מונחה עצמים, מדעי המחשב

חומר עזר לבחינה (מועדים א+ב) + הנחיות

שי אהרן, 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).
- אסור חומר עזר: אסור שימוש כלשהו ברשת, בפרט אסור להשתמש באינטרנט, במחשבים, בספרים או בכל חומר כתוב.
- הבחינה נערכת על דף: מילוי התשובות יעשה על גבי מחברת הבחינה בלבד.
- לבחינה זו מצורפים קטעי קוד, אם לא נאמר בפירוש אחרת, ניתן להשתמש בהם לפתרון כל אחד מסעיפי הבחינה.

בבחינה זו 7 עמודים:

- עמוד 1: הנחיות כלליות לבחינה
- עמוד 2-4: שאלות 1,2,3,4
- עמודים 5-7 נספח קוד (פורסם מראש)

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

- ממשק GraphInterface ב java וב python שמייצג גרף מכוון ממושקל.
  ממשק GraphAlgoInterface ב java וב python שמייצג מספר אלגוריתמים על גרפים מכוונים וממושקלים.
- דוגמאת קוד של המחלקה Point2D: ב java, ומחלקת בדיקה בסיסית.
- דוגמאת קוד של המחלקה Point2D: ב python, וקובץ בדיקה בסיסי.
- דוגמאת קוד בסיסית ב python לשימוש במבני נתונים קיימים.

בהצלחה!!

נספח: חומר עזר (פורסם מראש):

Graph: GraphInterface.java, GraphAlgoInterface.java, GraphInterface.py GraphAlgoInterface.py
Basics: Point2D.java, Point2DTest.java, Point2D.py, TestPoint2D.py, DS.py (lists, sets, dict.s)

```python
class GraphInterface:
    """ Abstract class representing a directed graph."""
    def v_size(self) -> int:
        """ @return: The number of vertices"""
        raise NotImplementedError
    def e_size(self) -> int:
        """ @return: The number of edges"""
        raise NotImplementedError
    def get_all_v(self) -> dict:
        """return a dictionary of all the nodes"""
    def all_in_edges_of_node(self, id1: int) -> dict:
        """ return a dictionary of all the in edges """
    def all_out_edges_of_node(self, id1: int) -> dict:
        """ return a dictionary of all the out edges"""
    def get_mc(self) -> int:
        """ @return: The Mode Counter """
        raise NotImplementedError
    def add_edge(self, id1: int, id2: int, weight: float) -> bool:
        """ @return: True if the edge was added successfully, """
        raise NotImplementedError
    def add_node(self, node_id: int, pos: tuple = None) -> bool:
        """ @return: True if the node was added successfully """
        raise NotImplementedError
    def remove_node(self, node_id: int) -> bool:
        """ Removes a node from the graph."""
        raise NotImplementedError
    def remove_edge(self, node_id1: int, node_id2: int) -> bool:
        """ Removes an edge from the graph."""
        raise NotImplementedError
```

```java
package moed_a;
import java.util.Iterator;
/** This interface represents a directional weighted graph. */
public interface GraphInterface {
    /** @return true iff there is a node.id==key in the Graph. */
    public boolean hasNode(int key);
    /** @return the weight of the edge (src,dest), -1 if none.*/
    public double getEdge(int src, int dest);
    /** Adds a new node to the graph with the given id=n. */
    public void addNode(int n);
    /** Connects an edge with weight w between node src-->dest. */
    public void connect(int src, int dest, double w);
    /** @return an Iterator over all the Nodes (ID). */
    public Iterator<Integer> iterAllV();
    /** @return an Iterator over all the out edges of node id.*/
    public Iterator<Integer> iterOutNodes(int id);
    /** Deletes the node from the graph (and all related edges).*/
    public void removeNode(int key);
    /** Deletes and return the weight(src,dest), -1 is none. */
    public double removeEdge(int src, int dest);
    /** @return the number of vertices (nodes) in the graph. */
    public int nodeSize();
    /** @return the number of edges (assume directional graph). */
    public int edgeSize();
    /** @return the Mode Count: for testing changes in the DS.*/
    public int getMC();
}
```

```python
import GraphInterface

class GraphAlgoInterface:
    """Abstract class representing algorithms on graphs"""
    def get_graph(self) -> GraphInterface:
        """ returns: the underlaying directed graph """
    def load_from_json(self, file_name: str) -> bool:
        """returns: True if the loading was successful"""
        raise NotImplementedError
    def save_to_json(self, file_name: str) -> bool:
        """@return: True if the save was successful, """
        raise NotImplementedError
    def shortest_path(self, id1: int, id2: int)->(float, list):
        """ :returns: shortest path, and distance """
        raise NotImplementedError
    def centerPoint(self) -> (int, float):
        """:returns The nodes id, min-maximum distance """
```

```java
package moed_a;
/** 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();
    /** @return a new and empty (no nodes) graph*/
    public GraphInterface getEmptyGraph();
    /** Saves the graph in JSON format */
    public boolean save(String file);
    /** Creates a graph from a JSON file.*/
    public boolean load(String file);
    /** 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
package moed_a;
/** This class represents a 2D point in the plane. */
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;}
    /** Return the 2D distance from this point to (0,0). */
    public double distance() {return this.distance(ORIGIN);}
    /** @return the 2D (Euclidean) distance between this and p2.
     */
    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);
    }
    /**
     * Check if this point equels to the other (p) point.
     * @param p the other point
     * @return true iff this point equels exactly to (p).
     */
    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
import moed_a.Point2D;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import static java.util.concurrent.TimeUnit.MILLISECONDS;
import static org.junit.jupiter.api.Assertions.*;

public 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: 2);
        for(int i=0;i<size;i=i+1) {
            p0 = p0.add(p1);
        }
        assertEquals(p0.x(), size, Point2D.EPS);
        assertEquals(p0.y(),  actual: 2*size, Point2D.EPS);
        assertNotEquals(p0.y(),  actual: 2.001*size, Point2D.EPS);
    }
}
```

```python
from numpy import sqrt
class Point2D:
  def __init__(self, x: float, y: float):
    self._x = x
    self._y = y
  def add(self, p):
    self._x += p._x
    self._y += p._y
  def __add__(self, p):
    return Point2D(self._x + p._x, self._y + p._y)
  def __sub__(self, p):
    return Point2D(self._x - p._x, self._y - p._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"Point2D ({self._x},{self._y})"
  def dist2(self):
    return self._x*self._x + self._y*self._y
  def dist(self, other):
    return sqrt((self-other).dist2())
  def __lt__(self, other):
    if isinstance(other, Point2D):
      return self.dist2() < other.dist2()
    return False
import unittest
from Point2D import Point2D

class TestPoint2D(unittest.TestCase):
    def setUp(self):
        self._p1 = Point2D(1,2)
        self._p0 = Point2D(0, 0)
        self._p2 = Point2D(4, 6)
    def test_eq(self):
        p1 = Point2D(1,2)
        self.assertEquals(p1, self._p1)
        self.assertNotEquals(self._p1, self._p2)
    def test_dist(self):
        self.assertEquals(self._p1.dist(self._p2),5)
    def test_dist2(self):
        self.assertEquals(self._p0.dist2(), 0)
        self.assertNotEquals(self._p1.dist2(), 0)

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

```python
# motivated by www.geeksforgeeks.org/
List1 = []
for i in range(0, 4):
    List1.append(i)
List1.insert(3, len(List1)*3)
List1.insert(-1, 'G 4 G')
List1.remove(2)
List1.pop(1)
List1.extend([5,6,7])
List1.insert(-2, [8,9,10])
print(List1)
# [0, 12, 'G 4 G', 3, 5, [8, 9, 10], 6, 7]

set1 = set()
for i in range(-2, 3):
    set1.add(i)
set1.add((3, 4))
set1.add(1)
set1.add(1)
set1.remove(1)
print(set1) #{0, 2, (3, 4), -1, -2}

Dict=dict({1:'Geeks',2:'For',3:'Geeks'})
Dict[5] = "five"
Dict.pop(2)
print(Dict)# {1: 'Geeks', 3: 'Geeks', 5: 'five'}
```