תכנות מוכוון עצמים 31632

פתרון מבחן סופי, מועד א', סמסטר ב' תשע"ד, 04/07/2014

שאלה 1 [30%]

השאלה מתייחסת לפרוייקט הסודוקו (Sudoku) שהתבצע במהלך הקורס.

א. רשום מפרט מבנה נתונים מופשט (Abstract Data Type) עבור תשבץ הסודוקו. השתדל לכלול בו רק את המתודות הציבוריות החשופות ללקוח (ולהסתיר מתודות פרטיות שאינן מעניינו של הלקוח)

ב. רשום קוד עבור המתודה resolve_singletons(self) לשם פיצוח התאים הריקים שיש להם פיתרון אחד ויחיד. תן הסבר מילולי קצר לאלגוריתם שמתבצע במתודה זו ותן הערכה מקורבת לגבי סיבוכיות הזמן שלו.

ג. עשה שימוש במחלקה Sudoku אשר בנית במהלך הסמסטר בכדי לכתוב פונקציה בשם fivers() אשר סופרת את כל התשבצים החוקיים (כולל אלכסונים) אשר לפחות שורה אחת בהן מכילה את הסדרה 12345. הסבר מילולי קצר ומשכנע ייתקבל באופן חלקי.

תשובה:

א. ייתכנו כמובן מספר תשובות אפשריות (בהתאם לדרישות אפשריות מצד לקוחות שונים). כמובן נתנה התחשבות מלאה בכל התשובות האפשריות שניתנו כתשובה לשאלה. דוגמא אופיינית לתשובה:

```
s = Sudoku(board)                    Constructor
    Create a new Sudoku object from a text string (board) which describes
    a Sudoku puzzle in a given string represntation like:
              0 0 3 0 2 0 6 0 0
              9 0 0 3 0 5 0 0 1
              0 0 1 8 0 6 4 0 0
              0 0 8 1 0 2 9 0 0
              7 0 0 0 0 0 0 0 8
              0 0 6 7 0 8 2 0 0
              0 0 2 6 0 9 5 0 0
              8 0 0 2 0 3 0 0 9
              0 0 5 0 1 0 3 0 0
    or:
      "700603000900000200000000000309100000000050802000004000200400000007000300800000000"

s.row(i)                             Accessor
    Get row i of the board as a list of 9 integers
    The integer 0 (zero) indicates an empty cell

s.col(j)                             Accessor
    Get column j of the board as a list of 9 integers
    The integer 0 indicates an empty cell

s.block(i, j)                        Accessor
    Get the block which contains the cell (i,j) as a list of 9 integers
    Ordered from top/down left/right
    A zero indicates an empty cell

s.diagonals()                        Accessor
    Get the two diagonals of the board as a pair of two lists of
    9 integers

s.solve()                            Mutator
    Compute all solutions to the puzzle

s.solutions                          Accessor
    A class member – list of all solutions to the puzzle
    as Sudoku objects
```

ב. ישנם כמובן דרכים רבות ומגוונות לפתור את הבעייה. הדגש צריך להיות על בחירת אלגוריתם חסכוני בזמן ומקום ככל האפשר. להלן דוגמא אפשרית (אך לא בהכרח הכי יעילה!):

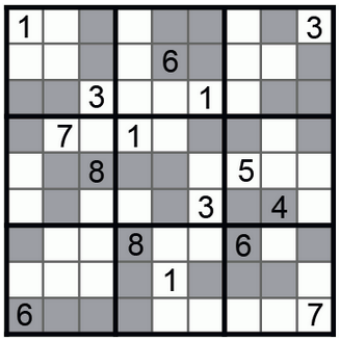
```
1  def resolve_singletons(self):
2      while True:
3          self.compute_values()
4          if all(len(self.values[s]) > 1 for s in self.values):
5              return True
6          for s,vals in self.values.items():
7              if len(vals) == 0:   # invalid cell! puzzle cannot be solved
8                  return False
9              elif len(vals) == 1:
10                 d = vals.pop()
11                 self.cell[s] = d
```

המתודה compute_values() מחשבת את קבוצת כל הערכים האפשריים שניתן להציב בכל תא ריק בתשבץ, ושומרת את כל המידע הזה במילון self.values (אשר המפתחות שלו הם התאים הריקים בלבד!).
בשורה 2 נפתחת לולאה אינסופית, אשר הדרך היחידה לצאת ממנה היא על ידי הבדיקה שמתבצעת בשורה 4: במידה ולכל התאים הריקים יש יותר מערך חוקי אחד אפשרי, אז הלולואה נעצרת (ולמעשה מתבצעת יציאה מהמתודה עם ערך מוחזר True).
אחרת, עבור כל תא ריק, מתבצעות שתי בדיקות: א. אם קבוצת הערכים שלו ריקה, חוזרים מיד מהמתודה עם ערך False (שמשמעותו שהלוח הנוכחי פסול ואין לו פיתרון), ב. אם יש רק ערך יחיד שניתן להציב בתא, אז ערך זה מוצב בתוך הלוח כערך קבוע (ההצבה מתבצעת במילון self.cell).
סיבוכיות הזמן של האלגוריתם: גודל הקלט (תשבץ סודוקו) הוא n (במקרה שלנו n=81). לולאת ה-while החיצונית יכולה להימשך לכל היותר n פעמים (מאחר ובכל שלב, דרוש למצוא לפחות סינגלטון אחד, ויש לכל היותר n תאים). לולאת ה-for הפנימית עוברת על רשימת התאים, שגם גודלה הוא לכל היותר n.
לכן סיבוכיות הזמן של האלגוריתם הנ"ל הוא $O(n^2)$.
תשובה זו כמובן מתאימה לאלגוריתם הנ"ל וייתכן שעבור אלגורימים אחרים היא תהיה שונה!
ג. ניתן לפתור את הבעייה בשתי דרכים עקריות: כפונקציה פשוטה המשתמשת במחלקה Sudoku באופן רגיל לחלוטין, או באמצעות ירושה. כל אחת מהדרכים תתקבל באופן מלא:

```
def fivers():
    Sudoku.max_solutions = float('Infinity')
    Sudoku.check_diagonals = True
    five = [1,2,3,4,5]
    s = Sudoku(81 * '0')
    s.solve()
    result = 0
    for sol in s.solutions:
        for i in range(1,10):
            row = sol.row(i)
            for j in range(1,6):
                if row[j:j+5] == five:
                    result += 1
                    break
    return result
```

אך פיתרון זה הוא בלתי יעיל באופן מחריד! אנו למעשה עוברים על כל לוחות הסודוקו הפתורים שיש ומחפשים בתוכם את הסדרה "12345" (בכל שורה). זה עשוי לארוך המון זמן וכמובן מתבצעים כאן המון חישובים מיותרים. אך זהו פיתרון נכון. פיתרון יעיל יותר ניתן לקבל באמצעות שימוש בירושה באופן דומה מאוד לפיתרון של השאלה הבאה (EvenOddSudoku).

שאלה 2 [30%]



השאלה מתייחסת לפרוייקט הסודוקו (Sudoku) שהתבצע במהלך הקורס.
אחת הגירסאות המעניינות של המשחק נקראת EvenOddSudoku.
יש שני סוגי משבצות בתשבץ: משבצות לבנות ומשבצות אפורות. במשבצות האפורות ניתן להציב מספרים זוגיים בלבד, ואילו במשבצות הלבנות ניתן להציב מספרים אי-זוגיים בלבד. כל שאר הכללים של התשבץ הרגיל נשארים בתוקפם גם עבור תשבץ זה (אפשר להתעלם מאלכסונים לחלוטין).
עליך לתכנן מחלקה חדשה בשם EvenOddSudoku לשם פיתרון סוג כזה של תשבצים. כמובן, בכדי לחסוך במאמץ, עליך להשתמש בירושה.

א. האם קיים הבדל בין מפרט מבנה נתונים מופשט (ADT) של Sudoku ובין זה של EvenOddSudoku? אם כן, ציין את ההבדל בלבד (בפרט בכל מה שקשור לבדיקת תקינות ובדיקת נכונות פיתרון).

ב. רשום קוד עבור מתודת הבנאי (constructor) של המחלקה EvenOddSudoku ותן הסבר מילולי קצר למה שמתבצע בתוכה.

ג. רשום קוד עבור מתודה is_solved(self) המקבלת תשבץ כזה ובודקת אם הוא פתור סופית.
רמז: בכדי לפשט, התחל עם מתודה בשם color_check(self) הבודקת את חוקיות הערכים על פי צבעי המשבצות, ורק לאחר מכן נסה לרשום את is_solved(self).

ד. אילו מתודות נוספות של המחלקה Sudoku ייצטרכו לעבור "דריסה" במסגרת המחלקה EvenOddSudoku? במידת האפשר רשום קוד מפורט אך גם הסבר מילולי משכנע יזכה להערכה מספקת.

ה. האם יהיה צורך לבצע תיקון כלשהו במחלקה Sudoku בכדי שהמחלקה EvenOddSudoku (היורשת את המחלקה Sudoku) תעבוד נכון? תן הסבר מילולי קצר לתשובתך.

תשובה:

א. אין שום הבדל במפרט של EvenOddSudoku מלבד במתודת הבנאי (constructor) שמלבד תאור הלוח, צריכה גם לקבל את רשימת התאים האפורים:

```
s = EvenOddSudoku(board, gray_squares)                    Constructor
    Create a new EvenOddSudoku object from a text string (board) which
    describes a Sudoku puzzle in a given string and list of gray squares like:

    gray_squares = [
       (1,3), (1,5), (1,6), (1,8),
       (2,3), (2,5), (2,6), (2,9),
       (3,1), (3,2), (3,8), (3,9),
       (4,1), (4,6), (4,7), (4,9),
       (5,2), (5,3), (5,4), (5,5),
       (6,2), (6,5), (6,7), (6,8),
       (7,1), (7,4), (7,7), (7,9),
       (8,4), (8,6), (8,7), (8,8),
       (9,1), (9,2), (9,3), (9,4),
    ]
    board = "700603000900000200000000000309100000000050802000004000200400000007000300800000000"
     s = EvenOddSudoku(board, gray_squares)
```

לכן, ההבדל הקל הזה מצביע על כך שיש להשתמש בירושה במקום לממש את ה-ADT הזה במלואו.

ב. גם כאן ייתכנו דרכים שונות לפתרון הבעייה: ללא ירושה (לא יעיל), ובאמצעות שימוש בירושה (inheritance). במקרה הנוכחי, שימוש בירושה נדרש בגוף השאלה.
ניתן לפתור את הבעייה בדרך הקצרה הבאה:

```
from sudoku import Sudoku, squares

class EvenOddSudoku(Sudoku):
    def __init__(self, text, gray_squares):
        Sudoku.__init__(self, text)
        self.gray_squares = gray_squares
        self.white_squares = [sq for sq in squares if not sq in self.gray_squares]

    def cell_values(self, i, j):
        values = []
        for v in Sudoku.cell_values(self, i, j):
            if int((i,j) in self.gray_squares) == 1 - v%2:
                values.append(v)
        return values

    def copy(self):
        return EvenOddSudoku(str(self), self.gray_squares)
```

בפתרון זה מתבצעת דריסה של המתודה cell_values ו-copy בלבד, ולכן אין צורך בשינוי כלשהו של המתודה is_solved (לכן כל מי שפתר את הבעייה בדרך זו, לא היה צריך לכתוב את is_solved וגם את color_check).

<u>פתרון שני לבעייה</u>:

```
class EvenOddSudoku(Sudoku):
    def __init__(self, text, gray_squares):
        Sudoku.__init__(self, text)
        self.gray_squares = gray_squares
        self.white_squares = [sq for sq in squares if not sq in self.gray_squares]

    def is_valid(self):
        if not self.color_check():
            return False
        return Sudoku.is_valid(self)

    def is_solved(self):
        if not self.color_check():
            return False
        return Sudoku.is_solved(self)

    def color_check(self):
        for sq in squares:
            data = self.cell[sq]
            if data == 0:
                continue
            if sq in self.gray_squares:
                if data%2 == 1:
                    return False
            else:
                if data%2 == 0:
                    return False

        return True

    def copy(self):
        return EvenOddSudoku(str(self), self.gray_squares)
```

ד. במקרה של הפיתרון הראשון, רק המתודה cell_values (ו-copy) נדרסת. בפיתרון השני, צריך לדרוס שתי מתודות: is_valid וגם is_solved. בפיתרון השני יש כמובן להוסיף את המתודה color_check. אפשר גם לשדרג את המחלקה Cell ומתודת ה-draw בכדי לאפשר ציור של תאים אפורים, אך זה לא נדרש באמת ...
ה. בכדי שהירושה תעבוד, יש צורך לתקן מעט את מחלקת הבסיס Sudoku: בכדי ליצר עותקים (clones) של עצמי Sudoku (או עצמי EvenOddSudoku) יש צורך בהוספת מתודת copy למחלקת הבסיס:

```
    def copy(self):
        return Sudoku(str(self))
```

ויש לתקן את המתודות solve ו- solve_ כך שבכל מקום שבו מתבצעת העתקה, משתמשים במתודה copy.
במחלקה EvenOddSudoku (בשני הפיתרונות) יש לדרוס את המתודה copy באופן הבא:

```
    def copy(self):
        return EvenOddSudoku(str(self), self.gray_squares)
```

שאלה 3 [40%]

לפניך מפרט של מבנה נתונים מופשט של טבלה (Table)

```
t = Table(m, n, value=None)
    Create a new table with number of rows = m, number of columns = n
    Row and columns indices start with 0
    all mxn cells initialized to value

t.numRows()
    Return the number of  rows in the table t

t.numCols()
    Return the number of columns in the table t

t.setitem(i, j, value)
    Sets table cell (i,j) to value
    Both indices i,j must be within valid bounds: 0<=i<nrows, 0<=j<ncols

t.getitem(i, j)
    Get the value of the cell (i,j)
    Both indices must be within valid bounds: 0<=i<m, 0<=j<n

t.clear(value)
    Set all table elements to value

t.row(i)
    Return values in row i as a list

t.col(j)
    Return columns j as a list
```

א. רשום טסט קצר היוצר טבלה בגודל 3x4 של מספרים שלמים הבוחן את כל המתודות של מבנה נתונים זה.
ב. ממש את מבנה הנתונים הנ"ל באמצעות מחלקה (class) בשם Table בשפת התיכנות Python. תוכל להשתמש בכל מבני הנתונים הבסיסיים של שפת Python בלבד (אין להשתמש בספריות חיצוניות!).

ג. בנה מחלקה בשם SquareMatrix שיורשת את המחלקה Table שבה מספר השורות שווה למספר העמודות (מטריצה ריבועית!) ומוסיפה שתי מתודות נוספות עבור חיבור וכפל של מטריצות.

ד. רשום טסט הבודק את פעולות החיבור והכפל של המחלקה SquareMatrix.
תוכל למשל להשתמש במטריצות בגודל 2x2.

תשובה:

א.

```
def test():
    t = Table(3, 4, 0)
    assert t.numRows() == 3
    assert t.numCols() == 4
    t.setitem(2, 1, 349)
    assert t.getitem(2,1) == 349
    assert t.row(2) == [0, 349, 0, 0]
    assert t.col(1) == [0, 0, 349]
    t.clear(37)
    for i in range(t.num_rows):
        for j in range(t.num_cols):
            assert t.getitem(i, j) == 37
    print "Test PASSED"
```

ב. יש בעיקרון שתי דרכים לייצג טבלה: כרשימת שורות, או כמילון שהמפתחות שלו הם כניסות (i,j).
נתחיל בפיתרון הראשון:

```
class Table:
    def __init__(self, m, n, value=None):
        self.num_rows = m
        self.num_cols = n
        self.rows = []
        for i in range(m):
            row = n * [value]
            self.rows.append(row)

    def setitem(self, i, j, value):
        self.rows[i][j] = value

    def getitem(self, i, j):
        return self.rows[i][j]

    def numRows(self):
        return self.num_rows

    def numCols(self):
        return self.num_cols

    def clear(self, value=None):
        for i in range(self.num_rows):
            for j in range(self.num_cols):
                self.rows[i][j] = value

    def row(self, i):
        return self.rows[i]

    def col(self, j):
        col = []
        for i in range(self.num_rows):
            col.append(self.rows[i][j])
        return col
```

פתרון שני המבוסס על מילון:

```
class Table:
    def __init__(self, m, n, value=None):
        self.num_rows = m
        self.num_cols = n
        self.dict = dict()
        for i in range(m):
            for j in range(n):
                self.dict[i,j] = value

    def setitem(self, i, j, value):
        self.dict[i,j] = value

    def getitem(self, i, j):
        return self.dict[i,j]

    def numRows(self):
        return self.num_rows

    def numCols(self):
        return self.num_cols

    def clear(self, value=0):
        for i in range(self.num_rows):
            for j in range(self.num_cols):
                self.dict[i,j] = value

    def row(self, i):
        row = []
        for j in range(self.num_cols):
            row.append(self.dict[i,j])
        return row

    def col(self, j):
        col = []
        for i in range(self.num_rows):
            col.append(self.dict[i,j])
        return col
```

ג.

```
from table import Table

class SquareMatrix(Table):
    def __init__(self, n, value=0):
        Table.__init__(self, n, n, value)

    def __add__(self, other):
        if other.num_rows != self.num_rows:
            raise Exception("Invalid matrix size")
        mat = SquareMatrix(self.num_rows)
        for i in range(self.num_rows):
            for j in range(self.num_rows):
                value = self.getitem(i,j) + other.getitem(i,j)
                mat.setitem(i,j, value)
        return mat

    def __mul__(self, other):
        if other.num_rows != self.num_rows:
            raise Exception("Invalid matrix size")
        n = self.num_rows
        mat = SquareMatrix(n)
        for i in range(n):
            row = self.row(i)
            for j in range(n):
                col = other.col(j)
                value = 0
                for k in range(n):
                    value += row[k] * col[k]
                mat.setitem(i,j,value)
        return mat
```

ד.

```
def test():
    a = SquareMatrix(2, 2)
    b = SquareMatrix(2, 3)

    c = a + b
    d = a * b
    for i,j in [(0,0), (0,1), (1,0), (1,1)]:
        assert c.getitem(i,j) == 5
        assert d.getitem(i,j) == 12

    print "TEST PASSED"
```