מונחה עצמים, מדעי המחשב

מבחן מועד ב' – 21/2/2021

אליזביט איצקוביץ, בעז בן משה

הנחיות כלליות:

- בבחינה זו 4 שאלות כולן חובה, כל שאלה 25 נקודות.
- משך הבחינה הוא שעתיים וחצי (2.5 שעות, 150 דקות) + 10 דקות להגשה.
- אסור חומר עזר: אסור שימוש כלשהו ברשת, בפרט אסור להשתמש באינטרנט, בספרים או בקוד שנכתב מראש. השימוש במחשב מותר אך ורק לצורך קריאת הבחינה והגשתה למודל.
- הבחינה נערכת על דף: מילוי התשובות יעשה על דפי התשובות בלבד – שפורסם והודפס מראש – חובה לרשום את ת"ז שלכם בראש כל עמוד.
- לאורך כל הבחינה חובה לאפשר צילום סביבת הבחינה ע"י הטלפון.
- בסוף הבחינה יש לצלם את כל 4 התשובות שלכם, להכניס אותם למסמך word (או שווה ערך) לפי הסדר, לשמור את המסמך בשם ת"ז שלכם ולהגיש – קובץ יחיד ל moodle (מי שמעוניין בכך יכול להגיש קובץ בפורמט pdf).
- רק במקרה שיש בעיה להעלות את הקבצים למודל נא לשלוח את הבחינה בפורום הבא; https://docs.google.com/forms/d/e/1FAIpQLSdOBoOkZ4OdVcgqW4xI0xthKf-wI3nGZ-kmmWsSI6EmH2JO5Q/viewform
  דואל חירום - רק למקרה ששום דבר לא עובד אפשר לכתוב לכתובת kcg@g.ariel.ac.il

בבחינה זו 7 עמודים:

- עמוד 1: הנחיות כלליות לבחינה
- עמוד 2: שאלות 1,2,3
- עמודים 3-4 שאלה 4
- עמודים 5-7 נספח קוד (עמוד 5 java, עמודים 6-7 python)

נספח קוד (עמודים 5-7) מצורף לבחינה (פורסם מראש):

a. ממשק GraphInterface ב java שמייצג גרף מכוון ממושקל, וכן ממשק GraphAlgoInterface – שמייצג מספר אלגוריתמים על גרפים מכוונים וממושקלים.
b. דוגמאת קוד של המחלקה Point2D: ב java, ומחלקת בדיקה בסיסית
c. מחלקה אבסטרקטית ב GraphAlgoInterface python שמייצגת מספר אלגוריתמים על גרפים מכוונים וממושקלים.
d. דוגמאת קוד של המחלקה Point2D: ב python, וקובץ בדיקה בסיסי.

בהצלחה!!

שאלה 1 (25 נקודות):

הגדרה : בגרף מכוון ממושקל צלע e(u,v) במשקל w תוגדר "סימטרית" אם ורק אם גם הצלע e'(v,u) קיימת בגרף ומשקלה זהה (w) בדיוק. מומלץ לקרוא את שני הסעיפים לפני עונים.

1.1 (10 נקודות) הניחו שקיימת לכם מחלקה בשם Graph (ב **java**) שמממשת את הממשק GraphInterface. הוסיפו למחלקה Graph שיטה יעילה שמחזירה האם הגרף הוא סימטרי (משמע, האם כל הצלעות בגרף הן סימטריות).

```
public boolean isSymmetric() {...}
```

1.2 (15 נקודות) הוסיפו למחלקה Graph (ב **java**) שיטה יעילה שמחזירה את **קבוצת** כל הצלעות הלא סימטריות בגרף (אם כל הגרף הוא סימטרי מחזירה רשימה ריקה)

```
public ArrayList<edge_data> allNoneSymmetric() {…}
```

הדרכה: אם קיימות שתי צלעות בין 1,2 ובין 2,1 במשקל שונה - יש להחזיר את שתיהן. אם צלע לא קיימת - אין להחזיר אותה.

שאלה 2 (25 נקודות):

שימו לב: בשאלה זו אין צורך לכתוב קוד כלל, אלא רק לתכנן מערכת!

במטלה 2 נדרשתם לתכנן מערכת תוכנה שכוללת לקוח יחיד שמתחבר ל"שרת" יחיד. תוכנת "הלקוח" שמאפשרת להזיז סוכנים ע"ג גרף (מכוון). המערכת נדרשה לקבל את העדכונים "משרת", להציג את מיקום הסוכנים והפוקימונים בממשק גרפי ולהזיז את סוכנים ע"ג הגרף. היכולות של השרת הוגדרו דרך ממשק בשם GameService,

כדי לדמות השהייה (delay) בתקשורת בין הלקוח לשרת, הוחלט להריץ בשרת כל פעולה בתהליך נפרד שבסופו תתווסף לו השהייה של התשובה (בנניח מספר אקראי בין 100-500 אלפיות שנייה, בין עשירית לחצי שנייה).

2.1 (10 נקודות) סטודנט מימש את "צד הלקוח" ע"י מחלקה אחת הכוללת ממשק גרפי המציג את הגרף ועליו את הסוכנים והפוקימנים, המחלקה כללה גם אלגוריתם לתכנון מסלול של הסוכנים ועבודה מול "השרת". הסבירו את הבעייתיות בפתרון המוצע, הציגו פתרון נכון יותר - בדגש על התייחסות לתבנית העיצוב הנדרשת.

2.2 (15 נקודות) הסבירו כיצד ניתן לתכנן "צד לקוח" שיקבל מהשרת (שכולל השהייה בין 100-500 אלפיות שנייה לכל פעולה) יותר מ 10 תשובות בשנייה, תארו מה נדרש לעשות ב"צד לקוח" כדי לוודא שאכן השרת יחזיר ללקוח יותר מ 10 תשובות בשנייה.

שאלה 3 (25 נקודות):

שימו לב: בשאלה זו אין צורך לכתוב קוד כלל, אלא רק לתכנן מערכת!

בשאלה זו נתכנן מערכת תכנון מסלול כדוגמאת Waze או GoogleMaps, כללית הלקוח שמעוניין להגיע למקום מסויים מזין את היעד: המידע עובר לשרת, שמחשב את המסלול הקצר ביותר ע"ג גרף התנועה ושולח חזרה ללקוח את המסלול המומלץ, במהלך הנסיעה נתוני הלקוח עוברים לשרת ומאפשרים לעדכן את גרף התנועה בהתאם למהירות האמיתית (ע"י חישוב מהירות אופיינית של כלל הלקוחות באותו מקום וזמן), כאשר יש שינוי מהותי בגרף התנועה שמשפיע על לקוח מסויים הוא מקבל התראה שזמן ההגעה (או המסלול המומלץ) השתנה.

3.1 (12 נקודות) הציעו שיטה לעדכון משקל הצלעות בגרף על פי דיווחי הלקוחות.

3.2 (13 נקודות) הסבירו כיצד יכול לקוח (מנווט) להתעדכן מהשרת בשינויים בזמן ההגעה הצפוי (או במסלול הרצוי) באופן יעיל.

הדרכה: ציינו את תבנית העיצוב הנדרשת לצורך מימוש המערכת.

שאלה 4 (25 נקודות):

המחלקה: Planet מייצגת כוכב לכת ומקבלת בבנאי את שם הכוכב והמיקום שלו בקואורדינטות x,y,z.

סעיף 4.1 (6 נק'): סטודנט כתב את הMain הבא:

כאשר הוא הריץ אותו הוא קיבל את הפלט שמודגש באדום .

הוסיפו קוד למחלקה Planet כך שבהרצת אותו הMain יתקבל הפלט שמודגש בירוק.

```
class Planet:
    def __init__(self, planet_name: str, pos: Tuple[float, float, float]):
        """
        :param planet_name: the name of the planet
        :param pos: the position of the planet(x,y,z)
        """
        self.name = planet_name
        self.pos = pos
```

```
if __name__ == '__main__':
    earth = Planet("Earth", (2, 3, 0))
    earth2 = Planet("Earth", (2, 3, 0))
    mars = Planet("Mars", (4, 10, -2))
    moon = Planet("Moon", (0.5, 0.5, 0.5))
    # 4.1
    print(earth == earth2)  # False , but Should be True
    print(earth - moon)  # TypeError: unsupported operand type(s) for -:'Planet' and 'Planet' should be (1.5 ,2.5 ,-0.5)
    print(earth)  # <__main__.planet object at 0x000001B2EB3D7508> but should be name:Earth,pos:(2, 3, 0)
```

דוגמא פלט עבור סעיף 4.1 (מודגש באדום - מצב קיים, מודגש בירוק - מצב נדרש).

הדרכה : חיסור של 2 כוכבי לכת יחסר בין הקואורדינטות של הכוכב הראשון (השמאלי) לקואורדינטות של הכוכב השני (הימני) כ tuple של (x,y,z).

המחלקה SolarSystem, הנתונה למטה הינה מחלקה אבסטרקטית שנועדה לקבץ כוכבי לכת לתוך מערכות שמש

```
class SolarSystem:
    def add_planet(self, planet: Planet) -> bool:
        raise NotImplementedError

    def remove_planet(self, planet_name: str) -> bool:
        raise NotImplementedError

    def get_planet(self, planet_name: str) -> Planet:
        raise NotImplementedError

    def closest_planet(self, pos: tuple) -> Planet:
        raise NotImplementedError
```

.

SolarSystem class

```python
if __name__ == '__main__':
    earth = Planet("Earth", (2, 3, 0))
    earth2 = Planet("Earth", (2, 3, 0))
    mars = Planet("Mars", (4, 10, -2))
    moon = Planet("Moon", (0.5, 0.5, 0.5))
    # 4.2
    sol = OurSolarSystem()
    sol.add_planet(earth)
    sol.add_planet(mars)
    sol.add_planet(moon)
    print(sol.get_planet("Earth"))  # name:Earth,pos:(2, 3, 0)
    sol.remove_planet("Earth")
    print(sol.get_planet("Earth"))  # should  None
    # 4.3
    print(sol.closest_planet((0, 0, 0)))  # name:Moon,pos:(0.5, 0.5, 0.5)
```

Main

<u>סעיף 4.2 (9 נק'):</u>

צרו מחלקה בשם OurSolarSystem שיורשת מ SolarSystem וממשו את השיטות הבאות:

- **<u>add_planet</u>**- מקבלת אובייקט של כוכב לכת במידה והוא לא קיים במערכת מוסיפה אותו. השיטה מחזירה True אם כוכב הלכת הוסף, אחרת False. הדרכה: נדרש פתרון יעיל מבחינת זמן ריצה.
- **<u>remove_planet</u>**- מקבלת שם של כוכב לכת ומוחקת אותו במידה והוא קיים. מהמערכת, מחזירה True אם כוכב הלכת נמחק, אחרת False. הדרכה: נדרש פתרון יעיל מבחינת זמן ריצה.
- **<u>get_planet</u>**- מקבלת שם של כוכב לכת ומחזירה אותו (את כל האובייקט) במידה והוא קיים, אחרת מחזירה None. הדרכה: נדרש פתרון יעיל מבחינת סיבוכיות זמן ריצה.

<u>סעיף 4.3 (10 נק'):</u>

הוסיפו למחלקה שייצרתם בסעיף 4.2 את השיטה **<u>closest planet:</u>** שיטה שמקבלת tuple המייצג נקודה בתלת מימד: (x,y,z) ומחזירה את האובייקט מסוג Planet הקרוב ביותר לנקודה.

הדרכה: השיטה צריכה להחזיר את הכוכב הקרוב ביותר לtuple שהתקבל. כאשר המרחק בין שתי נקודות תלת מימדיות P1,P2 מוגדר להיות:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

בהצלחה!!!

נספח: חומר עזר (פורסם מראש):

Java Interface: GraphInterface.java, GraphAlgoInterface.java
Java Classes: Point2D.java, Point2DTest.java,
Python: Point2D.py, TestStringMethod.py, GraphAlgoInterface.py

```java
public class Point2D {
    public static final double EPS = 0.00001;
    public static final Point2D ORIGIN = new Point2D( x: 0, y: 0);
    private double _x,_y;
    public Point2D(double x,double y) {_x=x; _y=y;}
    public Point2D(Point2D p) {this(p.x(), p.y());}
    public double x() {return _x;}
    public double y() {return _y;}
    public Point2D add(Point2D p) {
        Point2D a = new Point2D( x: p.x()+x(), y: p.y()+y());
        return a; }
    public String toString() {return _x+","+_y;}
    public double distance() {return this.distance(ORIGIN);}
    public double distance(Point2D p2) {
        double dx = this.x() - p2.x();
        double dy = this.y() - p2.y();
        double t = (dx*dx+dy*dy);
        return Math.sqrt(t);
    }
    public boolean equals(Object p) {
        if(p==null || !(p instanceof Point2D)) {return false;}
        Point2D p2 = (Point2D)p;
        return ( (_x==p2._x) && (_y==p2._y));
    }
}
```

```java
class Point2DTest {
    @Test
    void testDistance() {
        Point2D p1 = new Point2D( x: 3, y: 4);
        double d = p1.distance(Point2D.ORIGIN);
        double e = Math.abs(d-5);
        assertTrue( condition: e<Point2D.EPS);
    }
    @Timeout(value = 100,  unit = MILLISECONDS)
    @Test
    void add() {
        int size = 10000;
        Point2D p0 = new Point2D(Point2D.ORIGIN);
        Point2D p1 = new Point2D( x: 1, y: 1);
        for(int i=0;i<size;i=i+1) {
            p0 = p0.add(p1); }
        assertEquals(p0.x(), size, Point2D.EPS);
    }
}
```

```java
import java.util.Collection;
/** This interface represents a directional weighted graph. */
public interface GraphInterface {
    /** returns true iff there is a node with key id in this Graph.
     * @return true iff there is a node with key id in this Graph. */
    public boolean hasNode(int key);
    /** returns an edge_data with the data: (src,dest, weight),
     * null if none. */
    public edge_data getEdge(int src, int dest);
    /** adds a new node to the graph with the given node_data. */
    public void addNode(int n);
    /** Connects an edge with weight w between node src-->dest. */
    public void connect(edge_data e);
    /** Connects an edge with weight w between node src-->dest. */
    public void connect(int src, int dest, double w);
    /** This method returns a pointer (shallow copy) for the
     * collection representing all the nodes in the graph.*/
    public Collection<Integer> getV();
    /** Returns a pointer (shallow copy) for the collection of
     * all the edges getting out of the given node.*/
    public Collection<edge_data> getE(int node_id);
    /** Deletes the node (with the given ID) from the graph,
     * and removes all edges which starts or ends at this node.*/
    public void removeNode(int key);
    /** Deletes and return the edge from the graph. */
    public edge_data removeEdge(int src, int dest);
    /** Returns the number of vertices (nodes) in the graph. */
    public int nodeSize();
    /** Returns the number of edges (assume directional graph). */
    public int edgeSize();
    /** Returns the Mode Count: for testing changes in the graph. */
    public int getMC();
}
```

```java
/** This interface represents few Graph Algorithms
 * (on directed weighted graphs).*/
public interface GraphAlgoInterface {
    /** updates the underlying graph on which the
     * algorithms work on. */
    public void init(GraphInterface g);
    /** returns the graph (interface) on which the
     * algorithm works in */
    public GraphInterface getGraph();
    /** returns true iff the underlying graph is
     * strongly connected (as a directed graph).*/
    public boolean isConnected();
    /** returns the distance of the shortest path
     * between src and the dest (-1 if none)/ */
    public double shortestPath(int source, int dest);
}
```

```java
/** This interface represents the set of operations applicable on a
 * directional edge(src,dest) in a (directional) weighted graph. */
public interface edge_data {
    /** The id of the source node of this edge. */
    public int getSrc();
    /** The id of the destination node of this edge. */
    public int getDest();
    /** @return the weight of this edge (positive value).*/
    public double getWeight();
}
```

Python

```python
class Point2D:  # To extend a class put it inside the brackets

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def shift(self, x: float, y: float):
        self.x += x
        self.y += y

    def __eq__(self, other):
        if isinstance(other, Point2D):
            return self.x == other.x and self.y == other.y
        return False

    def __repr__(self):
        return f"Point 2D ({self.x},{self.y})"
```

```python
class GraphAlgoInterface:
    """This abstract class represents an interface of a graph."""
    def get_graph(self) -> GraphInterface:
        """   :return: the directed graph on which the algorithm works on."""
    def shortest_path(self, id1: int, id2: int) -> (float, list):
        """ Returns the shortest path (id1 -->id2) as a dist,list of nodes """
        raise NotImplementedError
    def is_component(self) -> bool:
        """ @return: iff the graph os strongly connected """
        raise NotImplementedError
```

Python unittest
assertEqual(a, b)# a == b
assertNotEqual(a, b)# a != b
assertTrue(x)# bool(x) is True
assertFalse(x)# bool(x) is False
assertIsNone(x)# x is None
The setUp() and tearDown() methods allow you to define instructions that will be executed before and after each test method.
Example:

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```