המכללה האקדמית להנדסה
אורט בראודה
ORT BRAUDE COLLEGE

המחלקה להנדסת חשמל ואלקטרוניקה

מבני נתונים ואלגוריתמים 31632

פתרון מבחן סופי, מועד ב', סמסטר א' תשע"ד, 24/02/2014

<u>הוראות לנבחן:</u> משך הבחינה שלוש שעות. חומרי העזר המותרים הם השקפים של הקורס שנמסרו באתר הקורס בלבד. שימוש במחשבונים ומכשירים אלקטרוניים אסור בהחלט. רשום את תשובותיך במחברת המצורפת לבחינה. ציין בבירור את מספר השאלה במחברת והשתדל להיות <u>קצר ועניני</u> (תשובות ארוכות מדי או נסיונות לתת מספר תשובות אפשריות <u>ייפסלו</u> את תשובתך!). הקפד על כתב יד ברור ומסודר, ומחק את כל חומר הטיוטא. השימוש בשפה האנגלית מותר. השאלון מכיל 6 שאלות, ונפרש על פני 3 עמודים.
בהצלחה !

<u>שאלה 1</u> [20%]

להלן מיפרט ADT של מבנה נתונים בשם Points. זהו מבנה נתונים המורכב מאוסף של נקודות במישור הגאומטרי הדו-מימדי. כל נקודה במישור תיוצג על ידי זוג סדור (x,y), כאשר x,y הם מספרים ממשיים (float):

```
p = Points()
   Create a new empty Points object p
p.add(x,y)
   Add a new point (x,y)
p.delete(x,y)
   Remove the point (x,y) if it is in p, otherwise do nothing
p.contains(x,y)
   Return True if p contains the point (x,y), otherwise returns False
p.xsection(c)
   Return list of all points (x,y) in p, such that x==c
p.ysection(c)
   Return list of all points (x,y) in p, such that y==c
```

א. כתוב מחלקה בשם Points בשפת Python אשר ממשת את המיפרט (ADT) הנ"ל באמצעות מבני הנתונים הבסיסיים של Python בלבד (כגון: list, set, tuple, dictionary).

ב. מהי הסיבוכיות זמן של פעולות xsection ו-ysection שקיבלת במימוש שלך?

ג. האם ניתן לממש את מבנה הנתונים Points באופן שונה כך שסיבוכיות פעולות xsection ו-ysection תהיה O(1)? אם כן, הצע רעיון כיצד זה ניתן לביצוע. אם לא, הסבר מדוע זה בלתי אפשרי?

<u>פתרון:</u>

```
class Points:
    def __init__(self):
        self.xpoints = dict()
        self.ypoints = dict()

    def add(self, x, y):
        if not x in self.xpoints:
            self.xpoints[x] = set()
        if not y in self.ypoints:
            self.ypoints[y] = set()
        self.xpoints[x].add((x,y))
        self.ypoints[y].add((x,y))

    def delete(self, x, y):
        self.xpoints[x].discard((x,y))
        self.xpoints[y].discard((x,y))
```

```
def contains(self, x, y):
        return (x,y) in self.xpoints[x]

def xsection(self, c):
    if c in self.xpoints:
        return self.xpoints[c]
    else:
        return []

def ysection(self, c):
    if c in self.ypoints:
        return self.ypoints[c]
    else:
        return []
```

א. נשתמש בשני מילונים (self.xpoints, self.ypoints) בכדי לאחסן את הנקודות שלנו. כל נקודה (x,y) תתאכסן למעשה בשני המילונים ביחד (כך שאנו נבזבז קצת שטח זיכרון אך נרויח מהירות). המילון הראשון ממפה מפתח x לקבוצת כל הנקודות שזו הקואורדינטה שלהם. יש לשים לב לכך ש-self.xpoints[x] הוא מבנה נתונים מסוג set. באופן דומה, המפתחות של המילון self.ypoints הוא מספרים ממשיים y, כך ש self.ypoints[y] הוא קבוצת הנקודות בעלות הקואודינטה y.

ב. בצורה זו, כל מה שנדרש בכדי לקבל את p.xsection(c) הוא ביצוע שתי פעולות פשוטות:

1. בדיקה אם c הוא מפתח במילון p.xpoints ?
2. אם כן, מחזירים את p.xpoints[c].
3. אם לא, מחזירים רשימה ריקה [ ]

טענה זהה חלה על המתודה p.ysection(c).

ג. כמובן התשובה לשאלה זו חיובית כפי שהראינו בסעיף הקודם.

שאלה 2 [6%]

נתונה הרשימה הבאה למיון:

L = [3, 2, 1, 6, 5, 4, 9, 8, 7, 12, 11, 10, …, 303, 302, 301]

כמה סבבים של האלגוריתם bubble_sort יידרשו בכדי למיין את L? כמה פעולות החלפת איברים יידרשו?

בצד שמאל רשום האלגוריתם bubble_sort הדרוש לביצוע המיון של הרשימה L. אין צורך לנמק. נדרשות תשובות סופיות בלבד.

<u>הערה:</u> סבב מוגדר כפעולה מלאה של גוף הלולאה while של האלגוריתם.

```
def bubble_sort(L):
    N = len(L)
    while True:
        sorted = True
        for i in range(0,N-1):
            if L[i+1] < L[i]:
                sorted = False
                L[i], L[i+1] = L[i+1], L[i]
        if sorted:
            return
```

<u>פתרון:</u>

מספיק יהיה לבחון את כל אחת מהשלשות … ,[9,8,7] ,[6,5,4] ,[3,2,1] בנפרד, מאחר שהאלגוריתם יסדר כל אחת מהשלשות האלה בנפרד ולא תתבצע החלפת איברים ביניהם. כל אחת מהשלשות תסתדר באופן הבא:

```
[3, 2, 1]    round 1
[2, 1, 3]    round 2
[1, 2, 3]    round 3
```

נדרשים אם כן שלושה סבבים לסדר את כל הרשימה בת 303 האיברים. בסיבוב הראשון התבצעו 2 החלפות, ובסיבוב השני התבצעה החלפה אחת. מכאן שנדרשים 3 החלפות עבור כל שלושה. יש לנו 101 שלושת כאלה, ולכן נדרשים 303 החלפות בכדי לסדר את הרשימה.

הערה: כל מי שלא ספר את המצב ההתחלתי כסיבוב קיבל את מלוא הניקוד (אפשר לכלול או לא לכלול אותו וזה לא משנה כלל)

שאלה 3 [32%]

בכל האלגוריתמים של שאלה זו, מותר להשתמש במתודות של list בלבד! (אין להשתמש ב- set או dict!)

א. רשום אלגוריתם יעיל ככל האפשר בשם zero_indices(L) בשפת Python המקבל רשימת מספרים שלמים L, ומחזיר זוג אינדקסים (i,j) שעבורם L[i]+L[j]==0 אם ישנם כאלה, אחרת יחזיר None.
לדוגמא:

```
L1 = [7, -2, 5, -6, 1, 2]
L2 = [2, 1, 9, -5]
zero_indices(L1) = (1, 5)
zero_indices(L2) = None
```

ב. הערך מהי סיבוכיות הזמן של האלגוריתם שמצאת בסעיף הקודם?

ג. האם האלגוריתם שמצאת הוא היעיל ביותר שייתכן? אם אתה סבור שלא, נסה להציע רעיון לאלגוריתם יעיל יותר.

ד. בנה אלגוריתם יעיל ככל האפשר בשם lequal(A,B) המקבל שתי רשימות של מספרים שלמים A, B, ומחזיר ערך בוליאני True אם הרשימות מכילות את אותן איברים (לא בהכרח באותו סדר או מספר כפילויות). במידה ולא, האלגוריתם יחזיר False. לדוגמא:

```
A = [8, 3, -5, 17, 3, -5, 3, 8, 3]
B = [3, 17, -5, 17, 8, 17, 17]
C = [17, -5, 1, 3, 8]
lequal(A,B) = True
lequal(A,C) = False
```

ה. הערך מהי סיבוכיות הזמן של האלגוריתם שמצאת בסעיף הקודם?

ו. האם האלגוריתם שמצאת עבור lequal(A,B) היעיל ביותר שייתכן? אם אתה סבור שלא, נסה להציע רעיון לאלגוריתם יעיל יותר.

<u>הערה:</u> בכל האלגוריתמים של שאלה זו, מותר להשתמש במתודות של list בלבד! (אין להשתמש במבני נתונים set או dict!). זוג האינדקסים (i,j) אינו חייב להיות על פי סדר כלשהו וגם אין הכרח ש-i≠j.

<u>פתרון:</u>

א. הרעיון זהה לתרגיל 5 שניתן בפרוייקט 5.
ראשית כל יש למיין את הרשימה A, לחשב את אורכה n ואז לרוץ עם שני אינדקסים i, j – הראשון מתחילת הרשימה מתקדם קדימה, והשני מסוף הרשימה הולך אחורה. אם A[i]+A[j] שלילי מקדמים את i, אם חיובים מסיגים את j.

ב. בכדי למיין את A, נדרשת סיבוכיות O(n log n), וברור כי האלגוריתם הנ"ל הוא בסיבוכיות O(n), כך שבסך הכל קיבלנו סיבוכיות של O(n log n).

ג. האלגוריתם שבסעיף הקודם הוא היעיל ביותר שידוע. לא דרוש נימוק להסבר מדוע לא ייתכן אלגוריתם יעיל יותר (מעבר להיקף הקורס).

ד. במקרה זה אנו זקוקים לפונקציית עזר remove_dups (שמוצגת בדף הבא), לשם הסרת כפילויות ברשימה ממוינת. גם כאן האלגוריתם מתבסס על מיון של שתי הרשימות בשלב הראשון. בשלב השני מתבצעת הסרה של כפילויות משתי הרשימות. במידה ומקבלים רשימות בעלות אורך שונה מחזירים מייד תוצאת False. אחרת מבצעים השוואה של איברים אחד מול אחד. ההמשך ברור.

ה. הסיבוכיות של המיון הכפול היא כמובן O(n log n).
הסיבוכיות של remove_dups היא בבירור O(n). ובחלק האחרון מתבצעות לכל היותר n השוואות, ולכן גם כאן הסיבוכיות הכללית היא O(n log n).

ו. במידה ואנו מוגבלים לשימוש ב-list בלבד, לא נוכל לשפר את האלגוריתם. אם נתיר את השימוש ב-set ו-dict, אז הבעייה הופכת לטריביאלית: set(A) == set(B)
זהו פיתרון בעל סיבוכיות O(n).

```
def zero_indices(A):
    A.sort()
    n = len(A)
    i, j = 0, n-1
    while True:
        if i>j:
            break
        x = A[i] + A[j]
        if x<0:
            i += 1
        elif x>0:
            j -= 1
        else:
            return i,j
    return None
```

```
def lequal(A, B):
    A = remove_dups(A)
    B = remove_dups(B)
    m = len(A)
    n = len(B)
    if not m == n:
        return False
    for i in range(n):
        if not A[i] == B[i]:
            return False
    return True
```

```
def remove_dups(A):
    if A == []:
        return []
    A.sort()
    a = A[0]
    result = [a]
    N = len(A)
    for i in range(1,N):
        b = A[i]
        if b>a:
            result.append(b)
            a = b
    return result
```

פונקציית העזר הדרושה עבור האלגוריתם השני.
הפונקציה remove_dups ממיינת תחילה את הרשימה A, ולכן ניתן להסיר כפילויות בצורה פשוטה יותר מאשר ברשימה לא ממוינת. בכל שלב שומרים את האיבר האחרון ששלפנו במשתנה a, בכל פעם שהאיבר הבא שווה a, מתעלמים ממנו. ברור שנדרשים כאן n פעולות חישוב בכדי להגיע למטרה, ולכן סיבוכיות האלגוריתם הזה היא O(n).

שאלה 4 [12%]

נתון גרף לא-מכוון g אשר קודקודיו הם המספרים [1,2,3,4,5,6,7] וצלעותיו נתונים על ידי טבלת שכנות משמאל.

Vertex	Adjacent Vertices
1	2, 3, 4
2	1, 4, 7
3	1, 4
4	1, 2, 3
5	7
6	7
7	2, 5, 6

א. שרטט את הגרף g באופן ברור במחברת.

ב. בצע אלגוריתם DFS שמתחיל בצומת 1 ובכל שלב שבו האלגוריתם צריך להתקדם לצומת הבא מתוך רשימת צמתים אפשריים, בחר את הצומת הראשון על פי הסדר הטבעי של המספרים. שרטט באופן ברור את העץ הפורש המתקבל כתוצאה מכך.

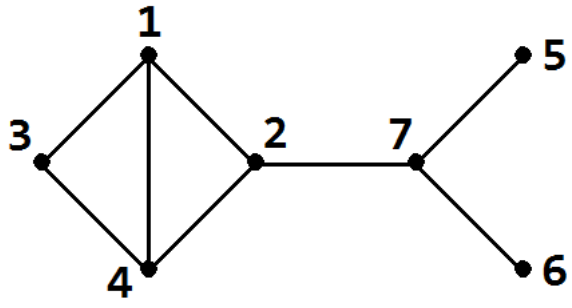
ג. רשום את רשימת הקודקודים על פי סדר הביקור (visit) שהאלגוריתם DFS ביצע (אין צורך להסביר).

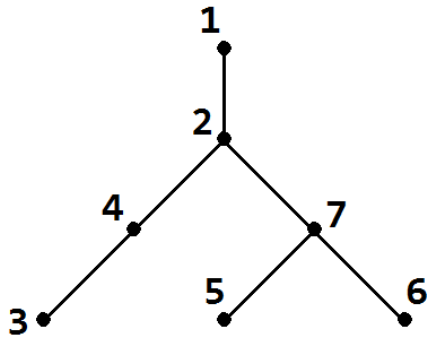
פתרון:

א. קל מאוד לשרטט את הגרף על פי טבלת השכנויות. (איור שמאלי)

ב. בצד ימין רואים את עץ ה-DFS שנוצר על ידי הפעלת האלגוריתם הסטנדרטי של DFS.

ג. הקודקודים נסרקים בסדר הבא: 1, 2, 4, 3, 7, 5, 6





שאלה 5 [15%]

להלן מיפרט ADT (חלקי) של מבנה נתונים מסוג עץ שנקרא SimpleTree בשפת Python.

```
t = SimpleTree()
    Create a new empty SimpleTree object
t.get_root()
    Return the root position of the tree (or None if tree is empty)
t.get_parent(p)
    Return the position of p's parent (or None if x is root)
t.get_children(p)
    Return the position of p's children
t.num_children(p)
    Return the number of children of position p
t.add_root(e)
    Place element e at the root of an empty tree
    Raise ValueError if tree nonempty
t.add_child(p, e)
    add a new element e at the end of the children of p
    Return the position of the new node
```

הנחת עבודה: המחלקה SimpleTree מומשה במלואה (בצרוף מתודות נוספות שאינן רלבנטיות כאן), וסיבוכיות הזמן של כל המתודות היא $O(1)$.

ענה על השאלות הבאות:

א. בנה אלגוריתם יעיל ככל האפשר בשם bignodes(t) בשפת Python המחשב את מספר כל המפתחות בעץ שיש להם יותר משני ילדים. עליך להשתמש במתודות הנ"ל בלבד.

ב. עץ t נקרא אחיד (uniform) אם לכל שני אחים (שני צמתים בעלי אבא משותף) יש אותו מספר צאצאים (descendants). בנה אלגוריתם יעיל ככל האפשר is_uniform(t) בשפת Python המקבל עץ t (מסוג SimpleTree) ומחזיר True אם t אחיד, False אם t אינו אחיד. עליך להשתמש במתודות הנ"ל בלבד.
מהי סיבוכיות הזמן של האלגוריתם שבנית?

<u>הערה:</u> מומלץ להשתמש בפונקציות עזר לשם פישוט הקוד.

<u>פתרון:</u>

```
def bignodes(t):
  root = t.get_root()
  if root is None:
    return 0
  return _bignodes(t, root)

def _bignodes(t, p):
  n = 0
  if t.num_children(p) > 2:
    n = 1
  for c in t.get_children(p):
    n += _bignodes(t,c)
  return n
```

```
def uniform(t):
  root = t.get_root()
  if root is None:
     return True
  return uniform_node(t, root)

def uniform_node(t, p):
  numbers = set()
  for c in t.get_children(p):
    n = num_descendants(t,c)
    numbers.add(n)
    if len(numbers) > 1 or not uniform_node(t,c):
      return False
    return True

def num_descendants(t, p):
  n = 0
  for c in t.get_children(p):
    n += 1 + num_descendants(t,c)
  return n
```

שאלה 6 [15%]

א. רשום אלגוריתם find_cycle3(g) בשפת Python אשר מקבל גרף מכוון g ומחפש מחזור (cycle) באורך 3. אם נמצא מחזור כזה בגרף g, האלגוריתם יחזיר את המחזור הראשון שימצא (כרשימת קודקודים), אחרת האלגוריתם יחזיר None. בכתיבת האלגוריתם תוכל להשתמש בכל המתודות של המחלקה Graph המוצגת בשקפים של הקורס.

ב. נתון גרף מכוון g (directed graph) שבו לכל קודקוד יש לכל היותר 5 קודקודים שכנים (יוצאים ונכנסים). מה תהיה סיבוכיות הזמן (worst case) של האלגוריתם במקרה כזה? התבסס על התכונות של מבנה הנתונים של גרף שבנינו בקורס (עליך להכיר את סיבוכיות הזמן של כל מתודה של גרף).

הערה: מחזור בגרף מכוון היא רשימת קודקודים באורך 2 או יותר [v0, v1, …, vn] שבה עבור כל i<n יוצאת קשת מקודקוד vi לקודקוד vi+1, ובסוף יוצאת קשת מקודקוד vn לקודקוד v0.

פתרון:

א.

```
from graph_utils import *
from dfs import *

# You need to run this file in the graph package that we have distributed
# in the course

def find_cycle3(g):
    for v1 in g.vertices():
        for e1 in g.incident_edges(v1):     # for every outgoing edge from v1
            v2 = e1.opposite(v1)            # take the opposite vertex v2
            for e2 in g.incident_edges(v2): # for every outgoing edge from v2
                v3 = e2.opposite(v2)        # take the opposite vertex v3
                for e3 in g.incident_edges(v3):
                    v4 = e3.opposite(v3)
                    if v4 == v1:            # we have a 3-cycle !
                        return [v1, v2, v3]
    return None
```

ב. מדובר בשלוש לולאות פנימיות, כל אחת בגודל של 5. מכאן שעבור כל קודקוד v בגרף, מספר הפעולות שידרשו בכדי לבדוק אם הוא כלול במחזור באורך 3 הוא 125 פעולות חישוב קבועות. בדיקת כל הקודקודים בגרף אם כך תיקח 125n צעדים. לכן הסיבוכיות במקרה זה היא O(n).