

מבוא למדעי המחשב 67101 – סמסטר ב' 2022

תרגיל 8 – Backtracking

להגשה בתאריך **18/5/2022** בשעה 22:00

הקדמה והערות חשובות

בתרגיל זה נתרגל שימוש ב **Backtracking**. אנא קראו את הדברים הבאים לפני תחילת עבודה:

- התרגיל הוא **לעבודה אישית בלבד** (הגשה ביחיד).
- מצורף קובץ **puzzle_solver.py** בו עליכם לממש את התרגיל. אין לשנות את חתימות הפונקציות בקובץ. הגישו קובץ ex8.zip שמכיל את **puzzle_solver.py** בלבד.
- לפני מימוש פונקציה, קראו את כל הסעיף, וודאו הבנה של הדוגמאות המצורפות.
- ניתן להוסיף פונקציות נוספות וניתן להשתמש בשאלות מאוחרות יותר בפונקציות שמומשו קודם.
- סגנון: הקפידו על תיעוד נאות ובחרו שמות משתנים משמעותיים. הקפידו להשתמש בקבועים (שמות משתנים באותיות גדולות), על פי ההסברים שנלמדו, רק אם יש בכך צורך.
- בכל השאלות ניתן להניח שהקלט חוקי (ראו פירוט בהמשך) ואין צורך לבצע בדיקות תקינות עבורו.
- אין לייבא (לעשות import) לספריות אחרות מלבד typing.
- חלק מהפונקציות בתרגיל זה רצות בזמן אקספוננציאלי ואינן מאוד יעילות. אם זאת, יש להשתדל ולא לעשות עבודה מיותרת. למשל, בעת מימוש backtracking יש לפסול מוקדם ככל הניתן כיוונים שלא יובילו לפתרון תקין.
- **קראו את כל המסמך לפני תחילת העבודה!**

מטרת התרגיל

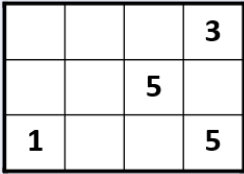
בתרגיל זה נציג חידת הגיון (משחק) ונשתמש בגישוש נסוג (Backtracking) על מנת לפתור אותה.

הגדרת המשחק

המשחק מתבצע על גבי טבלה מרובעת $n \times m$ (n שורות, m עמודות) אשר בחלק מתאיה ישנם מספרים. כל תא בטבלה יכול להצבע או בשחור או בלבן. המטרה היא לקבוע איזה תא יהיה שחור ואיזה לבן. כדי לקבוע איזה תא יהיה שחור ואיזה לבן, משתמשים באילוץ הבא: כל מספר בלוח מייצג את מספר התאים ה"נראים" מהתא שלו כולל התא שלו. תא יכול "להראות" מתא אחר, אם ורק אם שניהם לבנים ונמצאים באותה השורה או העמודה כאשר אין ביניהם תא שחור.

שימו לב: עבור לוח משחק נתון, יתכנו מספר פתרונות שונים, פתרון יחיד, או העדר פתרון.

דוגמאות:

פתרונות	לוח משחק
	
אין פתרון	
 	

שפה משותפת

בחלק זה נגדיר את האופן בו אנו מייצגים את המשחק בפייתון. נשים לב שלוח משחק מוגדר ע"י גודלו (מספר השורות n ומספר העמודות m) וע"י אוסף מספרים עם מיקומם על הלוח (קבוצת אילוצים).

קבוצת אילוצים:

עבור לוח משחק נתון, נגדיר את **קבוצת האילוצים** שלו להיות קבוצה של שלשות (row, col, seen) שמייצגות את אוסף המספרים על הלוח.

row - מספר השורה.

col - מספר העמודה.

seen - המספר שמופיע על הלוח.

לדוגמא, קבוצת האילוצים של לוח המשחק הבא:

0			3
		5	
1			

תהיה $constraints_set = \{(0, 0, 0), (0, 3, 3), (1, 2, 5), (2, 0, 1)\}$

לאורך כל התרגיל ניתן להניח שכל האילוצים תקינים: יש להם קואורדינטה (שורה ועמודה) שנמצאים בתוך גבולות הלוח, ו-*seen* הוא מספר שלם אי שלילי.

תמונה:

תמונה היא טבלה מרובעת $m \times n$ (n שורות, m עמודות) אשר תאיה ריקים וצבועים בשחור או לבן.

נייצג תמונה ע"י רשימה דו-מימדית המכילה אפסים ואחדות:

- תמונה $m \times n$ תיוצג ע"י רשימה של n רשימות מאורך m , המייצגות את שורות הטבלה.
- אינדקסים של רשימות מתחילים מ-0, לכן נמספר את שורות ועמודות התמונה החל מ-0.
- סדר השורות הוא מלמעלה ללמטה, והעמודות משמאל לימין. כך למשל השורה העליונה היא השורה ה-0, זו שמתחתיה היא השורה ה-1 וכן הלאה עד לשורה התחתונה שהיא השורה ה- $n-1$. באופן דומה העמודה השמאלית ביותר היא העמודה ה-0, והימנית ביותר היא ה- $m-1$.
- הספרה אפס תייצג תא שחור והספרה אחת תייצג תא לבן.

לדוגמא, התמונה הבאה:

תיוצג ע"י

$picture = [[0, 0, 1, 1], [0, 1, 1, 1], [1, 0, 1, 0]]$

שימוש במושגים לתיאור המשחק:

כעת, משהגדרנו את המושגים קבוצת אילוצים ותמונה:

- כל לוח משחק ניתן לייצוג ע"י שלשה $n, m, constraints_set$ המייצגת את מספר השורות, מספר העמודות, וקבוצת האילוצים בהתאמה.
- ניתן לתאר פתרון ללוח משחק ע"י תמונה.

לדוגמא:

השלשה $\{(0, 0, 0), (0, 3, 3), (1, 2, 5), (2, 0, 1)\}$, מתארת את לוח המשחק הבא:

0			3
		5	
1			

ללוח זה יש פתרון יחיד שנ ניתן לתאר ע"י התמונה:

שמיצגת בפיתון ע"י הרשימה הדו-מימדית הבאה:

$[[0, 0, 1, 1], [0, 1, 1, 1], [1, 0, 1, 0]]$

תמונה חלקית:

במהלך התרגיל, תממשו ע"י שימוש ב backtracking אלגוריתם המחפש פתרון כלשהו עבור לוח משחק. בתהליך חיפוש פתרון אנו נמצאים במצבי ביניים בהם אנו מחזיקים **תמונה חלקית** של הלוח, משמע יתכנו תאים שעדיין לא קבענו את צבעם. נייצג תא שערכו עדיין לא נקבע ע"י הערך -1.

לדוגמא, התמונה החלקית הבאה

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

תיוצג ע"י הרשימה

$[[1, 0, 1, 0], [0, 1, -1, 1], [-1, 0, 1, -1]]$

שימו לב: **תמונה** היא מקרה פרטי של **תמונה חלקית**.

לאורך כל התרגיל, ניתן להניח שכל תמונה או תמונה חלקית הן תקינות (הרשימה היא דו מימדית, יש בה לפחות שורה אחת ועמודה אחת, היא מכילה רק ערכים מותרים, וכל הרשימות הפנימיות הן מאותו אורך)

משימות

בחלק זה נממש את המשחק בפיתון. ממשו את הפונקציות הבאות:

1. מציאת מספר התאים ה"נראים" מתא:

1.1 משימה: כתבו פונקציה שמקבלת תמונה חלקית ומיקום על גביה, ומחזירה את מספר התאים ה"נראים" מהתא במיקום זה אם כל התאים הלא ידועים יחשבו כלבנים. תזכורת: תא יכול "להראות" מתא אחר, אם ורק אם שניהם לבנים ונמצאים באותה השורה או העמודה כאשר אין ביניהם תא שחור.

- חתימת הפונקציה:

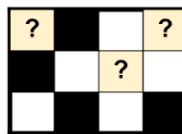
```
def max_seen_cells(picture: List[List[int]], row: int, col: int) -> int
```

- קלט הפונקציה:

- רשימה דו מימדית picture המייצגת תמונה חלקית.
- מספר שלם row המייצג אינדקס של שורה בתמונה החלקית.
- מספר שלם col המייצג אינדקס של עמודה בתמונה החלקית.
- פלט הפונקציה: מספר שלם השווה למספר התאים ה"נראים" מהתא בשורה ה row ועמודה ה col, כאשר תאים לא ידועים נחשבים לתאים לבנים.

- הערות למימוש:

- במקרה בו התא שחור, משמע אין תאים "נראים" ממנו, הפונקציה תחזיר את הערך 0.
- אסור לפונקציה לשנות את הקלט במהלך ריצתה עליו.



לדוגמא, עבור התמונה החלקית הבאה

המיוצגת ע"י:

```
picture = [[-1, 0, 1, -1], [0, 1, -1, 1], [1, 0, 1, 0]]
```

הרצת הפונקציה על הקלטים הבאים תחזיר:

```
max_seen_cells(picture, 0, 0) → 1
max_seen_cells(picture, 1, 0) → 0
max_seen_cells(picture, 1, 2) → 5
max_seen_cells(picture, 1, 1) → 3
```

1.2 משימה: כתבו פונקציה שמקבלת תמונה חלקית ומיקום על גביה, ומחזירה את מספר התאים ה"נראים"

מהתא במיקום זה אם כל התאים הלא ידועים יחשבו כשחורים.

• חתימת הפונקציה:

```
def min_seen_cells(picture: List[List[int]], row: int, col: int) -> int
```

ההנחות והקלט לפונקציה זו דומים לפונקציה max_seen_cells בסעיף הקודם.

עבור הדוגמא picture מסעיף קודם, הרצת הפונקציה על הקלטים הבאים תחזיר:

```
min_seen_cells(picture, 0, 0) → 0
```

```
min_seen_cells(picture, 1, 0) → 0
```

```
min_seen_cells(picture, 1, 2) → 0
```

```
min_seen_cells(picture, 1, 1) → 1
```

2. בדיקת פתרונות:

כתבו פונקציה שמקבלת תמונה חלקית וקבוצת אילוצים, ומחזירה מספר שלם בין 0 ל 2 המעיד על ההצלחה

בסיפוק האילוצים בתמונה החלקית.

לשם כך נגדיר שאילוץ מסויים (row, col, seen)

- מתקיים בדיוק בתמונה חלקית אם ניתן לקבוע במדויק את מספר התאים הנראים מהתא בשורה באינדקס row ובעמודה באינדקס col (צביעת התאים שלא נצבעו עד כה לא תשפיע על ערך זה) ואם מספר זה שווה בדיוק ל seen.
- האילוץ עשוי להתקיים בתמונה החלקית אם הוא לא "מתקיים בדיוק" אבל, seen נמצא בין min_seen_cells לבין max_seen_cells (כולל).
- אחרת האילוץ מופר.

• חתימת הפונקציה:

```
def check_constraints(picture: List[List[int]],  
constraints_set: Set[Tuple[int, int, int]]) -> int
```

קלט הפונקציה:

- רשימה דו מימדית picture המייצגת תמונה חלקית.
- קבוצת אילוצים constraints_set.
- פלט הפונקציה: מספר מתוך {0, 1, 2} הנקבע באופן הבא:
 - 0 אם לפחות אחד מהאילוצים ב constraint set מופר בתמונה החלקית.
 - 1 אם כל האילוצים מתקיימים בדיוק בתמונה החלקית.
 - 2 אחרת (כלומר אם אף אילוץ לא מופר, ויש אילוץ אחד לפחות שלא מתקיים בדיוק).

• הערות למימוש:

- אסור לפונקציה לשנות את הקלט במהלך ריצתה עליו.

לדוגמא, בהינתן

```
picture1 = [[-1, 0, 1, -1], [0, 1, -1, 1], [1, 0, 1, 0]]
picture2 = [[0, 0, 1, 1], [0, 1, 1, 1], [1, 0, 1, 0]]
```

הרצת הפונקציה על הקלטים הבאים תחזיר:

```
check_constraints(picture1, {(0, 3, 5), (1, 2, 5), (2, 0, 1)}) → 0
check_constraints(picture2, {(0, 3, 3), (1, 2, 5), (2, 0, 1)}) → 1
check_constraints(picture1, {(0, 3, 3), (1, 2, 5), (2, 0, 1)}) → 2
```

3. מציאת פתרון:

בסעיף זה נממש פתרון למשחק.

משימה: כתבו פונקציה שמקבלת קבוצת אילוצים וגודל טבלה (מספר שורות ומספר עמודות) המייצגים לוח משחק, ומחזירה תמונה המתארת פתרון אחד של הלוח, אם קיים.

• חתימת הפונקציה:

```
def solve_puzzle(constraints_set: Set[Tuple[int, int, int]],
                 n: int, m: int) -> Optional[List[List[int]]]
```

• קלט הפונקציה:

- קבוצת אילוצים constraints_set.
- n מספר השורות בלוח המשחק.
- m מספר העמודות בלוח המשחק.

• פלט הפונקציה:

- None אם לא קיים פתרון ללוח המשחק המושרה מהקלט.
- אחרת: תמונה אחת, המהווה פתרון ללוח המשחק המושרה.

• הערות למימוש:

- יתכנו מספר פתרונות עבור לוח משחק נתון, על הפונקציה להחזיר רק אחד מהם (לא משנה איזה).
- **אסור לפונקציה לשנות את הקלט** במהלך ריצתה עליו.
- **השתמשו בסעיף זה ב Backtracking.**
- שימו לב שאלמנט חשוב מאוד ב-Backtracking הוא פסילה מוקדמת של פתרונות שידוע שלא יכולים להתקיים (עוד לפני שמשלימים את כל הלוח).
- מותר (אבל לא חובה) לעשות שימוש בפונקציות הקודמות בעת הפתרון.
- שימו לב שבעת צביעת תא בתמונה חלקית רק אילוצים שקשורים לאותה שורה או עמודה של התא עשויים להיות מופרים.

דוגמא לקלטים ופלטים תקינים:

```
solve_puzzle({(0, 3, 3), (1, 2, 5), (2, 0, 1), (0, 0, 0)}, 3, 4) →  
[[0, 0, 1, 1], [0, 1, 1, 1], [1, 0, 1, 0]]
```

```
solve_puzzle({(0, 3, 3), (1, 2, 5), (2, 0, 1), (2, 3, 5)}, 3, 4) → None
```

```
solve_puzzle({(0, 2, 3), (1, 1, 4), (2, 2, 5)}, 3, 3) →  
[[0, 0, 1], [1, 1, 1], [1, 1, 1]]
```

או לחלופין תחזיר:

```
[[1, 0, 1], [1, 1, 1], [1, 1, 1]]
```

4. מציאת מספר הפתרונות ללוח משחק:

משימה: כתבו פונקציה שמקבלת קבוצת אילוצים וגודל טבלה (מספר שורות ומספר עמודות) המייצגים לוח משחק, ומחזירה את מספר הפתרונות שונים שיש ללוח זה.

▪ חתימת הפונקציה:

```
def how_many_solutions(constraints_set: Set[Tuple[int, int, int]],  
                        n: int, m: int) -> int
```

• קלט הפונקציה:

- קבוצת אילוצים constraints_set.
- n מספר השורות בלוח המשחק.
- m מספר העמודות בלוח המשחק.

• פלט הפונקציה:

- מספר הפתרונות השונים שמספקים במדויק את האילוצים הנתונים.

• הערות למימוש:

- **אסור לפונקציה לשנות את הקלט** במהלך ריצתה עליו.
- **השתמשו בסעיף זה ב Backtracking.**
- גם בסעיף זה חשוב לפסול מוקדם פתרונות חלקיים שידוע שלא ייתנו פתרון חוקי לבעיה.
- אין לאגור את כל הפתרונות במבנה נתונים על מנת לספור אותם כי זה תופס הרבה זכרון.

דוגמא לקלטים ופלטים תקינים:

```
how_many_solutions({(0, 3, 3), (1, 2, 5), (2, 0, 1), (2, 3, 5)}, 3, 4) → 0
```



```
how_many_solutions({(0, 3, 3), (1, 2, 5), (2, 0, 1), (0, 0, 1)}, 3, 4) → 1
how_many_solutions({(0, 2, 3), (1, 1, 4), (2, 2, 5)}, 3, 3) → 2
how_many_solutions({(i, j, 0) for i in range(3) for j in range(3)}, 3, 3) → 1
how_many_solutions(set(), 2, 2) → 16
how_many_solutions({(0, 3, 3), (2, 0, 1)}, 3, 4) → 64
```

5. יצירת לוח משחק מפתרון

בסעיף זה (שערכו 5% מניקוד כלל התרגיל) נרצה להפוך את היוצרות, בהינתן תמונה נחפש לוח משחק שתמונה זו היא הפתרון היחיד שלו. נרצה למצוא לוח משחק "חסכוני" – במובן שכל המספרים הרשומים על גביו נחוצים. כלומר, הסרת מספר מהלוח תפר את התנאי שהתמונה מתארת פתרון יחיד עבור הלוח. כתבו פונקציה שמקבלת תמונה ומחזירה קבוצת אילוצים שמקיימת את התנאים הבאים:

- התמונה מתארת פתרון ללוח המשחק המושרה מקבוצת האילוצים וממספר השורות והעמודות שבתמונה.
- פתרון זה הוא יחיד ללוח.
- הסרת איבר כלשהו מקבוצת האילוצים תגרום לכך שהפתרון לא יהיה יחיד יותר.

• חתימת הפונקציה:

```
def generate_puzzle(picture: List[List[int]])
    -> Set[Tuple[int, int, int]]
```

• קלט הפונקציה:

- רשימה דו מימדית picture המייצגת תמונה.

• פלט הפונקציה:

- קבוצת אילוצים המקיימת את התנאים שהוגדרו מעל.

• הערות למימוש:

- יתכנו מספר פלטים שונים תקינים עבור אותו הקלט, על הפונקציה להחזיר רק אחד מהם (לא משנה איזה).
- ניתן להניח קלט תקין.
- אסור לפונקציה לשנות את הקלט במהלך ריצתה עליו.

לדוגמא, עבור התמונה:

```
picture = [[1, 0, 0], [1, 1, 1]]
```

כל הפלטים הבאים תקינים:

```
generate_puzzle(picture) → {(0, 0, 2), (1, 2, 3)}
```

```
generate_puzzle(picture) → {(1, 0, 4), (0, 1, 0), (0, 2, 0)}
```

```
generate_puzzle(picture) → {(1, 0, 4), (0, 0, 2), (0, 2, 0)}  
generate_puzzle(picture) → {(1, 0, 4), (1, 1, 3), (0, 2, 0)}  
generate_puzzle(picture) → {(1, 0, 4), (1, 1, 3), (1, 2, 3)}  
generate_puzzle(picture) → {(1, 0, 4), (0, 1, 0), (1, 2, 3)}  
generate_puzzle(picture) → {(0, 0, 2), (1, 1, 3), (0, 1, 0), (0, 2, 0)}
```

בהצלחה!