<u>תרגיל 4 – רקורסיה</u>

תאריך פרסום: 19/11/2020

תאריך הגשה: 06/12/2020

מתרגל אחראי: אוריאל פרידמן

משקל התרגיל: 3 נקודות

<u>הנחיות כלליות:</u>

- העבודה תבוצע ביחידים.
- קראו את ההוראות לגבי הגשת תרגילי הבית באתר הקורס.
- מומלץ לקרוא את כל העבודה לפני תחילת הפתרון.
- כתבו תיעוד (הערות) שמסביר את הקוד שלכם. אין לכתוב הערות בעברית.
- עליכם להוריד את הקובץ "hw4.py" מתיקיית "תרגיל בית 4" מהמודל, ולהכניס את הקוד שלכם בשורות המתאימות. תבצעו את השינויים שלכם בהתאם להערות בקובץ.
- אין להשתמש בחבילות או במודולים, אלא אם נאמר במפורש.
- **ניתן להניח שהקלט תקין, אלא אם נכתב אחרת בשאלה.**
- העבודה תיבדק באופן אוטומטי ולכן על הפלטים להיות בדיוק כפי שמוגדר בתרגיל .
- העתקת קוד (משנים קודמות, מחברים או מהאינטרנט) עלולה להוביל לכישלון בקורס. **אל תעתיקו!**
- שאלות בנוגע לעבודה ישאלו ב-"פורום שאלות לתרגיל בית 4" במודל או בשעות הקבלה של המתרגל האחראי בלבד.
- את העבודה יש להגיש דרך מערכת ההגשה בכתובת:
  https://subsys.ise.bgu.ac.il/submission/login.aspx
- טרם ההגשה אנא וודאו:
  - כל אחד מהקבצים מתקמפל ורץ כנדרש.
  - המשתנים שכתבתם עם שמות משמעותיים (ללא שמות כמו a).
  - בתחילת כל שאלה כתבתם הערות לקוד באנגלית.
  - אין הדפסות מיותרות (למשל הרצה של טסטים).
- אתם רשאים להוסיף פונקציות עזר לבחירתכם(בנוסף לפונקציות העזר שאתם נדרשים לממש בעבודה), בתנאי שהן אינן מפרות את הוראות הסעיף בו בחרתם לעשות בהן שימוש. על פונקציות העזר להיכתב בתוך הקובץ hw4.py לצד שאר הקוד שלכם.
- בתרגיל זה **<u>אין להשתמש בלולאות</u>** **<u>או בפונקציות מובנות של פייתון</u>**, אלא אם צוין אחרת.
- מטרות העבודה: תרגול רקורסיה.

בהצלחה!

חלק א':

סדרת *פיבונאצ'י אותיות* מוגדרת כסדרת מחרוזות, בה כל מחרוזת S_n היא שרשור של שתי המחרוזות שקדמו לה בסדרה S_{n-1}, S_{n-2}, כאשר $S_0 = 'a'$, ו-$S_1 = 'bc'$.

דוגמאות: $S_2 = 'abc'$, $S_3 = 'bcabc'$.

ממשו פונקציה fibonacci_chars(n, k) אשר מחזירה את התו במקום ה-k של האיבר ה-n (S_n) בסדרת פיבונאצ'י אותיות.

הניחו כי $k < len(S_n)$, $k \geq 0$.

דוגמה 1:

בהינתן הקלט n=5 וk=8, סדרת הפיבונאצ'י אותיות שנוצרת היא 'bcabcabc**b**cabc' והתו שיחזור הוא 'b'.

```
print(fibonacci_chars(n=5, k=8))
b
```

דוגמה 2:

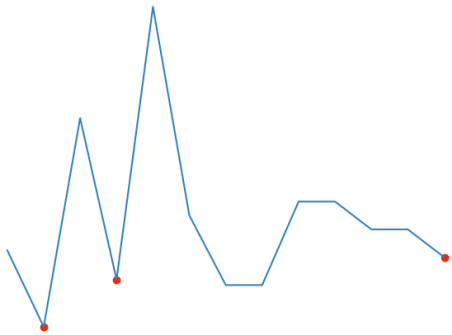
```
print(fibonacci_chars(n=1, k=0))
b
```

חלק ב':

ארגון מהנדסים ללא גבולות מעוניין לתכנן מערכת שאיבת מים לכפרים הרריים ומרוחקים. באזור כל כפר יש רכס המחולק ל – n *תאי שטח*, כאשר הגובה הממוצע של תא שטח נתון כמספר. נגדיר *אגן ניקוז* כתא שטח שנמוך ממש משני תאי השטח הצמודים אליו ובכך מנקז מים הזורמים אליו, תא שטח בקצה הרכס (תא שטח הראשון או האחרון) שנמוך מתא השטח הסמוך גם יכלל בהגדרה של אגן ניקוז.

לדוגמה, הכפר *בזינגה* נמצא בקרבת הרכס המתואר ע"י הרשימה:

[2.5, -3, 12, 0.4, 20, 5, 0, 0, 6, 6, 4, 4, 2]



ברכס זה, האינדקסים של אגני הניקוז (מסומנים באדום) הם: [1,3,12].

ממשו פונקציה בשם drainage_basins(elavation_ridge) המקבלת כקלט רכס כרשימת גבהים ומחזירה רשימה המכילה את כל האינדקסים של אגני הניקוז.

הניחו כי הרשימה elavation_ridge אינה ריקה.

דוגמה 1:

```
elevation_ridge = [2,3,1,4,2,5,6,1,6,5,4,4,2] # himalaya
print(drainage_basins(elevation_ridge))
```

```
[0, 2, 4, 7, 12]
```

דוגמה 2:

```
elevation_histogram = [0, 0, 0] # salt plains
print(drainage_basins(elevation_histogram))
```

```
[]
```

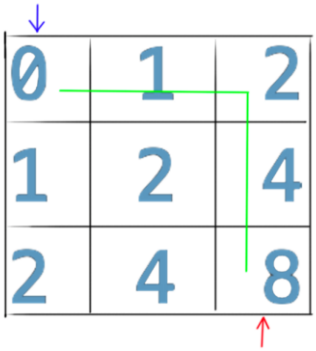
חלק ג':

לקראת העונה החדשה של "נינג'ה ישראל" יצרו מפיקי התוכנית מתקן חדש. המתקן החדש הוא מעין קיר טיפוס, הבנוי בצורת סבכה בגודל n * m, כאשר כל תא ij בסבכה מכיל פריט במשקל w_{ij}, ויכול לתמוך במשקל כולל של פריטים קטן **ממש** מ-$2w_{ij}$ (התעלמו ממשקל המשתתף). בהגעה לתא, המשתתף מחויב לאסוף את המשקולת בתא, ומשקלה מתווסף למשקל שצבר עד כה. אם סך משקל הפריטים המצטבר חורג מסף המשקל הנתמך בתא, המשתתף נופל למים ונפסל. *סוף המסלול* מוגדר כתא במיקום (n-1,m-1), כלומר התא האחרון בשורה האחרונה. המתקן בנוי במבנה שמאפשר מעבר אך ורק בין תאים סמוכים בכיוונים למעלה\למטה\ימינה\שמאלה, ללא אלכסונים. *מסלול חוקי* מוגדר כסדרת מעברים בין תאים שבהם סך משקל הפריטים המצטבר אינו חורג מסף המשקל הנתמך בכל אחד התאים בדרך.

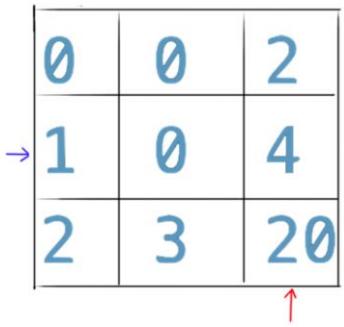
עליכם לממש תוכנה המקבלת כקלט את מבנה הסבכה, המשקולות בכל תא, ותא התחלה ומחזירה את מס' כל המסלולים החוקיים מההתחלה לסוף המסלול.

בדוגמאות הבאות חץ כחול מסמן את תא ההתחלה.

שישה מסלולים חוקיים (בירוק מסומן אחד מהמסלולים החוקיים)



לא קיים מסלול חוקי



A. ממשו פונקציה is_legit_track(grid, i1, j1, i2, j2, current_weight) אשר מחזירה ערך בוליאני המייצג את חוקיות המעבר מתא (i1,j1) לתא (i2,j2) בהינתן סך המשקל המצטבר current_weight והסבכה הנתונה grid, הממומשת כרשימה דו-מימדית כאשר התא ה-(i,j) מכיל את משקל הפריט בתא, w_{ij}. הניחו כי ערך הקורדינטות בטווח התקין, וכי סך המשקל המצטבר הנתון ≥ 0.

דוגמת ריצה:

```
grid = [
    [0, 0, 2 ],
    [1, 0, 4 ],
    [2, 3, 20]
]
print(is_legit_track(grid, 0, 0, 1, 0, 0))
print(is_legit_track(grid, 0, 0, 1, 1, 0))
print(is_legit_track(grid, 2, 0, 2, 1, 3))
```

```
True
False
False
```

B. ממשו פונקציה get_number_legit_tracks(grid, i1, j1) אשר מחזירה את מספר המסלולים החוקיים מתא ההתחלה (i1,j1) לתא הסיום, כאשר המשקל המשתתף מתחיל את המסלול ללא פריטים. הניחו כי ערך הקורדינטות בטווח התקין.

דוגמאות ריצה:

```
grid = [
    [0, 0, 2 ],
    [1, 0, 4 ],
    [2, 3, 20]
]
print(get_number_legit_tracks(grid, 0, 0))
```

```
0
```

```
grid = [
    [0, 1, 2],
    [1, 2, 4],
    [2, 4, 8]
]
print(get_number_legit_tracks(grid, 0, 0))
```

```
6
```

חלק ד':

עדי וישי עוברים בקרוב לדירת גן בבאר שבע, עליכם לעזור להם לבחור פרחים לשתילה בחצר בהתאם לתקציב נתון. לכל פרח נתון ערך אסתטי (סובייקטיבי) ומחיר. שני הערכים אינם שליליים.

A. ממשו פונקציה optimize_flowers_selection(flowers, budget), המקבלת את flowers - רשימת הפרחים הזמינים במשתלה ו-budget – התקציב. רשימת הפרחים היא רשימה של tuples עם שלושה ערכים (לפי הסדר): מחרוזת המייצגת את שם הפרח, הערך האסטטי, והעלות של הפרח. התקציב הוא ערך שלם לא שלילי. הפונקציה מחזירה tuple עם הערך האסתטי המצטבר המקסימלי בהינתן התקציב ואת שארית התקציב (ראו דוגמת הרצה). **שימו לב**, לא ניתן לבחור את אותו הפרח יותר מפעם אחת (קיימת יחידה אחת מכל פרח במשתלה).

דוגמאות ריצה:

```
flowers = [
    # flower_name , aesthetic_value, cost
    ("Lilac", 1, 3),
    ("Hibiscus", 2, 4)
]
print(optimize_flowers_selection(flowers, 4))
```

```
(2, 0)
```

```
flowers = [
    # flower_name , aesthetic_value, cost
    ("Lilac", 4, 3),
    ("Hibiscus", 2, 5),
    ("Tulip", 1, 1),
    ("Sunflower", 5, 2),
    ("Rose", 3, 4)
]
answer = optimize_flowers_selection(flowers, 4)
print("accumulated aesthetic value:{0}"\
"\nRemaining budget:{1}".format(answer[0], answer[1]))
```

```
accumulated aesthetic value:6
Remaining budget:1
```

B. ע"מ לחסוך בחישובים כפולים נתבקשתם לממש את הפונקציה מחדש בעזרת טכניקת Memoization. ממשו את הפונקציה optimize_flowers_selection_mem(flowers, budget) כך שלא תחזור על חישוב שכבר התבצע בעבר.

דוגמת ריצה:

```
flowers = [
    # flower_name , aesthetic_value, cost
    ("Lilac", 4, 3),
    ("Hibiscus", 2, 5),
    ("Tulip", 1, 1),
]
print(get_plants_to_buy_faster(flowers, 4))
```

```
(5, 0)
```

הדרכה: ממשו פונקציית עזר רקורסיבית שתעדכן מבנה נתונים אשר יכיל תוצאות חישובים קודמים.

נספח:

קטעי הקוד מהדוגמאות כטקסט -

חלק א':

```
print(fibonacci_chars(n=4, k=2))
```

```
print(fibonacci_chars(n=1, k=0))
```

חלק ב':

```
elevation_ridge = [2,3,1,4,2,5,6,1,6,5,4,4,2]
print(drainage_basins (elevation_ridge))
```

```
elevation_histogram = [0, 0, 0]
print(drainage_basins (elevation_ridge))
```

חלק ג' A:

```
grid = [
    [0, 0, 2 ],
    [1, 0, 4 ],
    [2, 3, 20]
]
print(is_legit_track (grid, 0, 0, 1, 0, 0))
print(is_legit_track (grid, 0, 0, 1, 1, 0))
print(is_legit_track (grid, 2, 0, 2, 1, 3))
```

חלק ג' B:

```
grid = [
    [0, 1, 2],
    [1, 2, 4],
    [2, 4, 8]
]
print(get_number_legit_tracks (grid, 0, 0))
```

```
grid = [
    [0, 0, 2 ],
    [1, 0, 4 ],
    [2, 3, 20]
]
print(get_number_legit_tracks (grid, 0, 0))
```

חלק ד' A:

```
flowers = [
    # flower_name , aesthetic_value, cost
    ("Lilac", 1, 3),
    ("Hibiscus", 2, 4)
]
print(optimize_flowers_selection (flowers, 4))
```

```
flowers = [
    # flower_name , aesthetic_value, cost
    ("Lilac", 4, 3),
    ("Hibiscus", 2, 5),
    ("Tulip", 1, 1),
    ("Sunflower", 5, 2),
    ("Rose", 3, 4)
]
answer = optimize_flowers_selection(flowers, 4)
print("accumulated aesthetic value:{0}"\
"\nRemaining budget:{1}".format(answer[0], answer[1]))
```

חלק ד' B:

```
flowers = [
    # flower_name , aesthetic_value, cost
    ("Lilac", 4, 3),
    ("Hibiscus", 2, 5),
    ("Tulip", 1, 1),
]
print(get_plants_to_buy_faster(flowers, 4))
```