תשובות למבחן וונגי: סמסטר 2025א', 20 בפברואר 2025, שאלון 463, מועד 66

תשובה לשאלה 1: (לתרגול השאלה בוונגי פלטפורמת הקוד לחץ כאן!):

```
public static int lcs(String st1, String st2, String st3) {
    return lcsHelper(st1, st2, st3, st1.length(), st2.length(), st3.length());
}

private static int lcsHelper(String st1, String st2, String st3, int i, int j, int k) {
    if (i == 0 || j == 0 || k == 0) {
        return 0;
    }

    if (st1.charAt(i - 1) == st2.charAt(j - 1) && st2.charAt(j - 1) == st3.charAt(k - 1)) {
        return 1 + lcsHelper(st1, st2, st3, i - 1, j - 1, k - 1);
    }

    return Math.max(Math.max(
                    lcsHelper(st1, st2, st3, i - 1, j, k),
                    lcsHelper(st1, st2, st3, i, j - 1, k)),
            lcsHelper(st1, st2, st3, i, j, k - 1));
}
```

תשובה לשאלה 2: (לתרגול השאלה בוונגי פלטפורמת הקוד לחץ כאן!):

```
public static int findDivisor(int[] arr) {
    int min = 0, minAbs = Integer.MAX_VALUE;

    // Find minimum value (ignoring 1s)
    for (int i = 0; i < arr.length; i++)
    {
        int abs = Math.abs(arr[i]);
        if (abs != 1 && abs < minAbs)
        {
            minAbs = abs;
            min = arr[i];
        }
    }

    // If no valid minimum found (array contains only 1s or is empty)
    if (minAbs == Integer.MAX_VALUE)
        return -1;

    // Check if min is a divisor of all numbers
    for (int i = 0; i < arr.length; i++)
    {
        if (Math.abs(arr[i]) != 1 && arr[i] % min != 0)
            return -1;
    }

    return min;
}
```

סיבוכיות זמן: $O(n)$, לולאה ראשונה למציאת המינימום, לולאה שנייה לבדיקת חלוקה בצורה ליניארית.

סיבוכיות מקום: $O(1)$ הוקצו מספר קבוע של משתנים.

תשובה לשאלה 3:

סעיף א': 3.

סעיף ב': השיטה מחזירה את העומק (המרחק) של הצומת שמכיל את הערך num בעץ הבינארי. העומק של השורש הוא 0, ילדיו בעומק 1 וכן הלאה. אם הערך לא נמצא בעץ, השיטה מחזירה Integer.MIN_VALUE.

סעיף ג': את הצומת שמכיל את הערך 11.

סעיף ד': השיטה מוצאת את האב המשותף הנמוך ביותר (LCA - Lowest Common Ancestor) של שני צמתים עם הערכים x ו-y בעץ הבינארי. האב המשותף הנמוך ביותר הוא הצומת העמוק ביותר בעץ שיש לו בתת-העץ שלו הן את הצומת עם הערך x והן את הצומת עם הערך y. אם אחד הערכים נמצא בצומת עצמו, הצומת הזה יוחזר כאב המשותף. אם אחד הערכים לא נמצא בעץ, השיטה מחזירה null.

סעיף ה': 5.

סעיף ו': השיטה מחשבת את אורך המסלול הקצר ביותר בין שני צמתים עם הערכים x ו-y בעץ הבינארי. היא עושה זאת על ידי מציאת האב המשותף הנמוך ביותר (LCA) של שני הצמתים, ואז חישוב סכום המרחקים מה-LCA לכל אחד מהצמתים. אם אחד הערכים לא נמצא בעץ, השיטה מחזירה Integer.MIN_VALUE.

תשובה לשאלה 4:

סעיף א':

{81, 17, 8, 9, 101, 100, 24, 8, 60, 10, 11, 12}

סעיף ב':

{81, 17, 8, 60, 101, 100, 24, 8, 9, 10, 11, 12}

סעיף ג':

{12, 11, 10, 9, 8, 24, 100, 101, 60, 8, 17, 81}

סעיף ד': כל רשימה באורך 7 המהווה פלינדרום תקיים לדוגמה

{1, 2, 3, 4, 3, 2, 1}

סעיף ה': השיטה מחליפה את מיקומם של שני איברים סימטריים ברשימה - האיבר במיקום k מההתחלה והאיבר במיקום k מהסוף. כלומר, אם אורך הרשימה הוא n, השיטה מחליפה בין האיבר במיקום k לבין האיבר במיקום (n-k+1). במקרה שk גדול מאורך הרשימה, השיטה מחזירה null. השיטה מטפלת במקרים מיוחדים כגון כאשר האיברים סמוכים זה לזה או כאשר אחד מהם הוא ראש הרשימה. אם מפעילים את השיטה על כל האינדקסים מ-1 עד מחצית אורך הרשימה, התוצאה היא היפוך מלא של הרשימה. במקרה של רשימה פלינדרומית (סימטרית), הפעלת השיטה לא תשנה את הרשימה.

תשובה לשאלה 5:

סעיף א': בקריאה ל-secret(a, 6, 2), השיטה מחפשת את המספרים 6 ו-2 במערך a.
6 נמצא באינדקס 8, ו-2 נמצא באינדקס 5. המרחק ביניהם הוא |5-8| = 3.
לכן, הערך המוחזר הוא: 3

סעיף ב': בקריאה ל-secret(a, 2, 6), השיטה מחפשת את המספרים 2 ו-6 במערך a.
2 נמצא באינדקס 5, ו-6 נמצא באינדקס 8. המרחק ביניהם הוא |8-5| = 3.
לכן, הערך המוחזר הוא: 3

סעיף ג':
כדי שהשיטה תחזיר 1, אנחנו צריכים למצוא שני ערכים במערך שנמצאים בדיוק במרחק של מקום אחד זה מזה.
הזוגות שמקיימים תנאי זה הם:
(3, 5) או (5, 3) - בעמדות 1 ו-2
(5, 4) או (4, 5) - בעמדות 2 ו-3
(4, 8) או (8, 4) - בעמדות 3 ו-4
(8, 2) או (2, 8) - בעמדות 4 ו-5
(3, 6) או (6, 3) - בעמדות 7 ו-8
(6, 5) או (5, 6) - בעמדות 8 ו-9
סך-הכל 12.

סעיף ד': השיטה לא יכולה להחזיר 9 עבור המערך הזה.
אורך המערך הוא 10, ולכן המרחק המקסימלי האפשרי בין שני איברים הוא 9 (מאינדקס 0 עד 9). כדי שהשיטה תחזיר ערך זה, שני האיברים x ו-y צריכים להופיע בדיוק פעם אחת במערך והם צריכים להיות בעמדות 0 ו-9. במערך שלנו, הערכים בעמדות אלה הם 1 ו-5, אבל 5 מופיע מספר פעמים במערך.
כשמחשבים את המרחק המינימלי, השיטה תמצא שהאיברים האלה קרובים יותר במקומות אחרים במערך.

סעיף ה': כן, יש זוגות פרמטרים שיחזירו את הערך 2. הזוגות הם:
(1, 5) או (5, 1) - כאשר 1 באינדקס 0 ו-5 באינדקס 2, המרחק הוא 2
(5, 8) או (8, 5) - כאשר 5 באינדקס 2 ו-8 באינדקס 4, המרחק הוא 2
(4, 6) או (6, 4) - כאשר 4 באינדקס 6 ו-6 באינדקס 8, המרחק הוא 2
שימו לב שלמרות שיש זוגות אחרים במערך שיש ביניהם מרחק של 2 עמדות, הפונקציה secret מחזירה את המרחק המינימלי בין ההופעות של הערכים, ולכן אם יש הופעות של אותם ערכים במרחק קטן יותר, הפונקציה תחזיר את המרחק הקטן יותר.

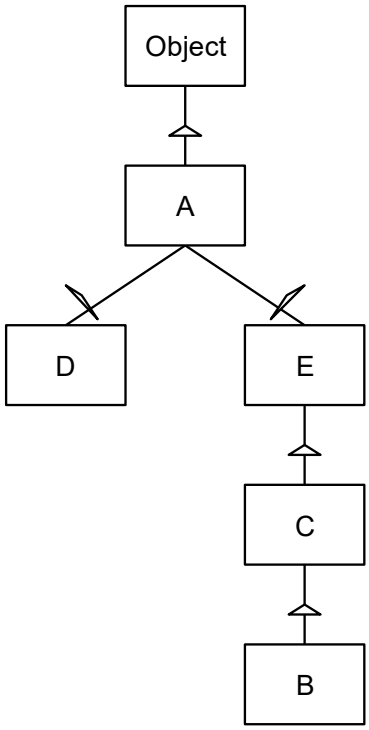
סעיף ו': ב-secret1, השיטה עוברת על המערך פעמיים באמצעות פעולת המודולו (i % a.length). זה מאפשר לה לשקול מרחקים שמקיפים את המערך.
בקריאה ל-secret1(a, 1, 6), השיטה מחפשת את הערכים 1 ו-6 במערך. הערך 1 נמצא באינדקס 0, והערך 6 נמצא באינדקס 8. אם נתייחס רק למעבר אחד על המערך, המרחק ביניהם הוא 8.
מאידך במעבר השני:
הערך 1 נמצא שוב באינדקס 0, שבמעבר השני מיוצג ע"י i = 10 (0 + אורך המערך)
הערך 6 נמצא באינדקס 8
המרחק |8-10| = 2 קטן יותר מ-8, ולכן השיטה מחזירה: 2.

סעיף ז': השיטה secret מחזירה את המרחק המינימלי בין מופע של הערך x למופע של הערך y במערך a.
משמעות הערך המוחזר: המספר המינימלי של אינדקסים שמפרידים בין הופעה כלשהי של המספר x לבין הופעה כלשהי של המספר y במערך.
במקרי קצה:

- אם x או y לא מופיעים במערך, מוחזר Integer.MAX_VALUE.
- אם x ו-y זהים, מוחזר המרחק המינימלי בין שתי הופעות עוקבות של אותו ערך במערך.
- אם אחד הערכים או שניהם לא מופיעים במערך, מוחזר Integer.MAX_VALUE.

תשובה לשאלה 6:

סעיף א':



סעיף ב':

```
public A()
public A(int value)
public A(Object obj)

public B()
public B(int i, double d, char c, boolean b)
public B(E e, A a)

public C()

public D(String s)

public E()
public E(A a)
```

סעיף ג': E ו-B.

סעיף ד':

```
// E:
public String f() {
    return "f1";
}

// B:
public String f() {
    return "f2";
}
```

מההדפסות בקוד נראה שמחלקה E מחזירה "f1" בשיטה ()f.

מחלקה B מחזירה "f2" בשיטה ()f ודורסת את השיטה מ-E.

מחלקה C יורשת את ()f מ-E ללא דריסה.

סעיף ה':

a) שגיאת קומפילציה: מנסים להשים אובייקט מטיפוס Object (o1) במשתנה מטיפוס A (a4), מתבצעת המרה מרומזת מטיפוס אב (Object) לטיפוס בן (A). בזמן ריצה, ג'אווה בודקת האם האובייקט אכן מסוג A, ומכיוון שהוא לא (הוא אובייקט Object רגיל) צריכה להיות שגיאת קומפילציה כי אין המרה מפורשת.

b) תקין: שני המשתנים הם מטיפוס Object, ולכן ניתן להשים משתנה אחד בשני ללא בעיה. פעולה זו תגרום ל-o2 להצביע לאותו אובייקט שאליו o1 מצביע.

c) שגיאת זמן ריצה: יש המרה מפורשת (explicit casting) של o1 לטיפוס A, ואז קריאה למתודה toString() ההמרה המפורשת עוברת קומפילציה, אך בזמן ריצה נזרקת שגיאת ClassCastException מכיוון ש-o1 הוא אובייקט מסוג Object ולא ניתן להמיר אותו לטיפוס A.

d) תקין: toString() היא שיטה שקיימת בכל אובייקט בג'אווה (נירשת מ-Object), ולכן a4 (שהוא מטיפוס A) יכול לקרוא לה ללא בעיה. התוצאה תהיה מחרוזת המייצגת את האובייקט, למשל "A@4dd8dc3" (אם המחלקה A לא מחליפה את מתודת toString()).

e) תקין: o2 נוצר במקור כאובייקט מסוג A (ראה Object o2 = new A(o1)), ולמרות שהוא מוחזק במשתנה מטיפוס Object, ניתן להמיר אותו בחזרה ל-A ללא בעיה בזמן ריצה. לאחר ההמרה, קריאה ל-toString() תפעיל את המתודה מהמחלקה A (או מהמחלקה Object אם A לא מחליפה אותה), ותחזיר מחרוזת כמו "A@568db2f2".