MD5 ellenrz összeg készítése

Ritkán elforduló feladat egy fájl vagy egyéb stream tartalmából MD5 ellenrz összeget készíteni, ám viszonylag hamar meg lehet találni a jó megoldást (a try-catch blokkot mindenki képzelje a forráskód köré):

```
MessageDigest checksum;
checksum = MessageDigest.getInstance("MD5");
byte[] buf = new byte[1024];
FileInputStream fis = new FileInputStream(file);
while (true)
{
  int len = fis.read(buf, 0, buf.length);
  if (len <= 0)
  {
  break;
  }
  checksum.update(buf, 0, len);
}
byte[] checksumValue = checksum.digest();
```

Az eredménykép elálló *checksumValue* egyik apró problémája, hogy *byte* tömb, amelybl általában hexadecimális eredményt szeretnénk kapni. Több megoldás közül is választhatunk, az egyik legrövidebb:

```
BigInteger checksumBigValue = new BigInteger(1, checksumValue);
String.format("%032x", checksumBigValue);
```

Ebben az esetben a *BigInteger* azon konstruktorát használjuk, amely az eredményül kapott tömbbl egy *BigInteger* példányt készít, majd ebbl a *String.forma* metódusával elállítjuk a 32 karakter hosszú hexadecimális számot a bevezet nullákkal együtt. Ez egy szép, egyszer, kényelmes, ellenben lassú megoldás, 10 millió futásra (három mérésbl) 21,6 és 22,7 másodperc közötti értékek jöttek el (ha egy sorba szervezzük, akkor se lesz gyorsabb, 21,8 és 22,6 másodperc közötti eredmények születtek). Külön mérve a két sort kiderül, hogy a *BigInteger* viszonylag jó hatékonysággal készít példányt magából, mivel 0,38 és 0,42 másodperc közötti futásidket sikerült mérni, megállapítható, hogy a *String.format* nem túl gyors, mivelhogy általános célra szolgál.

A kérdés, hogy vannak-e gyorsabb megoldások, amelyek *BigInteger* példányból hexadecimális kimenetet készítenek. Az els tipp a *BigInteger.toString* metódusa, amely képes különféle számrendszerekre konvertálni, így hexadecimális számrendszerbe is:

```
String md5Sum = checksumBigValue.toString(16);
```

A megoldás viszonylag gyors, 10 millió futásra 12,3-13,1ms közötti értéket kapunk, amely bven alatta van a *byte* tömbbl való *BigInteger* készítésnek, amely 380-320ms ideig fut azonos körülmények között.

Ha nem figyelünk rendesen, akkor dolgunk végeztével nyugodtan hátradlünk, miközben a programunk futása hibás, mert *toString* metódus nem törödik a bevezet nullákkal, így szélsséges esetben az alábbi eredményt kapjuk:

```
0
```

Amely helyett 32 darab nullát kellene kapnunk:

```
00000000000000000000000000000000
```

A különbség komoly problémákat okozhat, ha két MD5 hexadecimális érték különbözségére alapozunk egy megoldást.

Van tehát egy gyors, ám hibás megoldásunk, amelyet kicsit csiszolnunk kell: ki kell egészíteni 32 karakterre a kapott hexadecimális eredményt.

Az egyik közkedvelt megoldás 1,70-1,72 másodperc közötti idket fut:

```
StringBuilder sb = new StringBuilder(64);
sb.append("00000000000000000000000000000000");
sb.append(md5Sum);
result = sb.substring(md5Sum.length());
```

Ezt egy kis trükkel (thx pgm) 1,56-1,61 másodpercre tudjuk leszorítani:

```
StringBuilder sb = new StringBuilder("00000000000000000000000000000000");
sb.setLength(32-md5Sum.length());
sb.append(md5Sum);
```

Ez utóbbi megoldással a teljes MD5 ellenrz összeg létrehozását leszorítottuk a 21-22 másodpercrl 1,86-1,92 másodpercre, amely több mint tízszeres javulás.

Van gyorsabb megoldás? 😑