בחינה ב"מבוא לתכנות מונחה עצמים" + הצעת פתרון

מספר קורס: 7027910
סמסטר א', מועד א', 10.2.2020
מרצים: בעז בן משה, אליזבת איצקוביץ

משך הבחינה שעתיים וחצי
אין חומר עזר (אסור ספרים, אסור מחשבים, אסור מחברות - או כל חומר אחר)
מבנה הבחינה: 4 שאלות בעלות משקל שווה, יש לענות על כולן.
תשובות מסורבלות או דו משמעיות לא יזכו בניקוד מלא.

מותר בהחלט לענות על סעיף אחד בעזרת סעיף אחר.
לבחינה זו מצורפים החומרים הבאים:

1. ממשק שמייצג גרף מכוון, ממושקל (graph).
2. ממשק שמייצג קודקוד בגרף מכוון (node_data).
3. ממשק שמייצג צלע בגרף מכוון ממושקל (edge_data).

הקפידו על טוהר הבחינה !!

בהצלחה !

שאלה 1 (25 נקודות):

התייחסו לממשק של graph שמייצג גרף מכוון ממושקל (מצורף).

בשאלה זו נתכנן ונממש את המחלקה EdgeIterator שמממשת את הממשק Iterator<edge_data>:

```
public interface Iterator<T> {
	public boolean hasNext();  // return true iff there are still “unvisited” elements of T.
	public T next(); // return the “next” element of type T.
}
```

1.1 (5 נקודות) הסבירו במילים כיצד יש לממש את המחלקה מבחינת שדות מידע, ואופן פעולה.

1.2 (15 נקודות) ממשו באופן פשטני את המחלקה EdgeIterator

1.3 (5 נקודות) הסבירו במילים כיצד ניתן לוודא שמופע של ה EdgeIterator מעודכן לגרף, משמע: אם הגרף משתנה לאחר יצירת האובייקט (של האיטרטור) על אותו אובייקט EdgeIterator לזרוק שגיאה.

תשובה:

1.1 המחלקה EdgeIterator תממש את הממשק:
Iterator<edge_data>, ותכיל את השדות הפרטים הבאים: מערך דינאמי של צלעות ואינדקס.
למחלקה יהיה בנאי שיקבל graph יאתחל את מבנה הנתונים, יעבור על כל הקודקודים בגרף ולכל קודקודי ייקח את רשימת הצלעות שיוצאות ממנו (שיטה קיימת בממשק) ויוסיף אותה למבנה הנתונים. לבסוף יאתחל את האינדקס ל 0. השיטה hasNext פשוט תבדוק האם האינדקס קטן מגודל מבנה הנתונים והשיטה next תחזיר את האיבר במקום האינדקס ותוסיף אחד לאינדקס (או null אם אין איבר נוסף).

1.2

```
/**  * This class represents an Iterator<Edge> over a directed graph.*/
public class EdgeIterator implements Iterator<edge_data>{
	private ArrayList<edge_data> _edges;
	private int _ind;
	public EdgeIterator(graph g) {init(g);}
	@Override
	public boolean hasNext() {return _ind < _edges.size();}
	@Override
	public edge_data next() {
		edge_data ans =  null;
		if(hasNext()) {ans = _edges.get(_ind);_ind++;}
		return ans;
	}
	private void init(graph g) {
		_edges = new ArrayList<edge_data>();
		Iterator<node_data> v_iter = g.getV().iterator();
		while(v_iter.hasNext()) {
			int vid = v_iter.next().getKey();
			_edges.addAll(g.getE(vid));
		}
		_ind = 0;
	}
}
```

1.3 ניתן לשמור במחלקה (איטרטור) קישור לגרף וכן שדה מידע נוסף שיקבל את הערך של ה Mode_count של הגרף – בזמן האתחול, ואז בכל הפעולה של שיטה של האיטרטור נוודא שערך ה mode_count של הגרף לא השתנה מאז האתחול (אם הוא כן השתנה - נזרוק שגיאה).

שאלה 2 (25 נקודות):

התייחסו לממשק של graph, הניחו שקיימת לכם מחלקה בשם DGraph שמממשת את הממשק (שימו לב בשאלה זו אין צורך לכתוב קוד):

2.1 (12 נקודות) הסבירו באופן כללי איזו בעיה יכולה להיגרם אם מספר תהליכים ייגשו לאותו גרף. הדגימו בעיה ספציפית, אילו שיטות יש צורך לשנות כדי לאפשר לגרף לתמוך במספר תהליכים.

2.2 (13 נקודות) הסבירו באופן כללי (אין צורך לכתוב קוד) כיצד ניתן לאפשר לתהליכים שונים לקבל עדכון כאשר הגרף משתנה, ציינו את תבנית העיצוב הרלוונטית לכך.

תשובה:

2.1 אם שני תהליכים ייגשו לאותו גרף (או מבנה נתונים באופן כללי) ייתכן מצב שאינו thread-safe נניח כאשר שני התהליכים נכנסים לשיטות שמשנות את "מצב הגרף" כל אחד מהתהליכים בודק ש"ניתן לשנות" ואז שניהם ישנו את אותו מידע (פעמיים) – בנסיבות מסוימות מצב כזה יכול ליצור שגיאה, לדוגמא שני תהליכים ינסו (כל אחד) להוסיף **אותו** קודקוד, השיטה של הוספת קודקוד תוודא שהוא לא קיים (בשני התהליכים) ואז הוא יתווסף פעמיים. כדי לפתור את הבעיה נוכל לסנכרן (synchronized) את השיטות של graph, למעשה, מספיק לסנכרן רק את השיטות שמשנות את המצב בפנימי של האובייקט.

2.2 כללית אם נרצה ש"מבנה נתונים" יעדכן תהליכים (או אובייקטים באופן כללי יותר) בכך שהוא השתנה, נוכל לעשות זאת ע"י תבנית העיצוב של observer משמע התהליכים ירשמו למבנה הנתונים – ובכל שינוי שיתקיים במבנה הנתונים, כל מי שנרשם ל"שירות העדכונים" יקבל "התראת שינוי". במימוש מעשי ניתן לממש מחלקה נוספת שתאפשר מצד אחד לתהליכים להירשם לשינויים, ומצד השני בכל שינוי של מבנה הנתונים היא תעודכן ע"י מבנה הנתונים ישירות, ואז תעבור על כל רשימת התהליכים שנרשמו לשינויים ותעדכן אותם.

שאלה 3 (25 נקודות):

3.1 (8 נקודות) במטלות 3,4 נדרשתם להזיז רובוטים ע"ג גרף. המערכת נדרשה לעדכן את השרת (ולהתעדכן ממנו) וכן להציג את מיקום הרובוטים במשק גרפי. בעוד שההצגה במשק הגרפי צריכה להיות בקצב אחיד (נניח 10 פעמים בשנייה) העדכונים של "השרת" תלויים במיקום ומהירות של הרובוטים (בקצב משתנה ותלוי אירועים). הסבירו כיצד ניתן לפתור בעיה זו בצורה מערכתית מיטבית.

3.2 (9 נקודות) בקורס הצגנו שתי דרכים שונות לשמור מידע בקובץ: דרך ממשק serializable, ודרך json, הסבירו בקצרה כל אחת מהשיטות ועמדו על היתרונות והחסרונות של כל שיטה.

3.3 הניחו שלשני סטודנטים (Alice, Bob) יש פרויקט משותף ב github, הסבירו מהו Conflict, כיצד הוא נגרם וכיצד ניתן לפתור אותו, הדרכה: השתמשו במושגים הרלוונטיים מעולם ה git.

תשובה:

3.1 את הבעיה ניתן לפתור בעזרת שני תהליכים נפרדים:

- תהליך התצוגה: יציג בקצב קבוע (נניח 10 פעמים בשנייה) את המיקום המעודכן של הרובוטים בחלון הגרפי. מיקום זה יחושב לפי המקום הקודם, המהירות, והזמן שעבר מאז שכל רובוט עודכן מהשרת) – נשים לב שכדי לממש זאת נצטרך לשמור את המידע הרלוונטי בכל רובוט ולעדכן את השיטה שמחזירה את מיקום הרובוט – כך שתאפשר להניע את הרובוט מהמקום הקודם שלו בזמן * המהירות שלו (הערה: נשים לב ששיטה זו למעשה תזיז את הרובוט לאורך צלע יחידה, שכן לאחר הגעה לקודקוד (או לפרי) אנו מצפים שהתהליך השני (הלוגי) יעדכן מחדש את מיקום הרובוט).
- תהליך לוגי: ימפה וימיין את האירועים הקרובים לפי המועד הקרוב ביותר. ובהתאם (ימתין) יפנה לשרת יעדכן את יעדי הרובוטים ויתעדכן ממנו במיקומים המעודכנים (רובוטים + פירות). נשים לב שמועדי האירועים תלויים במיקום, מהירות ויעד של כל רובוט בייחס לגרף ולפירות עליו.

3.2 פורמט json הוא למעשה ייצוג טקסטואלי של מידע בעזרת פורמט של סוגרים מאוזנים בין מפתח לערך (שם לתוכן) . ב json ניתן להציג מחרוזות, ערכי אמת או שקר, מספרים, מערכים או מבנים. הפורמט משמש בעיקר להעברת מידע בין מחשבים (נניח שרת לקוח) והוא קריא ע"י בני אדם (בהיותו טקסטואלי), ונתמך ע"י שפות פיתוח רבות.
serializable הוא מנגנון שמירה והעברה של מידע בינארי ב java. המנגנון מבוסס על היכולת של השפה java למפות אובייקט מהזיכרון לייצוג בינארי שניתן לשמור אותו בקובץ או להעביר אותו בתקשורת. כדי שניתן יהיה לשוב ולבנות מהמידע אובייקט של java המערכת שומרת ייצוג קומפקטי של מבנה כל מחלקה שהאובייקטים שלה מרכיבים את האובייקט הנשמר (כל מחלקה שנרצה לשמור צריכה לממש את הממשק הריק serializable).

ל serializable יתרון בולט בכך שהיא למעשה אינה דורשת לכתוב קוד ופשוט "משטחת" אובייקט (מורכב ככול שיהיה) מהזיכרון לייצוג בינארי סדרתי שניתן לשמירה בקובץ או להעברה בתקשרות, הייצוג של המחלקות מהן נבנה האובייקט מאפשרות לבנות אותו מאוחר יותר (או במחשב אחר). החסרונות של השיטה כוללים את העובדה שמדובר בייצוג שמתאים רק לעבודה ב java, הוא אינו קריא ע"י משתמש אנושי והוא דורש שלא יהיה שינוי במבנה המחלקות של האוביקט שנשמר. ולפיכך היתרונות של ה json כוללים: התאמה לשפות תכנות רבות, ייצוג שאינו תלוי בגרסת קוד קונקרטית, ויכולת של משתמש אנושי לקרוא ולהבין את המידע.

3.3 שתי התשובות הבאות התקבלו – כתשובות מלאות:
קונפליקט: הוא מצב שבו קיימים שני (commits) של אותו פרויקט (repository), ושבשני ה commits נעשו שינויים. נניח שכל אחד מהמשתמשים (Alice & Bob) עשה שינוי אחר באותו קובץ (נניח אפילו באותה שורת קוד), ואז כאשר הם ינסו לאחד בין שני ה commits - git לא ידע כיצד לעשות זאת שכן יש שני שינויים שונים לאותו קובץ. כללית ניתן לפתור בעיה זו במספר דרכים: אחד המשתמשים יכול לקחת את השינויים שעשה השני (pull) ואז לעשות עליהם את השינויים שלו (ואז לאחד בניהם). יש כלים להתמודדות עם conflict שמאפשרים למשתמש להחליט איזה שינוי להכניס לגרסה המאוחדת. וכמובן שבפיתוח נכון עדיף מראש לחלק את האחריות בין מפתחים לרזולוציה של קובץ (מחלקה).

merge conflict הוא מצב שבו יש שני ענפים (branches) של אותו פרויקט (repository), ונניח שבשני הענפים נעשו שיניים, נניח שכל אחד מהמשתמשים (Alice & Bob) עשה שינוי אחר באותו קובץ (נניח אפילו באותה שורת קוד), ואז כאשר הם ינסו לאחד (merge) בין שני הענפים git לא יודע כיצד לעשות זאת שכן יש שני שינויים שונים לאותו קובץ. כללית ניתן לפתור בעיה זו במספר דרכים: יש כלי פתרון ל merge conflict שמאפשרים למשתמש להחליט איזה שינוי להכניס לגרסה המאוחדת. אחד המשתמשים יכול לקחת את השינויים שעשה השני (pull) ואז לעשות עליהם את השינויים שלו (ואז לאחד בניהם). וכמובן שבפיתוח נכון לעדיף מראש לחלק את האחריות בין מפתחים לרזולוציה של קובץ (מחלקה).

<u>שאלה 4 (25 נקודות):</u>
בשאלה זו נתכנן מערכת לניהול מערך מעליות בבניין. הניחו שמדובר במעליות "חכמות" שמחיבות את המשתמש להקיש את קומת היעד (בקומת המקור), ולאחר מכן המשתמש מקבל חיווי לגבי מספר המעלית שתיקח אותו לקומת היעד, ואז המעלית מגיעה לקומת המקור אוספת את המשתמשים ונעה כלפי היעדים (והמקורות) שהמערכת הגדירה לה.
הניחו שבבניין קיימות K מעליות, ו N+1 קומות (בין קומה 0 לקומה N)

4.1 (9 נקודות) אפיינו אילו מחלקות נדרשות כדי לבנות את המערכת: (לכל מחלקה ציינו: שם, תפקיד ופירוט יכולות)

4.2 (8 נקודות) תכננו באופן כללי את האלגוריתם שמשבץ כל בקשה של משתמש למעלית ספציפית. הדרכה: נסו לצמצם למינימום את משך ההמתנה הכללי של משתמשי המעליות.

4.3 (8 נקודות) תכננו מערכת סימולציה שתאפשר בדיקת ביצועים של אלגוריתם שיבוץ מבחינת משך ההמתנה הכללי עבור תרחיש בקשות של משתמשי המעליות בבניין.

תשובה:

4.1 המערכת תכלול את המחלקות הבאות:

- **מעלית**: כוללת מיקום נוכחי, רשימת סדורה של קומות (סדר ביקור בקומות לפי בקשות), הערכה של משך זמן ההגעה לכל בקשה (קומה). יכולת הוספת **בקשה** (מקור ← יעד), (בהגעה לקומת יעד (או לקומת מקור) היא נמחקת מהרשימה באופן אוטומטי).
- **בקשה**: מתארת ב**קשה** ל**מעלית** וכוללת זמן, קומת מקור וקומת יעד.
- **בניין**: מייצג אוסף של **מעליות** במבנה מסוים (עם כמות קומות).
- אלגוריתם: מקבל **בקשות** ל**מעלית** ב**בניין** מסוים ומשבץ אותן ל**מעליות** שונות – בזמן אמת.

4.2 האלגוריתם יבנה ע"י קבלה של בניין (אוסף מעליות). בהינתן בקשה (קומת מקור ← קומת יעד) האלגוריתם יעבור על כל המעליות ויבדוק מי מהן יכולה לשרת את הבקשה בזמן הקצר ביותר (גישה חמדנית). האלגוריתם ישבץ את הבקשה למעלית "הפנויה" ביותר – משמע למעלית שצפויה למלא את הבקשה בזמן הקצר ביותר – ראו פירוט מטה.

לכל מעלית יהיה מיקום נוכחי ואוסף של בקשות ששובצו עבורה (וכן מצב פנימי \ עולה או יורדת). בשלב הראשון המעלית תגיע לבקשה הראשונה ובהתאם למיקום קומת היעד תקבע את מצב הפנימי השלה (עולה או יורדת), ואז תנועה על לקומת היעד כאשר היא עוצרת (בדרך) רק לבקשות נוספות באותו כיוון (ובקשות אחרות ימתינו להיפוך המצב שלה) לאחר ההגעה ליעד הבקשה תוסר מהרשימה. הערכת משך מילוי הבקשה של מעלית יתבסס על המיקום הנוכחי שלה והבקשות הקיימות, והמצב שלה.

4.3 מערכת הסימולציה תכלול מנגנון לבניית **בניין** (כמות קומות כמות **מעליות**) ומערכת שתאפשר לחולל **בקשות** לפי זמן ולהזין אותן לאלגוריתם. נוכל להוסיף למערכת מחלקה שמייצגת "**משתמש**" שעובד בקומה מסוימת, מגיע בבוקר לעבודה (**בקשה** מקומה 0 לקומת העבודה שלו) יוצא בצהריים לאכול חוזר ובערב הולך הביתה. ע"י יצירה של מגוון **משתמשים** שונים נוכל לדמות עומסים שונים על המערכת ואז לחשב את משך זמן ההמתנה האופייני (ממוצע מקרה גרוע וכו') לבקשה כתלות באלגוריתם, בכמות הבקשות והשעה ביום.

בהצלחה!

חומר עזר מצורף:

```
/** This interface represents a directional weighted graph. */
public interface graph {
     /**return the node_data by the node_id (null - if none)*/
     public node_data getNode(int key);
     /** return the data of edge(src,dest), null if none */
     public edge_data getEdge(int src, int dest);
     /** add a new node to the graph with the node_data. */
     public void addNode(node_data n);
/** Connects an edge with weight w between src->dest. */
     public void connect(int src, int dest, double w);
     /** Return a reference (shallow copy) for the
      * collection representing all the nodes in the graph. */
     public Collection<node_data> getV();
     /** Return a reference (shallow copy) for the
      * collection representing all the edges getting out of
```

```
     * the given node (src) */
    public Collection<edge_data> getE(int src);
    /** Delete the node (with the given ID) from the graph */
    public node_data removeNode(int key);
    /** Delete the edge from the graph */
    public edge_data removeEdge(int src, int dest);
    /** return the number of nodes in the graph. */
    public int nodeSize();
     * return the number of edges (directional graph). */
    public int edgeSize();
/** return the Mode Count for testing changes in the graph. */
    public int getMC();
}

/** This interface represents the set of operations applicable
on a node (vertex) in a (directional) weighted graph. */
public interface node_data {
    /** Return the key associated with this node. */
    public int getKey();
}

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