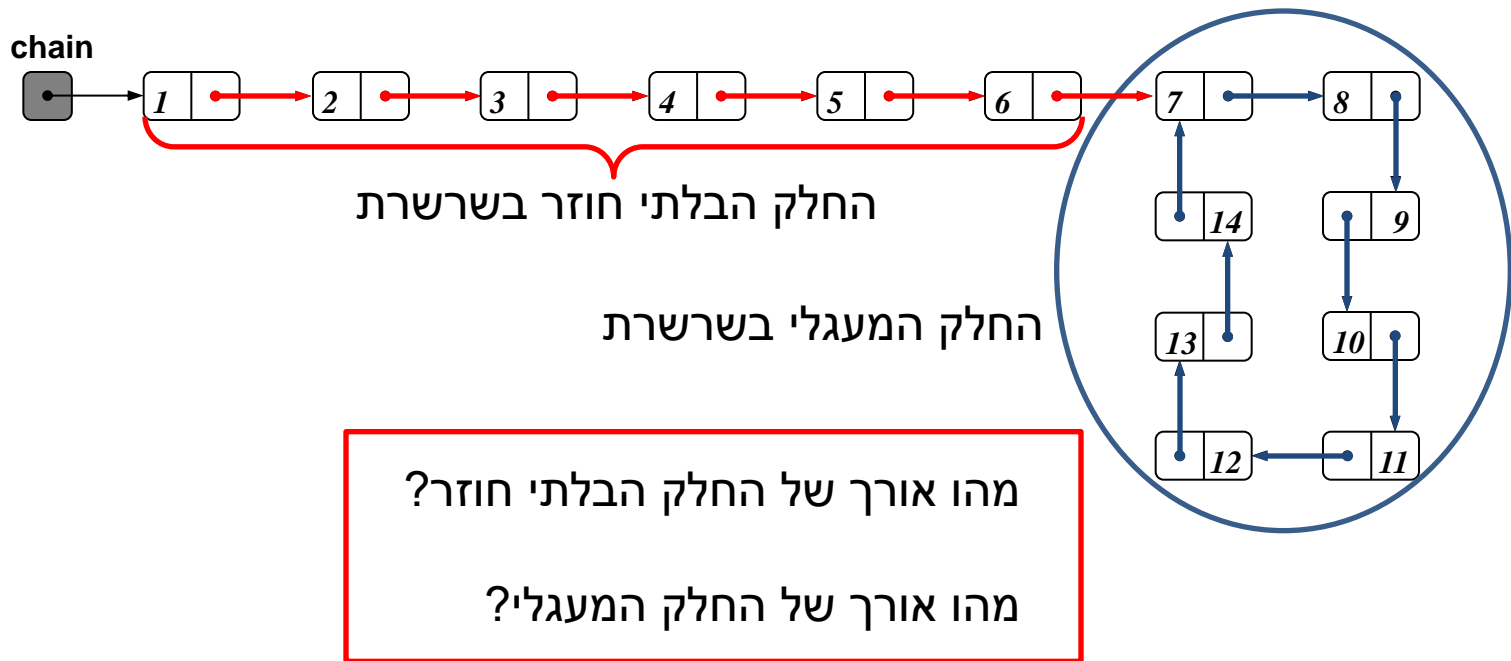


אלגוריתם גנרי "הארנב והצב"

סריקה של שרשרת חוליות

במהלך הסריקה של שרשרת החוליות יש לעבור על כל החוליה זו אחר זו, מתחילתה עד סופה ולעצור. אם ידוע שבסוף שרשרת חוליות קיימת חוליה שערך העוקב שלה הוא **null**, אז הסריקה תסתיים בתנאי העצירה הזה. ואם בשרשרת חוליות יש מעגל (החוליה "האחרונה" מצביעה לחוליה כלשהי בשרשרת), לא נגיע לתנאי העצירה והסריקה לא תסתיים.



אלגוריתם גנרי "הארנב והצב"

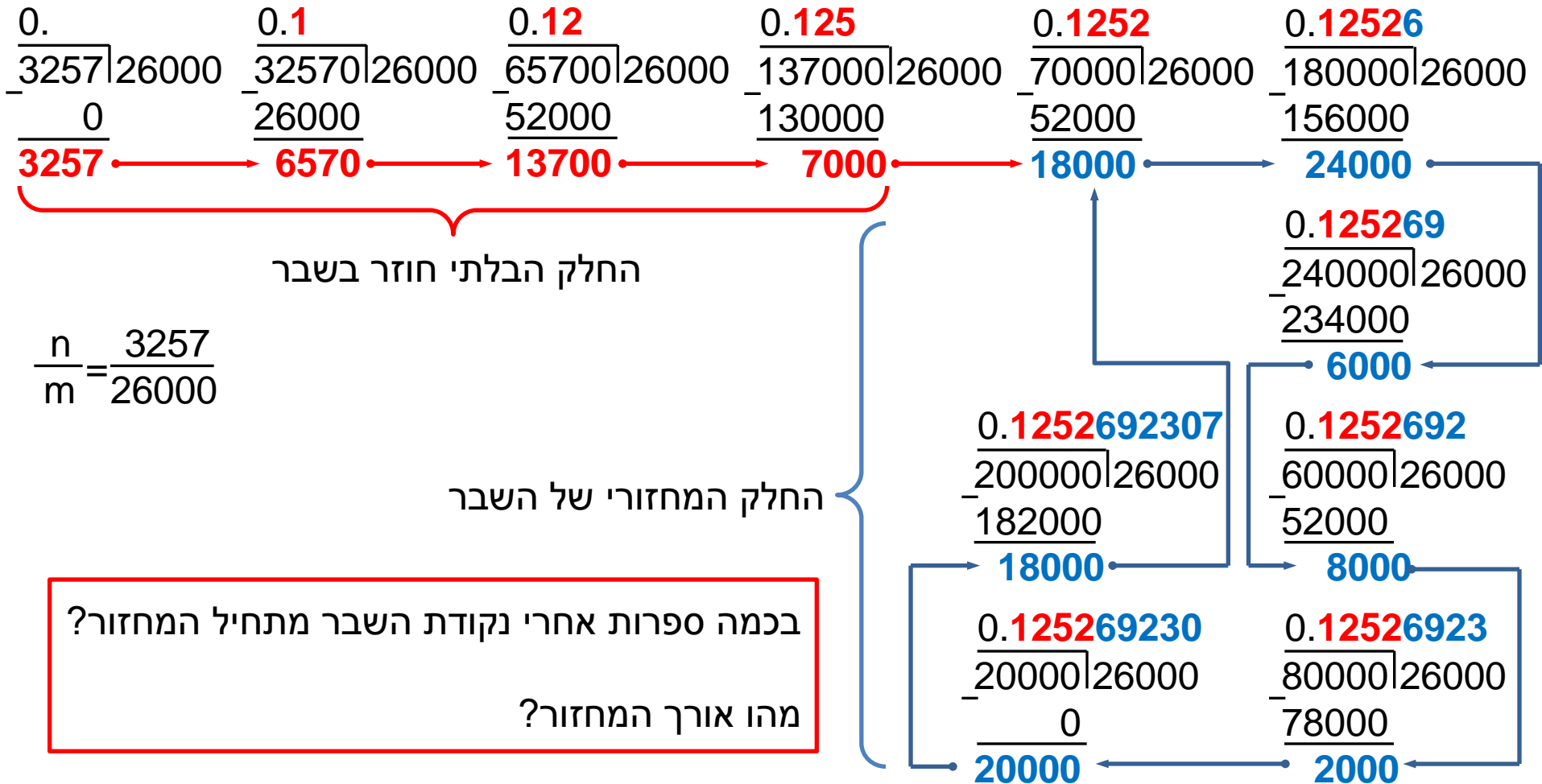
המרת שבר פשוט לשבר עשרוני

כדי להפוך שבר פשוט לשבר עשרוני יש לחלק את המונה במכנה. במהלך החילוק של המונה במכנה יכולות להתקיים שתי אפשרויות:

- החילוק יסתיים, והשארית האחרונה תהיה 0.
- אחת מהשאריות תחזור על עצמה ואחריה עוד שאריות שכבר הופיעו, וכך לא יסתיים החילוק לעולם.

אלגוריתם גנרי "הארנב והצב"

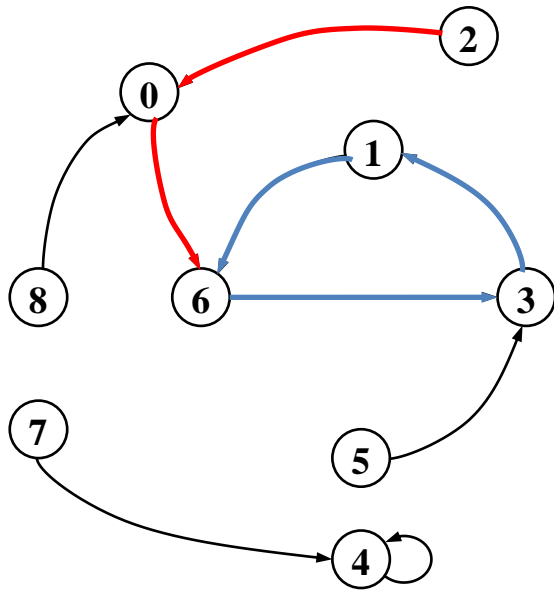
המרה שבר פשוט לשבר עשרוני



אלגוריתם גנרי "הארנב והצב"

גרף פונקציונלי

גרף פונקציונלי הוא גרף מכוון שבו דרגת היציאה של כל צומת לא גדולה מאחת. נצמיד לכל צומת בעץ מספר סידורי (בלתי תלוי בסדר הצמתים). פונקציית הגרף היא התאמה, שמשייכת לכל צומת בקבוצת הצמתים צומת יחיד באותה הקבוצה.



x	f(x)
0	6
1	6
2	0
3	1
4	4
5	3
6	3
7	4
8	0

מהו אורך של החלק הבלתי חוזר?
מהו אורך של החלק המעגלי?

נגדיר מסלול בגרף פונקציונלי כסדרה של צמתים שמתחילה בצומת מסוים. מעגל במסלול הוא סדרה של צמתים שמתחילה ומסתיימת באותו צומת.

במהלך סריקת המסלול בגרף מהצומת $x_0 = 2$, נקבל סדרה אין סופית: **2, 0, 6, 3, 1, 6, 3, 1, ...**

אלגוריתם גנרי "הארנב והצב"

תכנות פונקציונלי

נתבונן בפונקציה $f: X \rightarrow X$, שמוגדרת על קבוצה סופית X ומתאים לכל איבר $x \in X$ איבר יחיד $f(x)$ של אותה הקבוצה $(f(x) \in X)$.

אלגוריתם גנרי "הארנב והצב"

תכנות פונקציונלי

גרף פונקציונלי מוגדר
על ידי מערך graph

```
x = x0;  
x = graph[x];
```

אורך המערך הוא סופי

שבר פשוט $\frac{n}{m}$

```
x = n;  
x = (10 * x) % m;
```

מספר השאריות לפי
הבסיס m הוא סופי

שרשרת חוליות chain

```
x = chain;  
x = x.getNext();
```

מספר החוליות הוא
סופי

בכל חישוב עבור איבר התחלתי כלשהו x_0 נבנה סידרה של איברים (מסלול חישוב), שמתקבלים לפי אותו חוק – פונקציה f :

$$x_0 \xrightarrow{f(x_0)} x_1 \xrightarrow{f(x_1)} x_2 \xrightarrow{f(x_2)} x_3 \cdots x_i \xrightarrow{f(x_i)} x_{i+1} \cdots$$

כל איבר מחושב באמצעות האיבר הקודם לו בסדרה

כמות האיברים השונים במסלול החישוב לא יהיה גדול יותר מגודל הקבוצה, ובמוקדם או במאוחר תוצאת החישוב תחזור על עצמה או שהמסלול יתקע.

אלגוריתם גנרי "הארנב והצב"

ממשק פונקציונלי

- **ממשק פונקציונלי** הוא ממשק בעל פעולה אבסטרקטית אחת בלבד.
- החל מגרסה Java 8, ניתן להוסיף לכל ממשק מפעולות ברירת מחדל – פעולות ממומשות. בתחילת פעולות ברירת המחדל יש לציין "default".
- **אָנוֹטַצְיָה** (*@FunctionalInterface* (annotation) מבטיחה שהממשק מגדיר בדיוק פעולה אבסטרקטית אחת.

```
@FunctionalInterface
public interface IFunction<T> {
    T apply(T t);
}
```

הפעולה האבסטרקטית `apply` מחשבת את ערך הפונקציה עבור ארגומנט `t` ומחזירה את תוצאת החישוב.

אלגוריתם גנרי "הארנב והצב"

ממשק פונקציונלי

קטע הקוד הבא מגדיר את גוף הפונקציה האנונימית ומציב אותו בתוך משתנה מטיפוס הממשק
IFunction

```
IFunction<Node<T>> next = new IFunction<Node<T>>(){  
    @Override  
    public Node<T> apply(Node<T> x) {  
        return x.getNext();  
    }  
};
```



שקול

```
IFunction<Node<T>> next = x -> x.getNext();
```

סגנון פונקציונלי

אלגוריתם גנרי "הארנב והצב"

ממשק פונקציונלי

קטע הקוד הבא מגדיר את גוף הפונקציה האנונימית ומציב אותו בתוך משתנה מטיפוס הממשק IFunction

```
final int denom = 257;
IFunction<Integer> div = new IFunction<Integer>(){
    @Override
    public Integer apply(Integer x) {
        int res = (x * 10) % denom;
        if (res == 0)
            return null;
        return res;
    }
};
```



שקול

סגנון פונקציונלי

```
IFunction<Integer> div = x -> ((x * 10) % denom != 0 )?(x * 10) % denom: null;
```

אלגוריתם גנרי "הארנב והצב"

ממשק פונקציונלי

קטע הקוד הבא מגדיר את גוף הפונקציה האנונימית ומציב אותו בתוך משתנה מטיפוס הממשק
IFunction

```
final int[] graph = {6, 6, 0, 1, 4, 3, 3, 4, 0};  
IFunction<Integer> next = new IFunction<Integer>(){  
    @Override  
    public Integer apply(Integer x) {  
        if (x == -1)  
            return null;  
        return graph[x];  
    }  
};
```



שקול

```
IFunction<Integer> next = x -> graph[x] != -1 ? graph[x]: null
```

סגנון פונקציונלי

אלגוריתם גנרי "הארנב והצב"

מימוש האלגוריתם

```
public class LoopConfiguration {  
    private int prefixLength;           // ← האורך הבלתי חוזר בחישוב  
    private int loopLength;            // ← אורך המחזור של החישוב  
    public LoopConfiguration(int prefixLength, int loopLength) {  
        this.prefixLength = prefixLength;  
        this.loopLength = loopLength;  
    }  
    public int getPrefixLength() {  
        return this.prefixLength;  
    }  
    public int getLoopLength {  
        return this.loopLength;  
    }  
    public static <T> LoopConfiguration detectPrefixLength(IFunction<T> f, T start) {
```

פונקציית מעברים

מצב התחלתי



אלגוריתם גנרי "הארנב והצב"

מימוש האלגוריתם

```
public static <T> LoopConfiguration detectPrefixLength(IFunction<T> f, T start) {  
    T slow = start; // ← הפניה שמתקדמת באיבר אחד  
    T fast = start; // ← הפניה שמתקדמת בשני איברים  
    int prefixLength = 0;  
    do {  
        slow = f.apply(slow);  
        fast = f.apply(fast);  
        prefixLength++;  
        If (fast == null)  
            break;  
        prefixLength++;  
        fast = f.apply(fast);  
        If (fast == null)  
            break;  
    } while (!fast.equals(slow));  
  
    if (fast == null) // ← האם החישוב הוא סופי  
        return new LoopConfiguration(prefixLength, 0);  
}
```

השוואה בין שני עצמים
מטיפוס כלשהו

עריכה פקר ולרי

שקף מס 12

אלגוריתם גנרי "הארנב והצב"

מימוש האלגוריתם

```
slow = start;
prefixLength = 0;
while (!fast.equals(slow)) {
    slow = f.apply(slow);
    fast = f.apply(fast);
    prefixLength++;
}
int loopLength = 0;
do {
    fast = f.apply(fast);
    loopLength++;
} while (!fast.equals(slow));

return new LoopConfiguration(prefixLength, loopLength);
}
```

// ← חישוב האורך הבלתי חוזר
השוואה בין שני
עצמים מטיפוס כלשהו

// ← חישוב האורך המחוזר
השוואה בין שני
עצמים מטיפוס כלשהו

אלגוריתם גנרי "הארנב והצב"

היפּוֹתִיזָה קוֹלֵץ

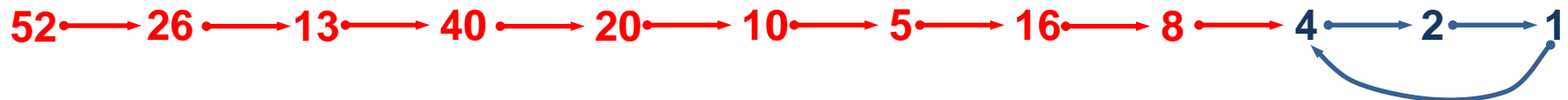
היפּוֹתִיזָה קוֹלֵץ (Collatz) קרויה על-שם מתמטיקאי גרמני לותר קולץ, שפרסם אותה ב-1937. ההיפּוֹתִיזָה קשורה למציאת מחזוריות בתהליך חישוב איטרטיבי של איברים בסדרה שנקראת **סדרת סירקיוז**. הסדרה היא סדרה של מספרים טבעיים שמתחילה במספר כלשהו ומוגדרת באופן רקורסיבי באמצעות כלל הנסיגה:

▪ אם מספר זוגי, אז נחלק אותו בשתיים

▪ אם מספר אי-זוגי, אז נכפיל אותו בשלוש ונוסיף לתוצאה אחת

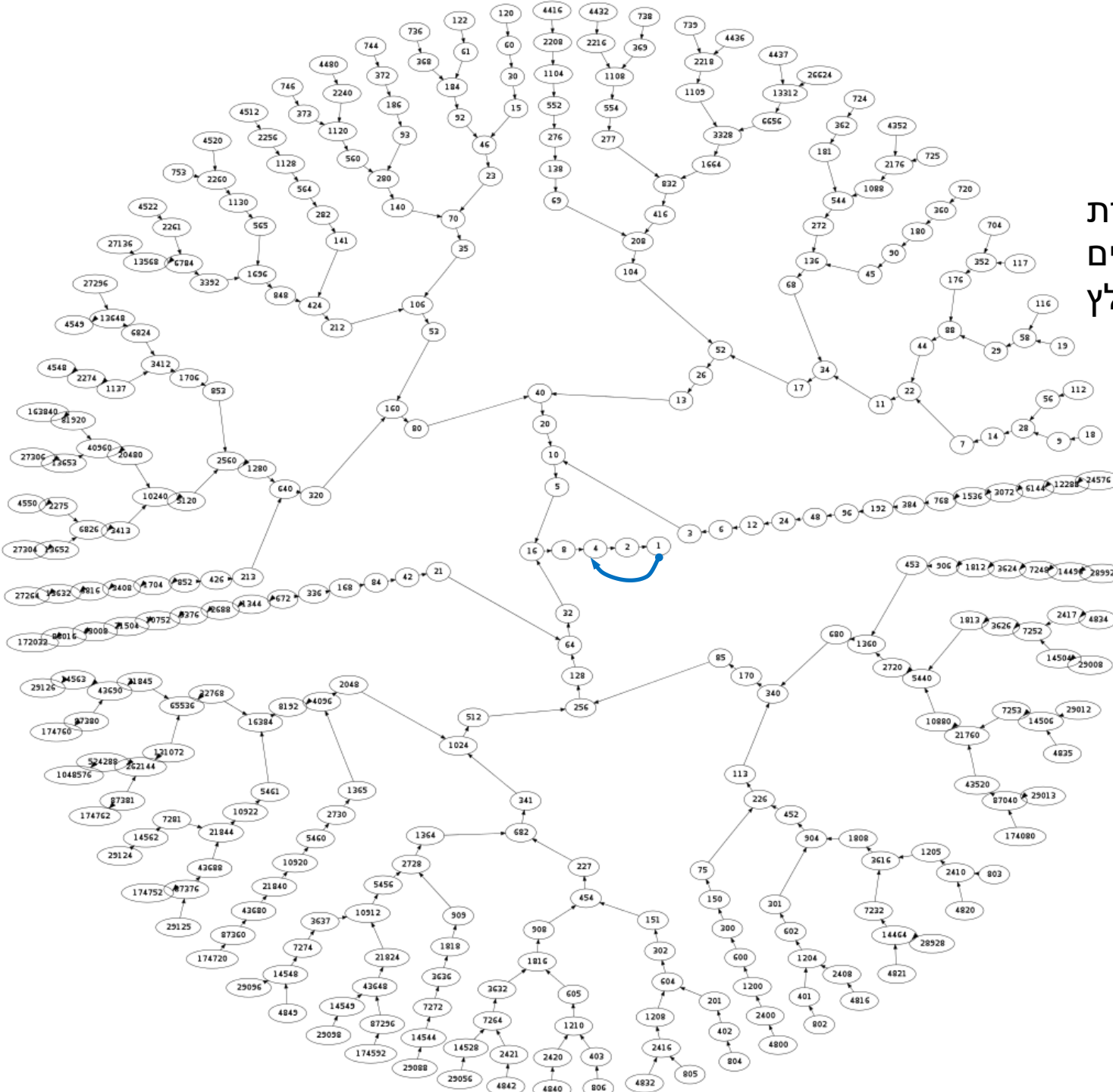
ההיפּוֹתִיזָה אומרת, שכל סדרת סירקיוז מכילה "זנב" (החלק הבלתי חוזר בסדרה) ואחריו חלק מחזורי - מעגל סגור של המספרים 1, 2, 4. וזה לא משנה מהי נקודת ההתחלה.

לדוגמה, הפעלת התהליך על המספר 52:



ההיפּוֹתִיזָה קלה לבדיקה בעזרת מחשב, אך לא ידועה עדיין שום הוכחה.

עץ המציג את סדרת
המספרים שמגיעים
למעגל על פי תהליך קולץ
עד כ-20 שלבי חישוב:



שקף מס 15

אלגוריתם גנרי "הארנב והצב"

פונקציית קולץ

קטע הקוד הבא מגדיר את גוף הפונקציה האנונימית ומציב אותו בתוך משתנה מטיפוס הממשק
IFunction

```
IFunction<BigInteger> funcCollatz = new IFunction<BigInteger>(){  
    @Override  
    public BigInteger apply(BigInteger x) {  
        BigInteger two = new BigInteger("2");  
        if (x.mod(two).equals(BigInteger.ZERO))  
            return x.divide(two);  
        return x.multiply(new BigInteger("3")).add(BigInteger.ONE);  
    }  
};
```



```
IFunction<BigInteger> funcCollatz =  
x -> (x.mod(new BigInteger("2")).equals(BigInteger.ZERO))?  
x.divide(new BigInteger("2")): x.multiply(new BigInteger("3")).add(BigInteger.ONE);
```

סגנון פונקציונלי

אלגוריתם גנרי "הארנב והצב"

בדיקת היפותיזה קולץ

```
BigInteger blnt = new BigInteger("922337203685477580789076543999999");  
LoopConfiguration lc =  
    LoopConfiguration.detectPrefixLength(functionCollatz, blnt);
```

```
System.out.println(lc.getPrefixLength() + " " + lc.getLoopLength());
```

860 3

```
for (int i = 0; i < lc.getPrefixLength(); i++)  
    blnt = functionCollatz.apply(blnt);
```

```
for (int i = 0; i < lc.getLoopLength(); i++) {  
    System.out.print(blnt + "->");  
    blnt = functionCollatz.apply(blnt);  
}
```

4->2->1->

אלגוריתם גנרי "הארנב והצב"

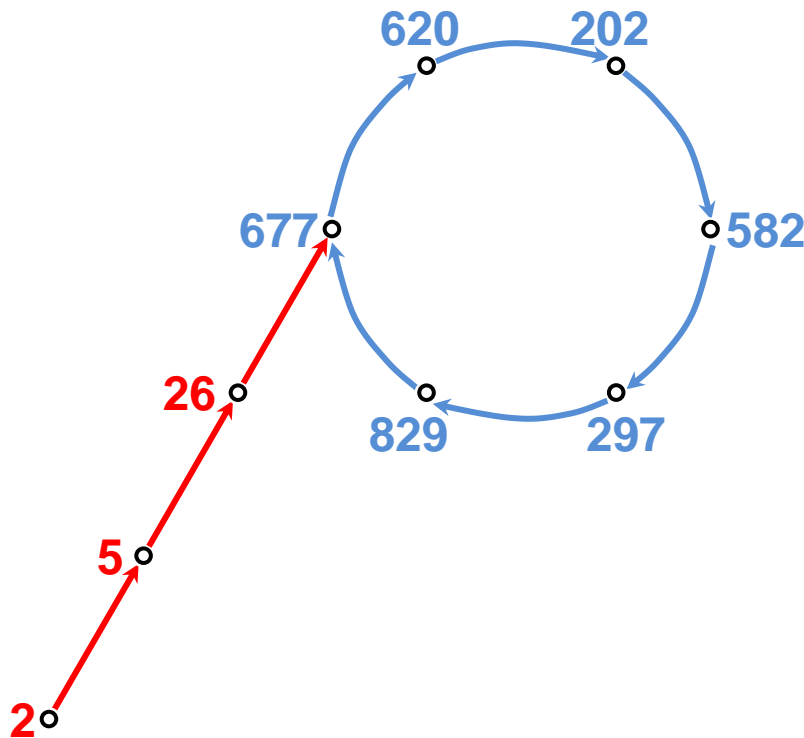
פירוק מספר שלם לגורמים

נתבונן בפונקציה: $f(x) = (x^2 + 1) \% n$

נשתמש בפונקציה בחישוב סדרת הערכים מנקודת התחלה $x_0 = 2$ עבור $n = 1387$.

הסדרה, שהתקבלה, היא סדרת ערכים בקבוצה סופית. הסדרה מוכרחת לחזור על עצמה, והמסלול שהיא מתארת יהפוך למחזורי.

צורת המסלול דומה לאות היוונית ρ



את אלגוריתם הסתברותי "ρ", למציאת גורמים ראשוניים, הציע המתמטיקאי האנגלי ג'ון פולרד ב-1975.

האלגוריתם מבוסס על האלגוריתם של פלויד ("הארנב והצב") למציאת מחזוריות.

אלגוריתם גנרי "הארנב והצב"

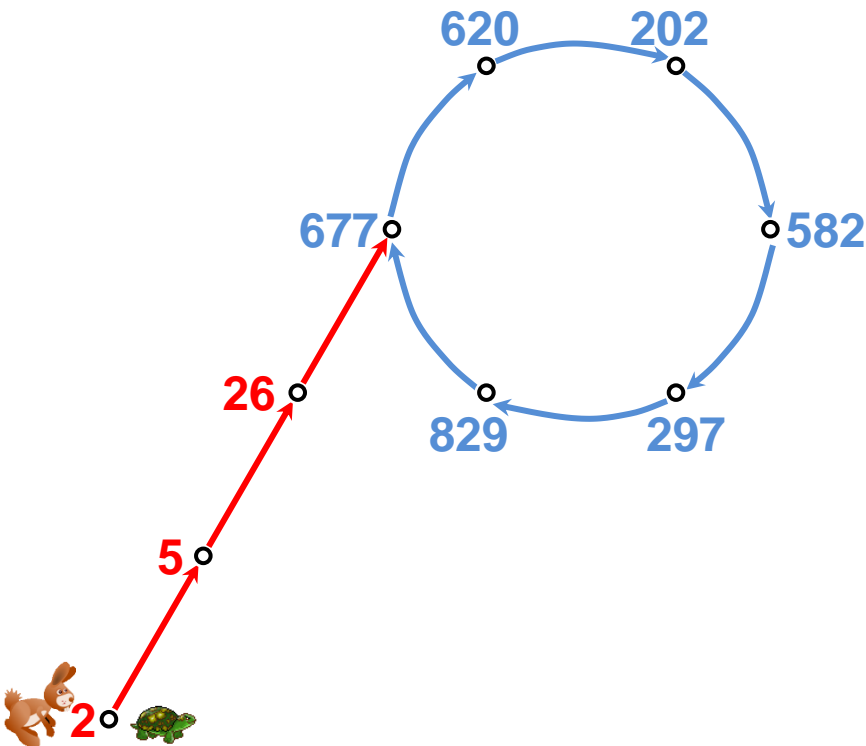
פירוק מספר שלם לגורמים

בהתאם לאלגוריתם, "הצב" ו"הארנב" מתחרים בחישוב סדרת ערכי הפונקציה $f(x) = (x^2 + 1) \% n$. המתחרים מחשבים במקביל את אותה הסדרה, כשמהירות החישוב של "הארנב" כפולה ממהירות "הצב". הסדרה מחושבת פעמיים על ידי המתחרים מאותה נקודת ההתחלה $x_0 = 2$.

בכל שלב התחרות מתבצעת השוואה בין תוצאות החישוב של המתחרים. ההשוואה נעשת על ידי חישוב המחלק המשותף המקסימלי של מספר n וערך מוחלט של הפרש תוצאות החישובים.

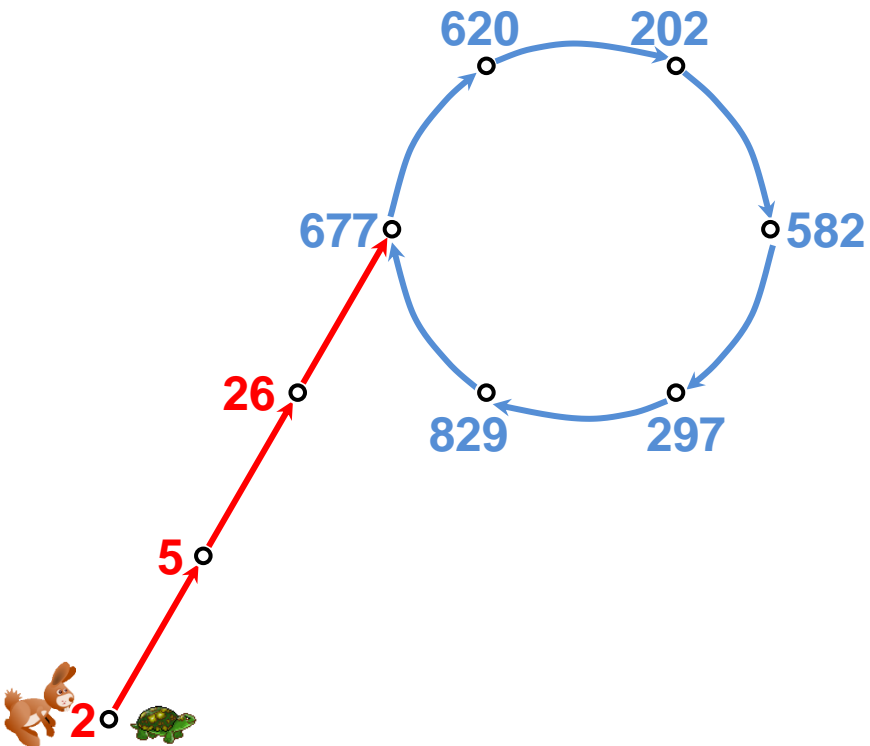
$(n, | \text{"ערך הצב"} - \text{"ערך הארנב"} |) = \text{GCD}$

התחרות מסתיימת, כאשר המחלק המשותף יהיה שונה מאחד. אם התחרות מסתיימת לפני ש"הצב" ו"הארנב" ייפגשו, מדובר במועמד לגורם ראשוני. אם בכל זאת המפגש התקיים (המחלק המשותף שווה למספר), האלגוריתם מסתיים בכישלון.



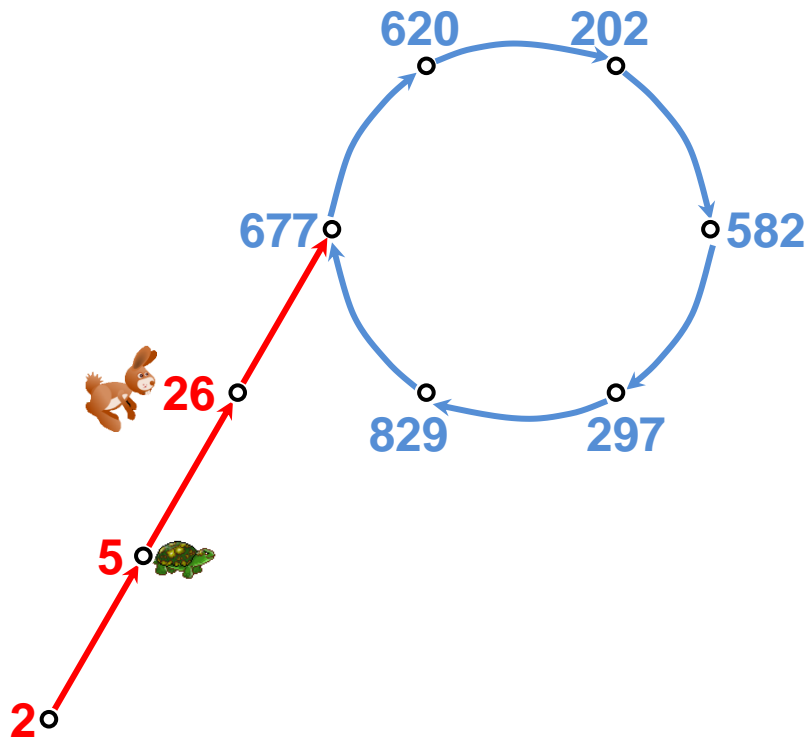
אלגוריתם גנרי "הארנב והצב"

פירוק מספר שלם לגורמים



אלגוריתם גנרי "הארנב והצב"

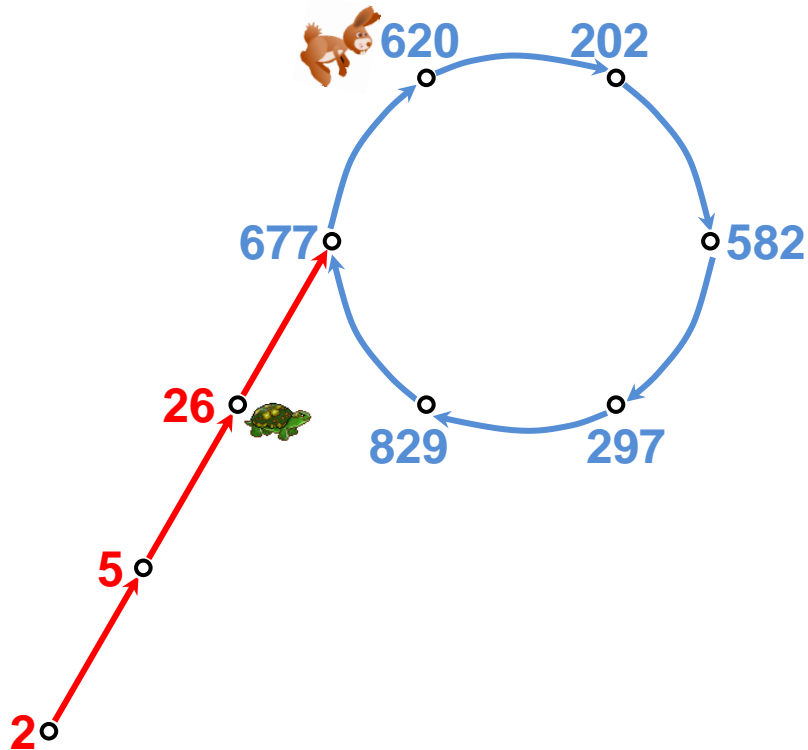
פירוק מספר שלם לגורמים



$$\text{GCD}(|26 - 5|, 1387) = 1$$

אלגוריתם גנרי "הארנב והצב"

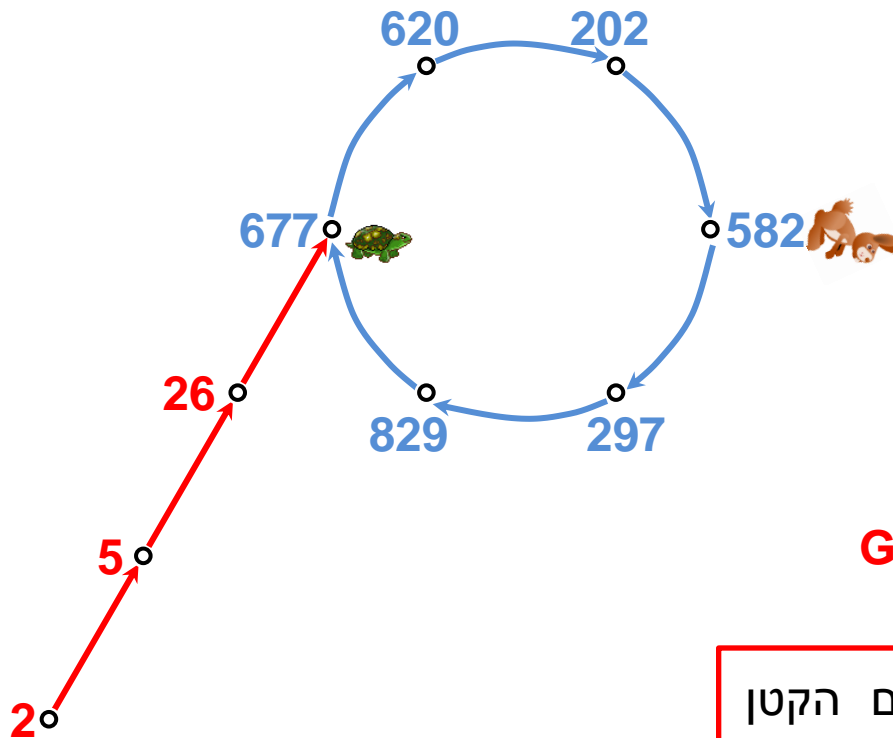
פירוק מספר שלם לגורמים



$$\text{GCD}(|620 - 26|, 1387) = 1$$

אלגוריתם גנרי "הארנב והצב"

פירוק מספר שלם לגורמים



$$\text{GCD}(|582 - 677|, 1387) = 19$$

19 הוא אחד מהגורמים, למעשה הגורם הקטן ביותר של המספר 1387.

אלגוריתם גנרי "הארנב והצב"

פירוק מספר שלם לגורמים

```
public class PollardRho {  
    final static BigInteger TWO = new BigInteger("2");  
    public static BigInteger rho(BigInteger n) {  
        if (n.mod(TWO).equals(BigInteger.ZERO))  
            return TWO;  
        BigInteger slow = TWO;  
        BigInteger fast = slow;  
        BigInteger c = BigInteger.ONE;  
        BigInteger divisor;  
        do{  
            slow = slow.multiply(slow).mod(n).add(c).mod(n);  
            fast = fast.multiply(fast).mod(n).add(c).mod(n);  
            fast = fast.multiply(fast).mod(n).add(c).mod(n);  
            divisor = fast.subtract(slow).gcd(n);  
        } while (divisor.equals(BigInteger.ONE));  
        return divisor;  
    }  
}
```


אלגוריתם גנרי "הארנב והצב"

פירוק מספר שלם לגורמים

```
public static void factor(BigInteger n) {  
    if (n.equals(BigInteger.ONE)) // ←  
        return;  
    if (n.isProbablePrime(20)) { // ←  
        System.out.println(n);  
        return;  
    }  
    BigInteger divisor = rho(n); // ←  
    if (n.equals(divisor)) { // ←  
        System.out.println("Error " + divisor);  
        return;  
    }  
    factor(divisor); // ←  
    factor(n.divide(divisor));  
}
```

כלל העצירה: השלם 1 אינו ראשוני ואינו פריק

כלל העצירה: אלגוריתם מילר-רבין

חיפוש מועמד לגורם ראשוני
החיפוש נכשל

זימונים רקורסיביים

אלגוריתם גנרי "הארנב והצב"

פירוק מספר שלם לגורמים

```
public static void main(String[] args){  
    BigInteger n = new BigInteger("44343535354351600000003434353");  
    factor(n);  
}
```

149

329569479697

903019357561501

אלגוריתם גנרי "הארנב והצב"

אם האלגוריתם מסתיים בכישלון, ישנן מספר אפשרויות:

- לחזור על האלגוריתם עם ערך התחלתי שונה. לדוגמה, מספר אקראי כלשהו הנמוך מ-n.
- לחזור על האלגוריתם עם פונקציה אחרת, בעלת מקדם חופשי גבוה מ-1. לדוגמה, מספר אקראי כלשהו הנמוך מ-n.

אלגוריתם גנרי "הארנב והצב"

```
public class PollardRho {
    final static BigInteger TWO = new BigInteger("2");
    final static SecureRandom random = new SecureRandom();
    public static BigInteger rho(BigInteger n) {
        if (n.mod(TWO).equals(BigInteger.ZERO))
            return TWO;
        BigInteger slow = new BigInteger(n.bitLength(), random);
        BigInteger fast = slow;
        BigInteger c = new BigInteger(n.bitLength(), random);
        BigInteger divisor;
        do{
            slow = slow.multiply(slow).mod(n).add(c).mod(n);
            fast = fast.multiply(fast).mod(n).add(c).mod(n);
            fast = fast.multiply(fast).mod(n).add(c).mod(n);
            divisor = fast.subtract(slow).gcd(n);
        } while (divisor.equals(BigInteger.ONE));
        return divisor;
    }
}
```

אלגוריתם גנרי "הארנב והצב"

פירוק מספר שלם לגורמים

```
public static void factor(BigInteger n) {  
    if (n.equals(BigInteger.ONE)) // ← כלל העצירה: השלם 1 אינו ראשוני ואינו פריק  
        return;  
    if (n.isProbablePrime(20)) { // ← כלל העצירה: בדיקת ראשוניות אשר "כמעט פועלת"  
        System.out.println(n);  
        return;  
    }  
    BigInteger divisor = rho(n); // ← חיפוש מועמד לגורם ראשוני  
    while (n.equals(divisor)) // ← אם החיפוש נכשל, לחזור עליו  
        divisor = rho(n);  
    factor(divisor); // ← זימונים רקורסיביים  
    factor(n.divide(divisor));  
}
```