בעיית שמונה המלכות

מבוא

בעיית שמונה המלכות היא: כיצד ניתן להציב שמונה מלכות על לוח שח בלי שהן תאיימנה אחת על השנייה. להזכירך, שתי מלכות מאיימות אחת על השנייה עם הן באותה שורה, באותה עמודה או באותו אלכסון.

למעשה נכליל את השאלה ל-n כלשהו. הבעיה שנפתור היא: כיצד ניתן להציב n מלכות בלוח שגודלו n×n?

האלגוריתם הבסיסי

האלגוריתם יתחיל מלוח בו בכל שורה יש מלכה הנמצאת בעמודה אקראית. בכל מהלך יזיז האלגוריתם מלכה יחידה לעמודה אחרת מהעמודה בה היא נמצאת. המלכה תישאר באותה שורה בה הייתה לפני המהלך. האלגוריתם יבצע את המהלך שיקטין ביותר את מספר זוגות המלכות המאיימות זו על זו. אם מספר האיומים יגיע לאפס – נמצא פתרון. אם אין מהלך שיקטין את מספר האיומים – האלגוריתם יחזור לשלב ההתחלתי ויצור לוח אקראי חדש.

לדוגמא, עבור n=4, הלוח ההתחלתי יכול להיות (Q מציין מלכה):

			Q
	Q		
		Q	
Q			

בלוח זה יש שני זוגות של מלכות המאיימות זו על זו (מצאי אותן). הזזת המלכה בשורה השלישית לעמודה האחרונה, יקטין את מספר זוגות המלכות, המאיימות זו על זו, לאחד. אין מהלך אחר, של מלכה בשורה שלה, שיקטין יותר את מספר האיומים (ודאי זאת).

הלוח החדש יראה כך:

Q			
		Q	
			Q
	Q		

אין שום מהלך שיקטין את מספר האיומים, ולכן האלגוריתם יגריל לוח התחלתי חדש.

האלגוריתם מבצע חיפוש מקומי, אקראי וחמדני

אלגוריתם שמחפש את מצב המטרה תוך התקדמות מהמצב הנוכחי למצב קרוב, ללא צורך לזכור מצבים בהם כבר נתקל (כמו האלגוריתם שהוצג בסעיף הקודם), נקרא חיפוש מקומי. הוא דורש פחות זיכרון מחיפוש שזוכר מצבים בהם נתקל בעבר, ובדרך כלל הוא פשוט יותר.

אלגוריתם אקראי הוא אלגוריתם שמשתמש בהגרלות כדי לפתור בעיה או שפותר בעיה בהסתברות מסוימת ולא בוודאות. האלגוריתם שהוצג מגריל את המצב ההתחלתי, ולכן הוא אקראי. בנוסף, מספר האיטרציות עד להגעה לפתרון אינו קבוע אלא תלוי במצב ההתחלתי שהוגרל. הרצה "חסרת מזל" עלולה להגריל שוב ושוב מצבים התחלתיים שלט ניתן להגיע מהם ללוח ללא איומים.

אלגוריתם חמדני הוא אלגוריתם, שבכל איטרציה, בוחר באיזשהו "הכי" התלוי במצב הנוכחי בלבד ("הכי" זה נקרא אופטימום לוקלי). האלגוריתם שהוצג בוחר לעבור למצב עם הכי מעט איומים (ממצב הלוח הנוכחי, ובלי להתחשב במצבים בהם כבר נתקל או שהוא עלול להיתקל בהם), ולכן הוא חמדני.

ייצוג הלוח

לוח בגודל n×n ייוצג ברשימה באורך n. כל איבר ברשימה יהיה מספר בין שלם בין 0 ל-n-1. כל איבר ברשימה ייצג שורה בלוח, וערכו ייצג את העמודה בה ניצבת המלכה, בשורה זו.

את הלוח הראשון, מהסעיף הקודם, תייצג הרשימה [1, 0, 2, 0]. את הלוח השני תייצג הרשימה [1, 3, 2, 0].

כיצד ימומש האלגוריתם בפייתון? פתחי את הקובץ queens.py, ותראי.

בניית הלוח האקראי ההתחלתי – הפונקציה rndboard(n)

הפונקציה מקבלת את n, מספר חיובי שלם וגדול מ-3 (למה על n להיות גדול משלוש?). היא מחזירה רשימה באורך n של שלמים אקראיים בטווח 0…n-1. הרשימה המוחזרת מייצגת לוח אקראי בגודל n×n, כמוסבר בסעיף הקודם.

עליך להשלים את הקוד בפונקציה. עליך להוסיף לולאה שתכניס ל-board את הערכים הנדרשים. הלולאה תחליף את ההערה. הריצי את הפונקציה על ערכים שונים של n, ובדקי שהיא מחזירה רשימה כנדרש.

חישוב מספר זוגות המלכות המאיימות זו על זו – הפונקציה threats(board)

הפונקציה threats מקבלת את הלוח board (המיוצג ברשימה, כפי שראית קודם).
הפונקציה סופרת, בעזרת המשתנה count, את מספר זוגות המלכות המאיימות זו על זו בלוח. המימוש שמופיע בקובץ queens.py שגוי ועליך לתקן אותו. הקוד הוא:

```
def threats(board):
    count = 0
    for i in range(len(board)-1):
        for j in range(i+1, len(board)):
            if board[i] == board[j]
                count += 1
    return count
```

השורה הראשונה מאפסת את count – מונה האיומים. השורה האחרונה מחזירה את ערכו, לאחר שנספרו מספר האיומים.

הדולאה (דולאה – הלחם של דו ולולאה. שם קצר ללולאה בתוך לולאה) באדום עוברת על כל זוגות האיברים ברשימה board, שהם כל זוגות המלכות על הלוח. הריצי ידנית, בטבלת מעקב, את הדולאה, וודאי שאת מבינה כיצד הדולאה עוברת על כל הזוגות.

ה-if, בירוק, בודק האם זוג המלכות מאיימות זו על זו, ואם כן המונה counter גדל ב-1. הבדיקה שגויה, ועליך לתקן אותה. שתי מלכות יכולות לאיים אחת על השנייה בשלושה אופנים:

1. שתי המלכות באותה שורה
2. שתי המלכות באותה עמודה
3. שתי המלכות באותו אלכסון

אופן הייצוג של הלוח מונע איומים מהסוג הראשון. התנאי ב-if בודק איומים מהסוג השני. איומים מהסוג השלישי לא נבדקים. עליך להרחיב את ה-if כך שיבדוק גם האם המלכה בשורה i והמלכה בשורה j נמצאות על אותו אלכסון.

ודאי שבפונקציה מחזירה ערך נכון. העבירי לפונקציה לוחות שונים ובדקי אם הערך שהיא מחזירה הוא, אכן, מספר זוגות המלכות המאיימות זו על זו.

הקטנת מספר האיומים בלוח – הפונקציה improve(board)

הפונקציה מקבלת את הלוח board (המיוצג ברשימה, כפי שראית קודם). הפונקציה מזיזה כל מלכה לכל העמודות בשורה שלה ובודקת איזה מהלך ממזער את מספר האיומים. לאחר שהפונקציה מסתיימת, board מכיל את מצב הלוח אחרי המהלך שממזער את מספר האיומים. הפונקציה מחזירה את מספר זוגות המלכות המאיימות זו על זו בלוח. הקוד נראה כך:

```
def improve(board):
    minimum = threats(board)
    improved = [0,board[0]]
    for r in range(len(board)):
        tmp = board[r]
        cols = list(range(len(board)))
        cols.remove(board[r])
        for c in cols:
            board[r] = c
            x = threats(board)
            if x < minimum:
                minimum = x
                improved = [r,c]
        board[r] = tmp
    return minimum
```

minimum הוא המשתנה שיחזיק את מספר האיומים המינימלי שנמצא עד כאן. improved מכיל את המהלך שיוביל למינימום האיומים הנ"ל. improved הוא רשימה של שני איברים. הראשון הוא השורה והשני העמודה של המיקום החדש של המלכה.

שתי השורות הראשונות מאתחלות את minimum למספר האיומים ב-board ואת improved למלכה בשורה הראשונה במיקומה הנוכחי.

הלולאה הירוקה עוברת על כל המלכות. גוף הלולאה יציב את המלכה בכל עמודה בשורה (חוץ מהעמודה בה היא נמצאת), ויבדוק איזה מיקום ממזער את מספר האיומים. השורות באדום שומרות את המיקום הנוכחי של המלכה (השורה הראשונה בגוף הלולאה) , ומשחזרות אותו (השורה האחרונה בגוף הלולאה). זה נעשה כדי להתחיל את הבדיקה, לכל מלכה, מ-board כפי שנשלח לפונקציה. השורות באפור מציבות ב-cols את רשימת כל

העמודות, חוץ מאשר העמודה בה נמצאת המלכה כעת. הלולאה התכולה מציבה את המלכה בכל אחת מן העמודות ב-cols ובודקת האם מספר האיומים מינימלי. אם כן, מספר האיומים המינימלי והמיקום נשמרים.

השורה האחרונה מחזירה את מספר האיומים בלוח החדש (או בלוח המקורי, אם לא נמצא מהלך להקטנת מספר האיומים). board מכיל את אותם ערכים שהכיל בתחילת ריצת הפונקציה. עלייך להוסיף, לפני פקודת ה-return, קוד שיבצע את המהלך שממזער את מספר האיומים בלוח (אם יש מהלך כזה...).

ודאי שהפונקציה מחזירה ערך נכון ומשנה את board, כנדרש. כעת תוכלי להשלים את הקוד של הפונקציה solve, הפותרת את החידה.

פתרון חידת n המלכות – הפונקציה solve(size)

הפונקציה solve מקבלת את הפרמטר size, ומדפיסה פתרון לחידה. מספר המלכות הוא size, וגודל הלוח הוא size×size. גם את הקוד של פונקציה זו יהיה עליך להשלים בעצמך.

```
def solve(size):
    b = rndboard(size)
    n = threats(b)
    while n>0:
        x = improve(b)
        if x == n:
            pass
    printboard(b)
```

השורות האדומות יוצרות לוח אקראי התחלתי, ובודקות את מספר הזוגות של מלכות שמאיימות זו על זו.

לולאת ה-while מתבצעת כל עוד לא נמצא לוח שאין בו איומים. קריאה לפונקציה improve מנסה להקטין את מספר האיומים. ה-if הירוק בודק האם מספר האיומים קטן לאחר הקריאה ל-improve. עליך למחוק את הפקודה pass, ולהשלים את הפקודה בעצמך. אם מספר האיומים לא השתנה, יש ליצור לוח אקראי חדש. אם מספר האיומים קטן יש להמשיך ולנסות להקטין את מספר האיומים.

השורה האחרונה, באפור, קוראת לפונקציה printboard אשר מדפיסה את הפתרון לחידה. עליך לכתוב בעצמך פונקציה זו. צורת ההדפסה נתונה להחלטתך. העיקר, שמהפלט יהיה ברור מהו פתרון החידה.

עכשיו תוכלי להריץ את solve(8) כדי לראות פתרון לבעיית שמונה המלכות.

שאלות נוספות

- נסי להריץ את solve עם מספר מלכות הולך וגדל. מה מספר המלכות המקסימלי עבורו נמצא פתרון בזמן סביר? מהו לדעתך זמן סביר?
- הרחיבי את הקוד כך שיספור כמה פעמים האלגוריתם יוצר לוחות אקראיים חדשים. נסי להעריך את הקשר בין מספר המלכות למספר הלוחות האקראיים שנוצרו.
- הרחיבי את הקוד כך שיספור כמה פעמים האלגוריתם יוצר לוחות אקראיים חדשים. הריצי את solve(20) מספר פעמים. מה המספר המינימלי של לוחות אקראיים שנוצרו? מה המספר המקסימלי של לוחות אקראיים שנוצרו?