בית ספר אפי ארזי למדעי המחשב
המרכז הבינתחומי הרצליה

מבוא למדעי המחשב

פתרון מבחן גמר, מועד ב'

2009

הנחיות:

- זמן המבחן: שלוש שעות.
- זמן הבחינה מוגבל, ויש לעבוד ביעילות. אם נתקעים בסעיף מסוים, כדאי לעזוב אותו ולרוץ הלאה.
- חומר סגור. השימוש במחשבים או במחשבונים אסור.
- ענו על כל השאלות על טופס המבחן. מומלץ להשתמש במחברות הבחינה כטיוטא. ניתן גם לכתוב תשובות במחברת הבחינה ולהפנות אליהן בצורה ברורה ומפורשת מטופס המבחן. חובה למחוק באופן ברור כל דבר שאינכם רוצים שנבדוק.
- בגמר הבחינה יש להחזיר את טופס המבחן + מחברת המבחן + כל דפי העזר הנלווים.
- אם תרגישו צורך לעשות הנחה מסוימת כדי לענות על כל שאלה שהיא, ניתן לעשות זאת, כל עוד ההנחה היא סבירה ומנוסחת בדיוק ובבהירות.
- אם אתם לא מסוגלים לתת תשובה מלאה לשאלה מסוימת, תנו תשובה חלקית. תשובה נכונה באופן חלקי תקבל ניקוד חלקי.
- התשובות חייבות להיות קצרות ולעניין. כתב היד חייב להיות קריא וברור. תשובות לא קריאות יקבלו ציון 0.
- אם התבקשתם לכתוב תכנית שאמורה לפעול על קלט מסוים, אזי התכנית לא אמורה לבדוק אם הקלט תקין, אלא אם כן נאמר כך בשאלה במפורש.
- התכניות שתכתבו תישפטנה, בין היתר, לפי האורך והאלגנטיות שלהן. תוכניות ארוכות או מסורבלות ללא צורך יקבלו פחות נקודות, אפילו אם הן ממלאות את המשימה שהוגדרה בשאלה.
- כל החומרים להם תזדקקו במהלך המבחן מתועדים בדפי העזר שקיבלת.
- לא יורדו נקודות על טעויות סינטקס טריוויאליות.

בהצלחה!

חלק א' (35 נקודות)

צפרדע עומדת לפני טור של n נורות דולקות. הנורות ממוספרות 2 עד n. הצפרדע ניגשת לנורה 2 ומתחילה לקפוץ על הנורות בצעדים של 2. הנורה הראשונה שהיא נוחתת עליה היא נורה 4; הנחיתה הבאה היא נורה 6, אח"כ נורה 8, וכן הלאה. כשהצפרדע גומרת לקפוץ קפיצות בגודל 2, היא חוזרת לנורה 3 ומתחילה לקפוץ קפיצות בגודל 3: הנורה הראשונה שהיא נוחתת עליה היא נורה 6, הנחיתות הבאות הן נורות 9, 12, וכן הלאה. אחר כך הצפרדע חוזרת לנורה 4 ומתחילה לקפוץ קפיצות בגודל 4, וכן הלאה. בכל פעם שהצפרדע נוחתת על נורה כלשהי, הנורה נכבית. נורות הבסיס מהן הצפרדע מתחילה לקפוץ (למשל, נורות 2 ו- 3) נשארות דלוקות, אלא אם כן קפיצות קודמות כיבו אותן (כמו למשל נורה 4). אם קפיצה כלשהי לוקחת את הצפרדע מעבר לנורה האחרונה (שמספרה n), הקפיצה לא מתבצעת.

בסוף התהליך, חלק מהנורות יהיו כבויות וחלק יישארו דלוקות.

מחלקות Bulbs ו- BulbsDemo (ראה דף עזר) מדמות את טור הנורות ואת קפיצות הצפרדע (נורה באנגלית זה bulb). קיראו את תיאור המחלקות ואת התיעוד שלהן.

1. (4 נקודות) בהינתן n כלשהו, איזה תכונה מתמטית מייצגות הנורות שנשארות דלוקות בסוף התהליך?

תשובה: מספרי הנורות הדלוקות הם המספרים הראשוניים מ 2 ועד n.

2. (15 נקודות) כיתבו את הקוד של המחלקה Bulbs (שלוש המתודות). שימו לב: אין צורך לחשוב על שיקולי יעילות, אלא לכתוב את המימוש שמתאר בצורה הישירה ביותר את התהליך שמתואר למעלה.

```
public class Bulbs {
  private Boolean[] b;

  public Bulbs (int n) {
    b = new Boolean[n];
    for (int i = 2; i < n; i++) b[i] = true;
  }

  public void jump () {
    for (int i = 2; i < b.length; i++)
      for (int j = i+i; j < b.length; j=j+i)
        b[j] = false;
  }

  public void printIndex () {
    for (int i = 2; i < b.length; i++)
      if (b[i])
        System.out.print(i + " ");
  }
}
```

3. (10 נקודות) הנח שיש לנו צפרדע עצלנית שמעוניינת להגיע לאותה תוצאה בדיוק בסיום התהליך תוך כדי מספר קפיצות קטן ככל האפשר. הצע שני שיפורים שונים שיגרמו להקטנה משמעותית של מספר הקפיצות שהצפרדע צריכה לעשות, ונמק למה הם ישפרו את המצב.

תשובה: שיפור אחד: אפשר לעצור את תהליך הקפיצות כשמגיעים לנורה מספר sqrt(n). נימוק: עבור כל שלושה מספרים טבעיים m, a, b עבורם מתקיים m = a * b, אם a < sqrt(m) אזי b >= sqrt(m). לכן, כשקופצים מ a בצעדים שגודלם a, אחת הקפיצות תכבה גם את b. שיפור שני: אם מתכוננים להתחיל לקפוץ מבסיס שהוא נורה מכובה, אין צורך לקפוץ ממנה והלאה. נימוק: אם הנורה a מכובה, הרי תהליך הקפיצות שכיבה את a כיבה כבר גם את כל הכפולות של a.

4. (6 נקודות) ממש את שני השיפורים שהצעת ע"י כתיבה מחדש של מתודת jump.

```
public void jump () {
    for (int i = 2; i < Math.sqrt(b.length); i++)
      if (b[i])
        for (int j = i+i; j < b.length; j=j+i)
          b[j] = false;
  }
```

<u>חלק ב'</u> (15 נקודות)

<u>ענו על אחת משאלות 5 או 6.</u>

5. נתונה נוסחה לקירוב הערך של π: $\pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - 4/15 + \ldots$

דרושה מתודה סטטית בשם pi שמקרבת את π בעזרת הנוסחה הנ"ל. המתודה מקבלת כפרמטר מספר שלם חיובי n שמייצג את מספר הגורמים שצריך להשתמש בהם כדי לחשב את π, ומחזירה double שמייצג את הערך המקורב (בדוגמא למעלה n=8).

```
static double pi (int n) {
  double sum = 0;
  boolean plus = true;
  for (int k = 0; k < n; k++) {
    double x = (double) 1 / (2*k+1);
    sum = sum + (plus ? +x : -x);
    plus = !plus;
  }
  return 4 * sum;
}
```

עוד פתרון אפשרי:

```
static double pi (int n) {
  double sum = 0;
  boolean plus = true;
  for (int k = 0; k < n; k++) {
    sum = sum + Math.pow(-1,k) / (2*k+1);
  }
  return 4 * sum;
}
```

6. כשמפקידים כסף בבנק מרוויחים עליו ריבית. למשל, נניח שהפקדנו 1000 שקלים בפיקדון שמציע ריבית שנתית של 5%. בסוף השנה הראשונה יהיה לנו 1050 שקלים. בסוף השנה השנייה יהיו לנו 1102.5 שקלים (שהם 1050 ועוד 5% מ 1050), וכן הלאה, עד שנוציא את הכסף מהבנק.

עליכם לממש מתודה בשם deposit המקבלת שלושה פרמטרים: double המייצג את גובה הפיקדון הראשוני בשקלים (למשל, 1000), double המייצג ריבית תקופתית קבועה (למשל, 0.05), ו- int המייצג את מספר התקופות שהכסף נשאר בבנק (למשל, 12). המתודה מחזירה את ערך הפיקדון בסוף תקופת הפיקדון. כיתבו מימוש רקורסיבי של המתודה הזאת.

```
public static double deposit(double sum, double rate, int n) {
  if (n==0) {
    return sum;
  else
    return deposit(sum, rate, n-1) * (1 + rate);
 }
```

חלק ג' (50 נקודות)

אחד הכלים האפשריים לפירוק String נתון ל "מילים" (tokens) שמופרדות זו מזו ע"י רווח אחד או יותר הוא מחלקת StringTokenizer. כשמאתחלים אובייקט מסוג StringTokenizer, מעבירים לו כפרמטר את הטקסט עליו צריך לפעול. מכאן ואילך, כשמפעילים על האובייקט את המתודה הבוליאנית hasMoreTokens(), המתודה מחזירה true אם יש עוד מילה אחת או יותר בטקסט ו- false אחרת. כשמפעילים על האובייקט את המתודה nextToken(), המתודה מחזירה String שמכיל את המילה הבאה בטקסט. ה API של מחלקת StringTokenizer מתואר בדף עזר, ואתם מתבקשים לעיין בו עכשיו.

דרושה מחלקה בשם CensoredStringTokenizer שפועלת באופן הבא. כשמאתחלים אובייקט מסוג CensoredStringTokenizer, מעבירים לו שני פרמטרים: (א) הטקסט עליו צריך לפעול, ו- (ב) מערך של מילים אותן צריך לצנזר מהטקסט. מכאן ואילך, כשמבקשים להחזיר את המילה הבאה מהטקסט, מקבלים את המילה הלא מצונזרת הבאה. בשלב זה אתם מתבקשים לקרוא את דף העזר שמתאר מימוש חלקי של מחלקת CensoredStringTokenizer ודוגמת שימוש בה.

בשאלה זאת אתם נדרשים לממש את מחלקת CensoredStringTokenizer לפי המימוש החלקי המתואר בדף העזר. חשוב להדגיש שזאת רק דרך אחת לממש את המחלקה הזאת, אך זאת בדיוק הדרך שבה אתם נדרשים לממש אותה.

7. (9 נקודות) כתוב מימוש של הקונסטקרקטור CensoredStringTokenizer.

```
public CensoredStringTokenizer(String text, String[] censoredWords) {
  this.censoredWords = new String[censoredWords.length];
  System.arraycopy(censoredWords, 0,
                   this.censoredWords, 0, censoredWords.length);
  tokenizer = new StringTokenizer(text);
  consume();
}
```

8. (9 נקודות) כתוב מימוש של המתודה consume.

```
// Consume all next tokens that should be censored.
private void consume() {
  while (tokenizer.hasMoreTokens()) {
    nextUncensoredToken = tokenizer.nextToken();
    if (!isCensored(nextUncensoredToken)) {
      return;
    }
  }
  nextUncensoredToken = null;
}
```

9-א. (9 נקודות) כתוב מימוש של המתודה nextToken.

```
public String nextToken() {
  if (nextUncensoredToken == null) {
    throw new NoSuchElementException();
  }
  String token = nextUncensoredToken;
  consume();
  return token;
}
```

9-ב. (3 נקודות) כתוב מימוש של המתודה hasMoreTokens.

```
public boolean hasMoreTokens() {
  return nextUncensoredToken != null;
}
```

9-ג. (5 נקודות) כתוב מימוש של המתודה isCensored.

```
public boolean isCensored(String word) {
  for (int i=0; i<censoredWords.length; i++) {
    if (word.equals(censoredWords[i])) {
      return true;
    }
  }
  return false;
}
```

10. (3 נקודות) למה צריך את הפקודה `import java.util.StringTokenizer` (הפקודה הראשונה במימוש המוצע)?

תשובה: המימוש עושה שימוש במחלקת StringTokenizer. פקודת import מאפשר גישה לשירותים (מתודות) של המחלקה הזאת.

11. (3 נקודות) למה המתודה consume מוגדרת `private`?

תשובה: כי היא מספקת שירות פנימי למתודות אחרות של המחלקה, ואין שום צורך לחשוף אותה לשאר העולם.

12. (3 נקודות) כפי שציינו למעלה, דף העזר מתאר אסטרטגיית מימוש מסוימת של מחלקת CensoredStringTokenizer. תאר באופן כללי אסטרטגיית מימוש אלטרנטיבית שעושה שימוש בעיקרון חשוב של תכנות מונחה עצמים, ונמק יתרון אחד של המימוש האלטרנטיבי שהצעת על פני המימוש שמתואר בדף העזר.

תשובה: אפשר לממש את מחלקת CensoredStringTokenizer כמחלקה שיורשת ממחלקת StringTokenizer. המימוש הזה יהיה יותר ברור וטבעי למי שמכיר כבר את מחלקת StringTokenizer.

13. (3 נקודות) ה API של מחלקת StringTokenizer מתחיל בשלושת השורות הבאות:

```
public class StringTokenizer
extends Object
implements Enumeration
```

הסבר בדיוק נמרץ מה משמעות השורה השנייה (extends Object)
והשורה השלישית (implements Enumeration)

תשובה: כמו כל המחלקות בשפת Java, המחלקה StringTokenizer יורשת ממחלקת Object. כמו כן, היא מממשת את הממשק (interface) ששמו Enumeration. במילים אחרות, היא מספקת מימוש לכל המתודות שמתועדות בממשק הזה.

14. (3 נקודות) ה API של מחלקת StringTokenizer מכיל שתי חתימות מתודות תחת אותו שם: nextToken. הסבר למה ה API מעוצב כך, ומה שם המנגנון הכללי בתכנות מונחה-עצמים שמאפשר את העיצוב הזה.

תשובה: שתי החתימות מאפשרות לקבל את המילה הבאה מהטקסט בשני אופנים שונים. באופן הראשון, לפי מפריד (delimiter) שהוא ברירת מחדל, מן הסתם רווח, ובאופן השני לפי מפריד שניתן לציינו כפרמטר. בגלל ששתי המתודות מספקות את אותו שירות כללי, רצוי לקרוא להן באותו שם. המנגנון הכללי שמאפשר זאת נקרא method overloading.

*** סוף המבחן ***

דף עזר: מחלקת Bulbs (מבנה המחלקה ודוגמת שימוש בה)

```
// Represents an array of bulbs, each being on or off.
public class Bulbs {
  private Boolean[] b;  // Array of bulbs, each being on or off.

  // Constructs an array of n bulbs and turns them all on.
  public Bulbs (int n) {
  }

  // Jumps on the bulbs in steps, starting from bulb number 2,
  // as described in the exam.
  public void jump () {
  }

  // Prints the numbers of all the bulbs that are on ("dlukot").
  public void printIndex () {
  }
}


// Example of using the Bulbs class

public class BulbsDemo {
  public static void main (String[] args) {
    Bulbs bulbs = new Bulbs(100);
    bulbs.jump();
    bulbs.printIndex();
  }
}
```

דף עזר: StringTokenizer API

מחלקת StringTokenizer היא חלק מספריית המחלקות של שפת Java. ה API שלה הוא כדלקמן:

```
public class StringTokenizer
extends Object
implements Enumeration<Object>
```

The string tokenizer class allows an application to break a string into tokens. A `StringTokenizer` object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed. A token is returned by taking a substring of the string that was used to create the `StringTokenizer` object. The following is one example of the use of the tokenizer. The code:

```
StringTokenizer st = new StringTokenizer("this is a test");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

Prints the following output:

```
this
is
a
test
```

Constructor Summary

StringTokenizer(String str) Constructs a string tokenizer for the specified string.

Method Summary

int	**countTokens**() Calculates the number of times that this tokenizer's `nextToken` method can be called before it generates an exception.
boolean	**hasMoreTokens**() Tests if there are more tokens available from this tokenizer's string.
Object	**nextElement**() Returns the same value as the `nextToken` method, except that its declared return value is `Object` rather than `String`.
String	**nextToken**() Returns the next token from this string tokenizer.
String	**nextToken**(String delim) Returns the next token from this string tokenizer using delim as delimiter.

הנחת הבסיס היא שה tokens מופרדים זה מזה ברווח (space character) אחד או יותר. שימו לב שאם רוצים, אפשר להעביר למתודה nextToken הגדרה אחרת של טקסט מפריד. הפרמטר הזה נקרא delim, קיצור של המילה delimiter, שמשמעותה באנגלית "מפריד".

דף עזר: מחלקת CensoredStringTokenizer: מימוש חלקי ודוגמת שימוש

המימוש החלקי שמתואר כאן מחזיק את המילה הלא מצונזרת הבאה בטקסט במשתנה nextUncensoredToken. מי שאחראי לתחזק את המשתנה הזה היא מתודה פרטית ששמה consume, שמשמעות שמה באנגלית היא "לאכול". המתודה הזאת מדלגת על ("אוכלת את") המילים המצונזרות הבאות בטקסט, ושמה ב- nextUncensoredToken את המילה הלא מצונזרת הבאה בטקסט.

```
import java.util.StringTokenizer;

/* Breaks a given text into tokens (words) while eliminating undesired words
from the text. The constructor receives a text and a list of words to be
censored. Each call to nextToken() will then return the next token in the
text that does not appear in the list of words to be censored. */

public class CensoredStringTokenizer {
  private String[] censoredWords;      // The words to be censored.
  private StringTokenizer tokenizer;   // Used to break the text into tokens.
  private String nextUncensoredToken:  // The next uncensored token.

  // Constructs a new CensoredStringTokenizer. Receives two parameters:
  // the string to be tokenized, and an array of words to be censored.
  public CensoredStringTokenizer(String text, String[] censoredWords) {
  }

  // Returns the next uncensored word in the text.
  public String nextToken() {}

  // Returns true if there are more uncensored words to be read
  // from the text, false otherwise.
  public boolean hasMoreTokens() {}

  // Returns true if the given word is in the list of words
  // to be censored, false otherwise.
  public boolean isCensored(String word) {}

  // Consumes all the next tokens in the text that should be censored,
  // and sets nextUncensoredToken to the value of the next uncensored token.
  private void consume() {}
}
```

קטע הקוד הבא מדגים שימוש במחלקה הזאת:

```
 public class CensoredStringTokenizerDemo {
   public static void main(String[] args) {
     String text = "eat the apple or the banana but not the sushi";
     String[] censoredWords = {"the", "and" , "or", "not"};
     CensoredStringTokenizer cst =
                  new CensoredStringTokenizer(text, censoredWords);
     while (cst.hasMoreTokens()) {
       System.out.println(cst.nextToken());
     }
   }
 }
```

הרצה של קטע הקוד הזה תגרום להדפסת הפלט הבא:

```
eat
apple
banana
but
sushi
```