מונחה עצמים, מדעי המחשב

מבחן מועד א' – 1/2/2021

אליזביט איצקוביץ, בעז בן משה

הנחיות כלליות:

- בבחינה זו 4 שאלות כולן חובה, כל שאלה 25 נקודות, כל שאלה מחולקת לשני סעיפים.
- 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 בדואל 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 (java עמוד 5, python עמודים 6,7)

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 python GraphAlgoInterface שמייצגת מספר אלגוריתמים על גרפים מכוונים וממושקלים.
d. דוגמאת קוד של המחלקה Point2D: ב python, וקובץ בדיקה בסיסי.

<u>שאלה 1 (25 נקודות):</u>

בשאלה זו נתייחס למנגנון בדיקות יחידה - JUnit Testing ב java:

1.1 (12 נקודות) הסבירו באופן כללי (אין צורך לכתוב קוד) כיצד ניתן לממש את המנגנון של JUnit Testing: מבחינת הרצה של כל השיטות הרלוונטיות (ניתן להניח שקיימת מחלקת בדיקות יחידה) התמודדות עם זריקת שגיאות, בדיקת זמן ריצה מקסימאלי של פונקציה (timeout).

1.2 (13 נקודות) הניחו שקיימת לכם מחלקה בשם Graph.java שמממשת את הממשק GraphInterface (ובעלת בנאי ריק) הוסיפו למחלקה GraphTest.java שיטה שבודקת שניתן לבנות גרף בעל 1000 קודקודים ו 10,000 צלעות אקראיות בזמן של פחות משנייה.

הדרכה: צלע אקראית היא צלע בעלת משקל 1 בדיוק, בין זוג סדור של קודקודים אקראיים (שונים זה מזה) שאין בניהם עדיין צלע (מכוונת). הפונקציה: ;()Math.random מחזירה מספר אקראי בין [0,1)

<u>שאלה 2 (25 נקודות):</u>

בשאלה זו נתייחס לממשק של אלגוריתמים על גרפים, כפי שנתון לכם בממשק GraphAlgoInterface (ב java), הניחו שקיימת לכם המחלקה GraphAlgo (בעלת בנאי ריק), שמממשת את הממשק GraphAlgoInterface. נשים לב שאם נפעיל מספר פעמים שיטה אלגוריתמית של GraphAlgo (על אותו קלט) - היא תחושב כל פעם מחדש - גם אם הגרף לא השתנה.

בשאלה זו עליכם לממש מחלקה בשם GraphAlgoMemory שמממשת את GraphAlgoInterface, המחלקה מאפשרת "לזכור" את תוצאות ההרצאה של השיטה (shortestPath) שהיא חישבה, ואם היא מקבלת בקשה לחשב shortestPath שחושב כבר (על אותו הקלט ואותו גרף) היא תחזיר אותו ללא צורך לחשב את האלגוריתם מחדש, אבל אם הגרף השתנה או הפרמטרים לאלגוריתמים לא חושבו בעבר - היא תחושב מחדש ותחזיר את תוצאת האלגוריתם.

2.1 (10 נקודות) הסבירו באופן כללי כיצד תממשו את המחלקה GraphAlgoMemory. התייחסו גם למקרה הכללי שבו נרצה "לזכור" את התוצאות של מספר שיטות שונות.

2.2 (15 נקודות) ממשו את המחלקה ב java – בסעיף זה התייחסו **<u>אך ורק לשיטה</u>** shotestPath (אין צרוך לתמוך בשיטה isConnected).

<u>שאלה 3 (25 נקודות):</u>

שימו לב: בשאלה זו <u>אין צורך לכתוב קוד כלל</u>, אלא רק לתכנן מחלקות!

הניחו שקיימת מחלקה בשם Server שמממשת ממשק של GraphInterface. <u>אך הקוד מקור של המחלקה אינו נתון לכם</u>, אלא רק קובץ jar (מוצפן - בדומה לשרת של מטלה 2). וידוע שהמחלקה אינה Thread-Safe:

3.1 (12 נקודות) הסבירו כיצד ניתן לכתוב מחלקה חדשה בשם ServerMultiCleints שתממש את הממשק GraphInterface ושתהיה Thread-Safe.

הדרכה: הניחו שקיים למחלקה Server בנאי (מעתיק) שמקבל GraphInterface.

3.2 (13 נקודות) הסבירו כיצד ניתן לכתוב את המחלקה חדשה ServerMultiCleints_100 שתהיה גם היא Thread-Safe וגם תאפשר לכל היותר 100 קריאות בשנייה לכלל השיטות מכלל התהליכים,

הדרכה: כל עוד השרת מקבל עד 100 קריאות בשנייה (לשיטות) "השרת" מפעיל את השיטות ("כרגיל"), אבל על כל קריאה שהיא יותר מ 100 בשנייה השרת צריך לזרוק שגיאה מסוג RuntimeException.

שאלה 4 (25 נקודות):

נתונות שתי המחלקות הבאות ב python (ראו עמודים 3-4):

1) המחלקה: Medic שמייצגת איש צוות רפואי.
2) המחלקה MedicPayroll נועדה לעזור לחשב את השכר של הצוות הרפואי , המורכב מאחיות ורופאים.

וכן קובץ main שמריץ - ומדגים את השימוש הנדרש במחלקות שעליכם לממש בשאלה זו.

מחלקת MedicPayroll נועדה לעזור לחשב את השכר של הצוות הרפואי , המורכב מאחיות ורופאים.

בפרט המחלקה צריכה לתמוך בשיטות הבאות:

- get_all_month_salary – מחזירה את סכום המשכורות של כל עובדי הרפואה עבור החודש הקרוב
- get_most_expensive_medic - מחזירה את איש הרפואה בעל המשכורת הגבוהה ביותר

נתונה הנוסחה לחישוב גובה המשכורת של "עובד רפואי" (רופא או אחות):

Nurse : 8,000 + 800 for every year of experience

Doctor : 12,000 + 1000 for every year of experience

4.1 (15 נקודות)

כתבו את המחלקות Nurse, ו Doctor - שמייצגות אחות ורופא בהתאמה, כך שהקוד ב main (כפי שמוצג בעמוד הבא) יחזיר את התוצאות שרשומות בהערות.

הדרכה:

1. אין לשנות את המחלקה MedicPayroll
2. מותר להוסיף שיטות ל Medic אך אסור לערוך שיטות שכבר קיימות במחלקה Medic (בפרט אסור לשנות את השיטה getSalary).

4.2 (10 נקודות)

כתבו מחלקת בדיקת יחידה (unit test) אשר תבדוק נכונות המחלקה TestMedicPayroll תוך שימוש במחלקות שייצרתם.

הדרכה: ניתן להיעזר בדוגמאות הקוד של מחלקת הבדיקה המצורפת לבחינה.

```python
class Medic:
    def __init__(self, name: str, family_name: str, id: str, experience: int):
        self.experience = experience
        self.name = name
        self.family_name = family_name
        self.id = id

    def get_salary(self):
        raise NotImplementedError

    def __str__(self):
        return f"name:{self.name} , id {self.id}"
```

```python
class MedicPayroll:

    def __init__(self):
        self.medics = {}

    def add_medic(self, medic):
        """
        :param medic: add medic to medics dict
        :return: None
        """
        self.medics[medic.id] = medic

    def get_medic(self, id: str) -> Medic:
        """
        :param id: get the medic with the id specified
        :return: Medic
        """
        return self.medics.get(id)

    def get_all_month_salary(self) -> int:
        """return the sum of salary of all medics"""
        sum = 0
        for med in self.medics.values():
            sum += med.get_salary()
        return sum

    def get_most_expensive_medic(self) -> Medic:
        """
        :return: Max of all the medics
        """
        # In this case The max() function returns
        # the item with the highest value in an iterable
        # by using the '>' operator for comparision.
        return max(self.medics.values())
```

```python
if __name__ == '__main__':
    nurse = Nurse("Noa", "Levi", "1223", experience=2)
    doc = Doctor("Michael", "Wag", "1111", experience=2)
    m = MedicPayroll()
    m.add_medic(nurse)
    m.add_medic(doc)
    print(nurse.get_salary())
    print(doc.get_salary())
    print(m.get_all_month_salary())  # output 23600 (Nurse 9600+ Doctor 14000)
    print(m.get_most_expensive_medic())  # name:Michael , id 1111
```

נספר: 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()
```