מבוא למדעי המחשב, מועד ב', פתרונות והערות

סמסטר א', תשפ"ב

מאחר ולא ניתן לקיים הליך של "ערעור על ערעור" עליכם לוודא היטב כי הערעור שאתם מגישים מפורט ומנומק היטב. לא מספיק לכתוב כי הפתרון שהוגש נכון ולבקש בדיקה חוזרת. יש לכתוב במפורט מדוע הפתרון שהוגש נכון או מדוע חלק ממנו שהורד עליו נקודות נכון. ערעור לא מפורט/מנומק היטב ידחה על הסף לא בדיקה. המלצתנו היא לצרף קוד שמדגים את נכונות הפתרון שהוגש, במידת הניתן. נא קראו בעיון את ההנחיות להגשת ערעור המפורסמות ביחידת ההוראה במודל.

שאלה 1

סעיף א'

פתרון לדוגמא

```
public BinaryExprTree(ExprToken[] exp) {
    Stack<BinaryExprNode> st = new Stack<BinaryExprNode>();
    for (int i = 0; i < exp.length; i = i + 1) {
        if (exp[i] instanceof NumberToken)
            st.push(new BinaryExprNode((NumberToken) exp[i]));
        else {
            BinaryExprNode right = st.pop();
            BinaryExprNode left = st.pop();
            st.push(new BinaryExprNode(
                    (OperatorToken) exp[i], left, right));
        }
    }
    root = st.pop();
}
```

הערות

1. שימו לב שיש חשיבות רבה לטיפוסים של המשתנים השונים בשאלה, המחסנית מקבלת רק טיפוס מסוג קודקוד והבנאים גם הם מקבלים טיפוסים שונים מסוג OperatorToken ו - NumberToken בלבד.
2. יש צורך להתייחס לעובדה שהערכים של המערך exp עם reference מטיפוס ExprToken אבל עם instance של OperatorToken או NumberToken, לכן עדיין יש צורך בהמרה לטיפוס המתאים בעת קריאה לבנאים השונים.
3. מימושים שניסו להכניס את כל הערכים אל תוך המחסנית אחד אחרי השני ואז לבנות את העץ נפסלו, מכיוון שרובם התעלמו מהעקרונות שצוינו לעיל והשתמשו בשיטות שאינן מתאימות כגון insert.

סעיף ב'

פתרון לדוגמא

בעץ:

```
public String toString() {
    return root.toString();
}
```

בצומת:

```
public String toString() {
    String output;
    if (data instanceof OperatorToken) {
        output = "(" + left.toString();
        output = output + data.toString();
        output = output + right.toString() + ")";
    } else
        output = data.toString();
    return output;
}
```

הערות

1. יש להקפיד על בדיקת null לפני ניגשים לבן שמאלי / ימני.
2. יש להקפיד על שרשור באופן נכון, כך שהסוגריים יוצבו במקומם.
3. בסעיף זה לא נדרש והתבקשתם שלא להוסיף סוגריים מיותרות. למשל, על מספרים.

שאלה 2

פתרון לדוגמא

```
import java.util.NoSuchElementException;

public class StackAsQueue<T> implements Stack<T> {

    private Queue<T> q;
    private int size;

    public StackAsQueue() {
        q = new MysteryQueue<T>();
        size = 0;
    }

    public boolean isEmpty() {
        return q.isEmpty();
    }

    public T peek() {
        if (isEmpty())
            throw new NoSuchElementException();
        return q.peek();

    }

    public T pop() {
        if (isEmpty())
            throw new NoSuchElementException();
        size = size - 1;
        return q.dequeue();
    }

    public void push(T element) {
        q.enqueue(element);
        for (int i = 0; i < size; i = i + 1) {
            q.enqueue(q.dequeue());
        }
        size = size + 1;
    }
}
```

הערות

כללי

1. שימו לב כי היה נוח יותר להניח כי ראש המחסנית נמצא בראש התור q. הנחה זו מקלה מאוד על מימוש השיטות peek ו- pop. יחד עם זאת, גם מימוש שהניח כי ראש המחסנית נמצא בזנב התור, התקבל.
2. שימו לב כי הטיפוס של המשתנה הנוסף הוא int. לא ניתן להניח כי הוא יכול להצביע על איברים מסוג T המאוחסנים במחסנית.
3. שימוש במבני נתונים כגון מערך/מחסנית/תור (אחר מ-q) גרמו לפסילת הסעיף הרלוונטי בהתאם להנחיות בשאלה האוסרות על כך.

סעיף 2.3

1. פתרונות שביצעו pop ו- push לצורך מימוש peek התקבלו. שימו לב שזה לא מימוש יעיל.

הערות משותפת למימוש POP+PUSH בסעיפים 2.4+2.5

1. שימוש ברקורסיה לצורך שמירת האיברים במחסנית הקריאות גרמה להיפוך האיברים השמורים בתור. לכן מימוש כזה הוא שגוי. העובדה שניתן לכם שדה נוסף מטיפוס int מרמזת על הכיוון הנכון של הפתרון, בו מבצעים לולאה בודדת כדי לגשת לזנב התור. לא בוצעה הפחתת נקודות על אי השימוש בשדה.
2. היו פתרונות רקורסיביים שהגדירו את המשתנה הנוסף כמציין של מספר האיברים במחסנית. בחלק מפתרונות אלו, המשתנה שמחזיק את מספר האיברים במחסנית קיבל ערך שגוי עקב עדכונים כפולים או שגויים.

שאלה 3

סעיף א'

פתרון לדוגמא

```
private static List<Integer>[] split(List<Integer> elements) {
    List<Integer>[] answer = new List[2];
    answer[0] = new LinkedList<>();
    answer[1] = new LinkedList<>();
    Iterator<Integer> it = elements.iterator();
    int i = 0;
    while (it.hasNext()) {
        if (i % 2 == 0)
            answer[0].add(it.next());
        else
            answer[1].add(it.next());
        i = i + 1;
    }
    return answer;
}
```

סעיף ב'

פתרון לדוגמא

```
private static List<Integer> merge(List<Integer> list1, List<Integer>
list2) {
    List<Integer> answer = new LinkedList<Integer>();
    Iterator<Integer> it1 = list1.iterator();
    Iterator<Integer> it2 = list2.iterator();
    Integer next1 = null;
    Integer next2 = null;
    if (it1.hasNext()) next1 = it1.next();
    if (it2.hasNext()) next2 = it2.next();

    while (next1 != null | next2 != null) {
        boolean inc1 = false;
        if (next1 == null) {
            answer.add(next2);
        } else if (next2 == null) {
            answer.add(next1);
            inc1 = true;
        } else if (next1.compareTo(next2) < 0) {
            answer.add(next1);
            inc1 = true;
        } else {
            answer.add(next2);
        }
        if (inc1) {
            if (it1.hasNext()) next1 = it1.next();
            else next1 = null;
        } else {
            if (it2.hasNext()) next2 = it2.next();
            else next2 = null;
        }
```

```
    }

    return answer;
}
```

סעיף ג'

פתרון לדוגמא

```
public static List<Integer> sort(List<Integer> elements) {
    List<Integer> answer;
    if (elements.size() == 0) {
        answer = new LinkedList<>();
    } else if (elements.size() == 1) {
        answer = new LinkedList<>();
        answer.add(elements.get(0));
    } else {
        List[] split = split(elements);
        answer = merge(sort(split[0]), sort(split[1]));
    }
    return answer;
}
```

הערות

1. נדרש לממש אלגוריתם מיון המבוסס על מיזוג. פתרונות שלא התבססו על מיזוג לא קיבלו נקודות.

שאלה 4

סעיף א'

פתרון לדוגמא

```
public static boolean isGradual(int[] array, int sum) {
    boolean ans = array[0] == 0;
    for (int i = 1; i < array.length & ans; i = i + 1) {
        int diff = array[i] - array[i - 1];
        boolean abs = diff <= 1 & diff >= -1;
        ans = ans & abs;
        sum = sum - array[i];
    }
    return ans & sum == 0;
}
```

הערות

1. המערך array הינו מטיפוס int ולכן יש לבדוק שההפרש בין כל שני איברים הוא: 0,1 או -1.

סעיף ב'

פתרון לדוגמא

```
public static boolean existGradual(int n, int sum) {
    return existGradual(n, sum, 0);
}

//  Solution A:
public static boolean existGradual(int n, int sum, int num) {
    if (n == 1)
        return sum == 0;
    return existGradual(n - 1, sum - num, num) ||
                existGradual(n - 1, sum - (num - 1), num - 1) ||
                existGradual(n - 1, sum - (num + 1), num + 1);

}

//  Solution B:
public static boolean existGradual(int n, int sum, int num) {
    if (n == 1)
        return (sum - num) == 0;
    return existGradual(n - 1, sum - num, num) ||
                existGradual(n - 1, sum - num, num - 1) ||
                existGradual(n - 1, sum - num, num + 1);
}
```

הערות

1. בהגדרת הסעיף נתון כי n הוא חיובי ולכן מקרה הבסיס צריך להבדק עבור n==1.
2. פתרונות מסויימים בהם מקרה הבסיס בודק n==0 & sum==0, מבצעות פי 3 קריאות רקורסיביות אך עדיין מחזירות פתרון נכון.

סעיף ג'

פתרון לדוגמא

```
public static void printGradual(int n, int sum) {
    printGradual(n, sum, 0, "0");
}

//  Solution A:
public static void printGradual(int n, int sum, int num, String acc) {
    if (n == 1 & sum == 0)
        System.out.println(acc);

    if (n > 1) {
        printGradual(n - 1, sum - num, num, acc + " " + num);
        printGradual(n - 1, sum - (num - 1), num - 1,
                                                acc + " " + (num - 1));
        printGradual(n - 1, sum - (num + 1), num + 1,
                                                acc + " " + (num + 1));
    }
}

//  Solution B:
public static void printGradual(int n, int sum, int num, String acc) {
    if (n == 1 & (sum - num) == 0)
        System.out.println(acc);
    if (n > 1) {
        printGradual(n - 1, sum - num, num, acc + " " + num);
        printGradual(n - 1, sum - num, num - 1, acc + " " + (num - 1));
        printGradual(n - 1, sum - num, num + 1, acc + " " + (num + 1));
    }
}
```

הערות

1. פתרונות מסויימים בהם מקרה הבסיס בודק n==0 & sum==0, מבצעות פי 3 קריאות רקורסיביות ולכן מדפיסות כל מחרוזת 3 פעמים בניגוד לנדרש בסעיף.