סדרת פיבונאצ'י

הקדמה

סדרת פיבונאצ'י היא סדרה רקורסיבית (סדרה שבה האיבר ה-n נקבע על סמך איברים קודמים), המוגדרת על-ידי כלל הנסיגה הבא:

$$F_1 = F_2 = 1$$
$$F_n = F_{n-2} + F_{n-1}$$

כאשר F_n הוא איבר במקום ה-n בסדרת פיבונאצ'י.

לאיברי הסדרה, הנקראים "מספרי פיבונאצ'י", יש כמה תכונות מעניינות. אחת מהן קשורה ליחס בין איברים עוקבים של הסדרה, כלומר למנה $\frac{F_n}{F_{n-1}}$.

אם מסתכלים על הגבול של המנה הזו - $\lim_{n\to\infty} \frac{F_n}{F_{n-1}}$ (כלומר, לאיזה מספר המנה מתקרבת כש-n שואף לאינסוף), מקבלים מספר אי-רציונלי: $\Phi = \frac{1+\sqrt{5}}{2} = 1.618033....$.

המספר הזה מכונה "**יחס הזהב**" או "**מספר הזהב**" ונהוג לסמנו באות Φ.

פתרון

דרך פשוטה למציאת איבר n בסדרת פיבונאצ'י הוא בעזרת יחס הזהב,

חשוב - שיטה זו עובדת החל מאינדקס החמישי והשיטה יודעת לחשב עד לאיבר באינדקס ה-34 ולא יותר מזה!

שיטה **אינדוקטיבית**:

```java
static double PHI = (1 + Math.sqrt(5)) / 2;
static int[] fib = { 0, 1, 1, 2, 3, 5 };

public static int goldenInduction(int n) {
   if(n <= 5) {
       return fib[n];
   }
   int ans = fib[5];
   for (int i = 5; i < n ; i++) {
       ans = (int) Math.round(ans * PHI);
   }

   return ans;
}
```

שיטה **רקורסיבית**:

```java
static double PHI = (1 + Math.sqrt(5)) / 2;
static int[] fib = { 0, 1, 1, 2, 3, 5 };

public static int goldenRecursive(int n) {
   if(n <= 5) {
       return fib[n];
   }
   if(n == 6)
       return (int) Math.round(5 * PHI);

   return (int) Math.round( goldenRecursive(n-1) * PHI);
}
```

פתרון בעזרת נוסחת binet בסיבוכיות זמן **קבוע** $O(1)$, אך גם כאן יש הגבלה עד איבר באינדקס 34 ולא יותר מזה! הנוסחה: $S_n = \Phi^n - \frac{(-\Phi^{-n})}{\sqrt{5}}$.

```java
static double PHI = (1 + Math.sqrt(5)) / 2;

public static int binetFormula(int n) {
   return (int) ((Math.pow(PHI, n) - Math.pow(-PHI, -n))/Math.sqrt(5));
}
```

עבור איברי ב**אינדקס שלילי** של סדרת פיבונאצ'י נראה את הפתרון **הרקורסיבי** הבא,
כאשר הפתרון לא נפתר עם יחס הזהב אלא כפתרון פשוט שעובד **לכל איבר n** ולא מוגבל כמו הפתרונות הקודמים שהצגנו עבור יחס הזהב.

```java
public static int fiboRecursive(int n) {
   if( n == 0 || n == 1) {
       return n;
   }
   return fiboRecursive(n-1) + fiboRecursive(n-2);
}


public static int negativeFiboRecursive(int n) {
   if(n >= 0) { // if its not a negative index then use the positive method.
       return fiboRecursive(n);
   }
   return negativeFiboRecursive(n+2) - negativeFiboRecursive(n+1);
}
```

סדרת פיבונאצ'י ומטריצות

שיטה נוספת להחזיר איבר באינדקס n בסדרת פיבונאצ'י היא בעזרת חישוב נוסחת הסדרה על-ידי מטריצות.

$$\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^n = \begin{pmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{pmatrix}$$

כאשר n הוא האיבר באינדקס n של סדרת פיבונאצ'י.

אלגוריתם

- כדי לחשב את סדרת פיבונאצ'י נשתמש בכפל מטריצות.
- נגדיר מטריצה ראשונית להיות ערכי ההתחלה של הסדרה ונכפיל אותה n-1 פעמים.

הוכחה

נוכיח באינדוקציה את נכונות האלגוריתם.

- הנחת האינדוקציה - $A^n = \begin{pmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{pmatrix}$.
- צעד האינדוקציה - צריך להוכיח כי $A^{n+1} = \begin{pmatrix}F_{n+2} & F_{n+1}\\ F_{n+1} & F_n\end{pmatrix}$:

$$A^{n+1} = A^n \cdot A = \begin{pmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{pmatrix} \cdot \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix} = \begin{pmatrix}F_{n+2} & F_{n+1}\\ F_{n+1} & F_n\end{pmatrix}$$

∎

כדי לקבל פתרון נכפול את $\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}$ n פעמים, ונחזיר את הפתרון שהתקבל במיקום $mat[0][0]$ כלומר את F_{n+1} זאת מאחר וסדרת פיבונאצ'י מתחילה באיבר 0 ועד עכשיו ספרנו אינדקסים החל מאיבר 1.

מימוש הפתרון בשיטה **אינדוקטיבית** בזמן ריצה $O(n)$

```java
public static int fibMatrix(int n) {
   if (n <= 0) {
       return 0;
   }

   int[][] fiboMatrix = { {1,1}, {1,0} };

   matrixPower(fiboMatrix,n);

   return fiboMatrix[0][0];
}

private static void matrixPower(int[][] matrix, int n) {

   int[][] ans = { {1,1}, {1,0} };
```

```
    for (int i = 1; i < n-1; i++) {
        matrixMultiply(matrix,ans);
    }

}

private static void matrixMultiply(int[][] mat1, int[][] mat2) {

    int x =  mat1[0][0] * mat2[0][0] + mat1[0][1] * mat2[1][0];
    int y =  mat1[0][0] * mat2[0][1] + mat1[0][1] * mat2[1][1];
    int z =  mat1[1][0] * mat2[0][0] + mat1[1][1] * mat2[1][0];
    int w =  mat1[1][0] * mat2[0][1] + mat1[1][1] * mat2[1][1];

    mat1[0][0] = x;
    mat1[0][1] = y;
    mat1[1][0] = z;
    mat1[1][1] = w;
}

public static void main(String[] args) {
    System.out.println(fibMatrix(6));
}
```

בשיטה **הרקורסיבית**, הנוסחה: $\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^n = \begin{pmatrix}F_{n+1} & F_n\\ F_n & F_{n-1}\end{pmatrix}$, תספק לנו פתרון בסיבוכיות זמן ריצה של רק $O(log\ n)$
בדומה לפתרון חישוב חזקה ברקורסיה, ננצל את אותה השיטה לטובת הפתרון שלנו.
נכפול באופן רקורסיבי את המטריצה, המימוש מאוד דומה לקודם עם הבדל בפונקציה power.

```
     public static int fibMatrixLog(int n) {
        if (n <= 0) {
            return 0;
        }

        int[][] fiboMatrix = { {1,1}, {1,0} };

        matrixPowerLog(fiboMatrix,n-1);

        return fiboMatrix[0][0];
    }



    private static void matrixPowerLog(int[][] matrix, int n) {

        if(n == 0 || n == 1)
```

```
            return;

        int[][] ans = { {1,1}, {1,0} };

        matrixPowerLog(matrix, n/2);
        matrixMultiplyLog(matrix,matrix);

        if(n%2 != 0) {
            matrixMultiplyLog(matrix,ans);
        }

    }


   private static void matrixMultiplyLog(int[][] mat1, int[][] mat2) {

        int x =  mat1[0][0] * mat2[0][0] + mat1[0][1] * mat2[1][0];
        int y =  mat1[0][0] * mat2[0][1] + mat1[0][1] * mat2[1][1];
        int z =  mat1[1][0] * mat2[0][0] + mat1[1][1] * mat2[1][0];
        int w =  mat1[1][0] * mat2[0][1] + mat1[1][1] * mat2[1][1];

        mat1[0][0] = x;
        mat1[0][1] = y;
        mat1[1][0] = z;
        mat1[1][1] = w;
    }

    public static void main(String[] args) {
        System.out.println(fibMatrixLog(6));
    }
```