוני באמצעות הפונקציה ()is_probably_prime מסעיף ב' עם ערך num_iterations=10 ותדפיס לקובץ output_ex1.txt את המספרים וליד כל מספר אם הוא ראשוני או לא.
למשל, אם הקלט הוא

```
191
36
26
77
31
97
61
796
15
353
```

אז הפלט יהיה

```
191 is prime
36 is not prime
26 is not prime
77 is not prime
31 is prime
97 is prime
61 is prime
796 is not prime
15 is not prime
353 is prime
```

מצורפים קובץ קלט וקובץ פלט לדוגמה.

2. (25%) כיתבו פונקציה רקורסיבית num_sums שמקבלת מספר שלם חיובי n ורשימה parts של מספרים שלמים חיוביים **שונים**. הפונרציה תחזיר את מספר הדרכים לכתוב את n כסכום של מספרים מתוך הרשימה parts. לסדר המספרים בסכום יש חשיבות.
דוגמה: הקריאה num_sums(6, [1,3,5]) תחזיר 8, כי יש 8 אפשרויות לכתוב את 6 כסכום של המספרים 1, 3 ו-5

```
1 + 1 + 1 + 1 + 1 + 1
1 + 1 + 1 + 3
1 + 1 + 3 + 1
1 + 3 + 1 + 1
1 + 5
3 + 1 + 1 + 1
3 + 3
5 + 1
```

אפשר להניח שקלט הפונקציה תקין. כלומר, כל המספרים הם שלמים וחיוביים וכל המספרים ברשימה שונים.

עכשיו כיתבו תוכנית num_sums.py שקוראת מקובץ input_ex2.txt שורות של מספרים. בכל שורה המספר הראשון הוא המספר n ושאר המספרים הם הרשימה parts. עבור כל שורה של קלט הפונקציה תדפיס לקובץ output_ex2.txt את מספר הדרכים לכתוב את n כסכום של מספרים מתוך הרשימה parts. למשל, אם הקלט הוא

```
6 1 3 5
5 1 2 3
4 1 2
```

אז הפלט יהיה:

```
8
13
5
```

אין להניח שקלט התוכנית תקין. יש לבדוק שבכל שורה יש לפחות שני מספרים שלמים חיוביים ושכל המספרים שמהם צריכים להרכיב את הסכומים שונים. אם שורה לא תקינה יש לכתוב בפלט Error. שימו לב לדוגמאות הפלט המצורפות.

3. (25%) כיתבו פונקציה רקורסיבית print_sums שמקבלת מספר שלם חיובי n, רשימה parts של מספרים שלמים חיוביים **שונים** וקישור לקובץ (מה שמוחזר ע"י open).
הפונקציה תדפיס לקובץ את כל הדרכים לכתוב את n כסכום של מספרים מתוך הרשימה parts. לסדר המספרים בסכום יש חשיבות.

עכשיו כיתבו תוכנית print_sums.py שקוראת מקובץ input_ex3.txt שורות של מספרים. בכל שורה המספר הראשון הוא המספר n ושאר המספרים הם הרשימה parts. עבור כל שורה של קלט הפונקציה תדפיס לקובץ output_ex3.txt את כל הדרכים לכתוב את n כסכום של מספרים מתוך הרשימה parts.
למשל, אם הקלט הוא

```
6 1 3 5
4 1 2
```

אז הפלט יהיה:

```
6 as sum of [1, 3, 5]:
6 = 1 + 1 + 1 + 1 + 1 + 1
6 = 1 + 1 + 1 + 3
6 = 1 + 1 + 3 + 1
6 = 1 + 3 + 1 + 1
```

```
6 = 1 + 5
6 = 3 + 1 + 1 + 1
6 = 3 + 3
6 = 5 + 1

4 as sum of [1, 2]:
4 = 1 + 1 + 1 + 1
4 = 1 + 1 + 2
4 = 1 + 2 + 1
4 = 2 + 1 + 1
4 = 2 + 2
```

הנחיות נוספות:

1) אין להניח שקלט התוכנית תקין. יש לבדוק שבכל שורה יש לפחות שני מספרים שלמים חיוביים ושכל המספרים שמהם צריכים להרכיב את הסכומים שונים. אם שורה לא תקינה יש לכתוב בפלט Error. שימו לב לדוגמאות הפלט המצורפות.
2) את רוב הקוד של התוכנית ניתן לקחת מהתרגיל הקודם

הנחיות הגשה:

1- יש להגיש תוכניות שרצות ללא שגיאות. תוכנית שתוגש עם שגיאות תקבל לכל היותר חצי מהנקודות.
2- יש לכתוב הערות לתוכנית: docstring בתחילת כל פונקציה, הסבר קצר בתחילת התוכנית, הסבר בתחילת לולאות.
3- אין להשתמש במודולים מלבד מודולים סטנדרטיים כמו math, random, sys,
4- יש לפתור כל שאלה בקובץ נפרד עם סיומת py.
5- יש להגיש את כל הקבצים בקובץ אחד מכווץ עם סיומת zip. שם קובץ ה zip צריך להיות מספר הת"ז שלכם ומספר עבודת הבית. למשל, 222222222_hw5.zip
6- כל קובץ יתחיל בהערה ובה המידע הבא:
   א. שם הסטודנט
   ב. מס' תעודת זהות
   ג. מספר דף התרגילים
   ד. שם התוכנית
7- למשל, עבור תרגיל 1 בדף 5:

```
"""
Student: Madonna Louise Ciccone
ID: 121212121
Assignment no. 5
Program: primes.py
"""
```

שימו לב: יש להקפיד על הנחיות ההגשה האלה. הגשה שלא בדיוק בפורמט הזה לא תקבל את מלוא הנקודות ואף עלולה להיפסל.