Prímfelbontás

Forrás Bence: prímfelbontás

```
public class primf {

   public static boolean pre (long N) {
          // prímteszt külön függvényben
   }

   public static void main (String args[]) {
          // főprogram
   }

}
```

Forrás Bence: prímfelbontás

```
public static void main (String args[]) {
   long N = Long.parseLong(args[0]);

   for (long e = 2;e<=N;e++) {
      if (pre(e) == true && N%e == 0) {
         long M = N;
         while (M%e == 0) {
            System.out.println(e);
            M = M/e;
         }
      }
   }
}
```

Forrás Bence: prímfelbontás

```
public static void main (String args[]) {
   long N = Long.parseLong(args[0]);

   for (long e = 2;e<=N;e++) {
      if (pre(e) == true && N%e == 0) {
         long M = N;
         while (M%e == 0) {
            System.out.println(e);
            M = M/e;
         }
      }
   }
}
```

if(pre(e) && N % e == 0)

```
public static boolean pre (long N) {
   boolean pr;

   if (N == 2 || N == 3) { pr = true;
   } else {
      long d = 2;
      long f = (long) Math.sqrt(N);

      while (d<=f && N%d != 0) { d++; }

      if (N%d != 0) {
         pr = true;
      } else {
         pr = false;
      }
   }
   return pr;
}
```

Nem dolgoztunk feleslegesen?

Ami gyorsíthatunk

- A 2-vel való oszthatóságot külön kezeljük, majd elég a páratlan osztókat tesztelni
  (d := d +2)
- Ha van prímtesztünk, akkor az összetett számok osztóit elég N/2-ig keresni.
- SŐT!
  A prímteszt felesleges. Ha növekvő sorrendben teszteljük az osztókat, akkor mindig prímeket találunk.

Hatékonyabb prímfelbontás

```
Be(N)
D := 2
Ciklus amíg N > 1
  Ciklus amíg D osztója N-nek
    N := N / D
    D kiírása a felbontásba
  Ciklus vége
  D := D + 1
Ciklus vége
```

felbont.java

```
class felbont{
   public static void main (String args[]) {
      long N = Long.parseLong(args[0]);

      long D = 2;
      while(N > 1){
         while(N % D == 0){
            System.out.println(D);
            N /= D;
         }
         D++;
      }
   }
}
```