```
//package hogehoge.com;

import java.io.PrintWriter;
import java.io.File;
import java.io.IOException;


public class S1GP {
//定数の定義
static final BigNumber minPrimeNum = new BigNumber("10000"); //最小の素数(5ケタ)
static final int minPrimeNumLength = 5; //最小の素数の桁数(5ケタ)
static int searchLimit = 1; //各桁ごとの探索回数
static final int checkPrimeMode = 1; //素数判定モード

//変数の定義
static int CompositeNumNLength; //Nの桁数
static int PrimeNumALength; //Aの桁数
static int PrimeNumBLength; //Bの桁数

static BigNumber PrimeNumA; //Aの候補
static BigNumber PrimeNumB; //Bの候補
static BigNumber CompositeNumN; //Nの候補
static BigNumber CompositeNumNMax; //Nの最大値
static BigNumber PrimeNumALast; //Aの確定値
static BigNumber PrimeNumBLast; //Bの確定値
static BigNumber CompositeNumNLast; //Nの確定値
//ファイル出力先
static String outfile = "output.txt";

//打ち切り判定用
static boolean isClose=false;
```

```
//package hogehoge.com;

import java.io.PrintWriter;
import java.io.File;
import java.io.IOException;
import java.math.BigInteger;

public class S1GP {
//定数の定義
static final BigInteger minPrimeNum = new BigInteger("10000"); //最小の素数(5ケタ)
static final int minPrimeNumLength = 5; //最小の素数の桁数(5ケタ)
static int searchLimit = 1; //各桁ごとの探索回数
static final int checkPrimeMode = 1; //素数判定モード

//変数の定義
static int CompositeNumNLength; //Nの桁数
static int PrimeNumALength; //Aの桁数
static int PrimeNumBLength; //Bの桁数

static BigInteger PrimeNumA; //Aの候補
static BigInteger PrimeNumB; //Bの候補
static BigInteger CompositeNumN; //Nの候補
static BigInteger CompositeNumNMax; //Nの最大値
static BigInteger PrimeNumALast; //Aの確定値
static BigInteger PrimeNumBLast; //Bの確定値
static BigInteger CompositeNumNLast; //Nの確定値



//打ち切り判定用
static boolean isClose=false;
```

```
static long startTime=-1;
static long maxTime=0;
static long limitTIme=50*1000;
static long previousTime=0;
static long nowTime=0;

public static void main(String[] args) {
// TODO Auto-generated method stub

//実行時間計測
long startTime = System.currentTimeMillis();

//初期化
CompositeNumNLast=BigNumber.ZERO;
PrimeNumALast =BigNumber.ZERO;
PrimeNumBLast=BigNumber.ZERO;

//入力値の取得
// java S1GP Yの桁数 打ち切り判定時間 探索レベル 厳密解判定する最大桁数
CompositeNumNLength = Integer.parseInt(args[0]);



//デフォルト値を設定
limitTIme=50*1000; //打ち切り判定時間(デフォルト50秒)
searchLimit = 1; //探索レベル (デフォルト1)
int useExactSol = 15; //厳密解判定する最大桁数 (デフォルト15)

if(args.length>=2) limitTIme = Integer.parseInt(args[1]) * 1000;
if(args.length>=3) searchLimit = Integer.parseInt(args[2]);
if(args.length>=4) useExactSol = Integer.parseInt(args[3]);
```

```
static long startTime=-1;
static long maxTime=0;
static long limitTIme=50*1000;
static long previousTime=0;
static long nowTime=0;

public static void main(String[] args) {
// TODO Auto-generated method stub

//実行時間計測
long startTime = System.currentTimeMillis();

//初期化
CompositeNumNLast=BigInteger.ZERO;
PrimeNumALast =BigInteger.ZERO;
PrimeNumBLast=BigInteger.ZERO;

//入力値の取得
// java S1GP Yの桁数 打ち切り判定時間 探索レベル 厳密解判定する最大桁数
CompositeNumNLength = Integer.parseInt(args[0]);

//ファイル出力先
String outfile = "output_"+ CompositeNumNLength + ".txt";

//デフォルト値を設定
limitTIme=50*1000; //打ち切り判定時間(デフォルト50秒)
searchLimit = 1; //探索レベル (デフォルト1)
int useExactSol = 15; //厳密解判定する最大桁数 (デフォルト15)

if(args.length>=2) limitTIme = Integer.parseInt(args[1]) * 1000;
if(args.length>=3) searchLimit = Integer.parseInt(args[2]);
if(args.length>=4) useExactSol = Integer.parseInt(args[3]);
```

```
//設定値の表示
System.out.println("CompositNum N Length = " + CompositeNumNLength );
System.out.println("Limit Time = " + limitTIme + "msec" );
System.out.println("Serach Level = " + searchLimit );
System.out.println("Use Exact Solution Method = " + useExactSol );

//厳密解を算出
if(CompositeNumNLength <= useExactSol){
long result[] = ExactSolution.exactSol(CompositeNumNLength);

//ファイル出力
PrintWriter pw = null;
try {
// 出力先ファイル
File file = new File(outfile);

pw = new PrintWriter(file);
pw.println("" + result[0] + ","
+ result[2] + ","
+ result[2]);

} catch (IOException e) {
e.printStackTrace();
} finally {
if (pw != null) {

// ストリームは必ず finally で close します。
pw.close();
}
}
//実行時間計測
```

```
//設定値の表示
System.out.println("CompositNum N Length = " + CompositeNumNLength );
System.out.println("Limit Time = " + limitTIme + "msec" );
System.out.println("Serach Level = " + searchLimit );

System.out.println("Use Exact Solution Method = " + useExactSol );

//厳密解を算出
if(CompositeNumNLength <= useExactSol){
long result[] = ExactSolution.exactSol(CompositeNumNLength);

//ファイル出力
PrintWriter pw = null;
try {
// 出力先ファイル
File file = new File(outfile);

pw = new PrintWriter(file);
pw.println("" + result[0] + ","
+ result[2] + ","
+ result[2]);

} catch (IOException e) {
e.printStackTrace();
} finally {
if (pw != null) {

// ストリームは必ず finally で close します。
pw.close();
}
}
//実行時間計測
```

```
long stopTime = System.currentTimeMillis();

//実行時間を出力
System.out.println(" Run Time = " + (stopTime - startTime) + " ms " );
return;
}

//最適解を算出
//BigNumberクラス用素数リストの作成
BigNumber.initPrimeList();

//素数探索用マスク作成
BigNumber.makePrimeMask();

//最初の桁数の決定
PrimeNumALength = minPrimeNumLength;
PrimeNumBLength = CompositeNumNLength - minPrimeNumLength +1;

//最大のNをセット
StringBuffer buf = new StringBuffer();
for (int i = 1; i <= CompositeNumNLength; i++) {
buf.append("9");
}
CompositeNumNMax = new BigNumber(buf.toString());
System.out.println("CompositeNumNMax = " + CompositeNumNMax );

//searchAB();

/// 離れた桁数→近い桁数  ex)100ケタなら5･95桁⇒･･･⇒49･51
//Bの桁数がminPrimeNumLength(=5ケタ)以上の間、A,Bを探索
if(CompositeNumNLength < 141){
■ while ( PrimeNumALength <= PrimeNumBLength) {
```

```
long stopTime = System.currentTimeMillis();

//実行時間を出力
System.out.println(" Run Time = " + (stopTime - startTime) + " ms " );
return;
}

//最適解を算出
//BigIntegerクラス用素数リストの作成
BigNumber.initPrimeList();

//素数探索用マスク作成
BigNumber.makePrimeMask();

//最初の桁数の決定
PrimeNumALength = minPrimeNumLength;
PrimeNumBLength = CompositeNumNLength - minPrimeNumLength +1;

//最大のNをセット
StringBuffer buf = new StringBuffer();
for (int i = 1; i <= CompositeNumNLength; i++) {
buf.append("9");
}
CompositeNumNMax = new BigInteger(buf.toString());
System.out.println("CompositeNumNMax = " + CompositeNumNMax );

//searchAB();

/// 離れた桁数→近い桁数  ex)100ケタなら5･95桁⇒･･･⇒49･51
//Bの桁数がminPrimeNumLength(=5ケタ)以上の間、A,Bを探索
while ( PrimeNumALength <= PrimeNumBLength) {
```

```
long temp1 = System.currentTimeMillis();
searchAB();
judgeClose();
if(isClose) break;
long temp2 = System.currentTimeMillis();
System.out.println(" Run Time = " + (temp2 - temp1) + " ms " );
PrimeNumALength++;
PrimeNumBLength--;
}
```

```
}else{
// 近い桁数→離れた桁数  ex)100ケタなら49･51桁⇒･･･⇒5･95
//Bの桁数がminPrimeNumLength(=5ケタ)以上の間、A,Bを探索
PrimeNumALength = (int)Math.ceil(CompositeNumNLength/2.0);
PrimeNumBLength = CompositeNumNLength/2;

while ( PrimeNumALength >= minPrimeNumLength) {
long temp1 = System.currentTimeMillis();
searchAB();
judgeClose();
if(isClose) break;
long temp2 = System.currentTimeMillis();
System.out.println(" Run Time = " + (temp2 - temp1) + " ms " );
PrimeNumALength--;
PrimeNumBLength++;
}
}
```

```
long temp1 = System.currentTimeMillis();
searchAB();
judgeClose();
if(isClose) break;
long temp2 = System.currentTimeMillis();
System.out.println(" Run Time = " + (temp2 - temp1) + " ms " );
PrimeNumALength++;
PrimeNumBLength--;
}

// 近い桁数→離れた桁数  ex)100ケタなら49･51桁⇒･･･⇒5･95
//Bの桁数がminPrimeNumLength(=5ケタ)以上の間、A,Bを探索
while ( PrimeNumALength >= minPrimeNumLength) {

//searchAB();

PrimeNumALength--;

PrimeNumBLength++;
}
```

```
//結果を出力
System.out.println("----------" );

PrintWriter pw = null;
try {
// 出力先ファイル
File file = new File(outfile);

pw = new PrintWriter(file);
pw.println("" + PrimeNumALast.getWordsRaw() + ","
+ PrimeNumBLast.getWordsRaw() + ","
+ CompositeNumNLast.getWordsRaw());

} catch (IOException e) {
e.printStackTrace();
} finally {
if (pw != null) {
// ストリームは必ず finally で close します。
pw.close();
}
}

//Nを出力
//CompositeNumN = PrimeNumA * PrimeNumB;
System.out.print("Decrease Approach CompositeNumN = " +
formatNumber(CompositeNumNLast) + " = " +
formatNumber(PrimeNumALast) + " * "+
formatNumber(PrimeNumBLast) +
" searchLimt = " + searchLimit + " checkPrimeMode=" +
checkPrimeMode);

//実行時間計測
long stopTime = System.currentTimeMillis();
```

```
//結果を出力
System.out.println("----------" );

PrintWriter pw = null;
try {
// 出力先ファイル
File file = new File(outfile);

pw = new PrintWriter(file);
pw.println("" + PrimeNumALast + ","
+ PrimeNumBLast + ","
+ CompositeNumNLast);

} catch (IOException e) {
e.printStackTrace();
} finally {
if (pw != null) {
// ストリームは必ず finally で close します。
pw.close();
}
}

//Nを出力
//CompositeNumN = PrimeNumA * PrimeNumB;
System.out.print("Decrease Approach CompositeNumN = " +
formatNumber(CompositeNumNLast) + " = " +
formatNumber(PrimeNumALast) + " * "+
formatNumber(PrimeNumBLast) +
" searchLimt = " + searchLimit + " checkPrimeMode=" +
checkPrimeMode);

//実行時間計測
long stopTime = System.currentTimeMillis();
```

```
//実行時間を出力
System.out.println(" Run Time = " + (stopTime - startTime) + " ms " );

}

private static void judgeClose(){
if(startTime <0)
startTime = System.currentTimeMillis();
nowTime = System.currentTimeMillis() - startTime;

//打ち切り判定
if(limitTIme - nowTime < maxTime * 3)
isClose =true;

//最大時間の更新
if( nowTime - previousTime > maxTime)
maxTime=nowTime - previousTime;

//今回時刻を記憶
previousTime = nowTime;
//次へ

}

private static void judgeExit(){
if(startTime <0)
startTime = System.currentTimeMillis();
nowTime = System.currentTimeMillis() - startTime;

//打ち切り判定
System.out.println("---- judge ------" + nowTime);

if(58000 < nowTime ){
//結果を出力
System.out.println("---- force stop ------" );
```


```
//実行時間を出力
System.out.println(" Run Time = " + (stopTime - startTime) + " ms " );

}

private static void judgeClose(){
if(startTime <0)
startTime = System.currentTimeMillis();
nowTime = System.currentTimeMillis() - startTime;

//打ち切り判定
if(limitTIme - nowTime < maxTime * 3)
isClose =true;

//最大時間の更新
if( nowTime - previousTime > maxTime)
maxTime=nowTime - previousTime;

//今回時刻を記憶
previousTime = nowTime;
//次へ

}

private static void judgeExit(){
if(startTime <0)
startTime = System.currentTimeMillis();
nowTime = System.currentTimeMillis() - startTime;

//打ち切り判定
System.out.println("---- judge ------" + nowTime);

if(58000 < nowTime ){
//結果を出力
System.out.println("---- force stop ------" );
```

```
PrintWriter pw = null;
try {
// 出力先ファイル
File file = new File(outfile);


pw = new PrintWriter(file);
pw.println("" + PrimeNumALast.getWordsRaw() + ","
+ PrimeNumBLast.getWordsRaw() + ","
+ CompositeNumNLast.getWordsRaw());

} catch (IOException e) {
e.printStackTrace();
} finally {
if (pw != null) {
// ストリームは必ず finally で close します。
pw.close();
}
}

//Nを出力
//CompositeNumN = PrimeNumA * PrimeNumB;
System.out.print("Decrease Approach CompositeNumN = " +
formatNumber(CompositeNumNLast) + " = " +
formatNumber(PrimeNumALast) + " * "+
formatNumber(PrimeNumBLast) +
" searchLimt = " + searchLimit + " checkPrimeMode=" +
checkPrimeMode);

//実行時間計測
long stopTime = System.currentTimeMillis();
```

```
PrintWriter pw = null;
try {
// 出力先ファイル
String outfile = "output_"+ CompositeNumNLength + ".txt";
    File file = new File(outfile);

pw = new PrintWriter(file);
pw.println("" + PrimeNumALast + ","
+ PrimeNumBLast + ","
+ CompositeNumNLast);

} catch (IOException e) {
e.printStackTrace();
} finally {
if (pw != null) {
// ストリームは必ず finally で close します。
pw.close();
}
}

//Nを出力
//CompositeNumN = PrimeNumA * PrimeNumB;
System.out.print("Decrease Approach CompositeNumN = " +
formatNumber(CompositeNumNLast) + " = " +
formatNumber(PrimeNumALast) + " * "+
formatNumber(PrimeNumBLast) +
" searchLimt = " + searchLimit + " checkPrimeMode=" +
checkPrimeMode);

//実行時間計測
long stopTime = System.currentTimeMillis();
```

```
//実行時間を出力
System.out.println(" Run Time = " + (stopTime - startTime) + " ms " );
System.exit(0);
}

}

private static void searchAB(){

System.out.println("----------" );
System.out.println("PrimeNum A Length = " + PrimeNumALength );
System.out.println("PrimeNum B Length = " + PrimeNumBLength );

//Aを探索する
//Aの初期値をセット
StringBuffer buf = new StringBuffer();
buf.append("1");
for (int i = 1; i < PrimeNumALength; i++) {
buf.append("0");
}
PrimeNumA = new BigNumber(buf.toString());

//A,Bを探索する
for(int j=1;j<=searchLimit;j++){

//System.out.println("PrimeNumA2(initial) = " + PrimeNumA );
PrimeNumA = nextPrimeNum(PrimeNumA, 0);

//System.out.println("PrimeNumA2 = " + PrimeNumA );

//Bを探索する
//A、Bの桁数が同じ／異なるケースで場合分け
/*if(PrimeNumALength==PrimeNumBLength){
PrimeNumB = PrimeNumA.add(BigNumber.ONE); //AとBの桁数が同
```

```
//実行時間を出力
System.out.println(" Run Time = " + (stopTime - startTime) + " ms " );
System.exit(0);
}

}

private static void searchAB(){

System.out.println("----------" );
System.out.println("PrimeNum A Length = " + PrimeNumALength );
System.out.println("PrimeNum B Length = " + PrimeNumBLength );

//Aを探索する
//Aの初期値をセット
StringBuffer buf = new StringBuffer();
buf.append("1");
for (int i = 1; i < PrimeNumALength; i++) {
buf.append("0");
}
PrimeNumA = new BigInteger(buf.toString());

//A,Bを探索する
for(int j=1;j<=searchLimit;j++){

//System.out.println("PrimeNumA2(initial) = " + PrimeNumA );
PrimeNumA = nextPrimeNum(PrimeNumA, 0);

//System.out.println("PrimeNumA2 = " + PrimeNumA );

//Bを探索する
//A、Bの桁数が同じ／異なるケースで場合分け
/*if(PrimeNumALength==PrimeNumBLength){
PrimeNumB = PrimeNumA.add(BigInteger.ONE); //AとBの桁数が同じ
```

```
じ 場合、(確定したA)-1 をセット
}else{
buf = new StringBuffer();
buf.append("1");
for (int i = 1; i < PrimeNumBLength; i++) {
buf.append("0");
}
PrimeNumB = new BigNumber(buf.toString()); //AとBの桁数が異なる場合、B^10-1 をセット
}
System.out.println("PrimeNumB(initial) = " + PrimeNumB );
*/
//B探索開始
//N桁となる最大のB2を求める
PrimeNumB = CompositeNumNMax.divide(PrimeNumA);
CompositeNumN = PrimeNumA.multiply(PrimeNumB);
//System.out.println("PrimeNumB(2nd candidate(max)) = " + PrimeNumB +
// " N = " + CompositeNumN );

//候補Nが現在のN以下か
if( CompositeNumN.compareTo(CompositeNumNLast) <=0){
//以下の場合
//B、Nを確定（何もしない）
//System.out.println("PrimeNumB(2nd candidate(max)) is no candidate");
//大きい場合
}else{
//1つ大きい素数を探す
PrimeNumB = previousPrimeNum(PrimeNumB, 0);
CompositeNumN = PrimeNumA.multiply(PrimeNumB);
```

```
場合、(確定したA)-1 をセット
}else{
buf = new StringBuffer();
buf.append("1");
for (int i = 1; i < PrimeNumBLength; i++) {
buf.append("0");
}
PrimeNumB = new BigInteger(buf.toString()); //AとBの桁数が異なる場合、B^10-1 をセット
}
System.out.println("PrimeNumB(initial) = " + PrimeNumB );
*/
//B探索開始
//N桁となる最大のB2を求める
PrimeNumB = CompositeNumNMax.divide(PrimeNumA);
CompositeNumN = PrimeNumA.multiply(PrimeNumB);
//System.out.println("PrimeNumB(2nd candidate(max)) = " + PrimeNumB +
// " N = " + CompositeNumN );

//候補Nが現在のN以下か
if( CompositeNumN.compareTo(CompositeNumNLast) <=0){
//以下の場合
//B、Nを確定（何もしない）
//System.out.println("PrimeNumB(2nd candidate(max)) is no candidate");
//大きい場合
}else{
//1つ大きい素数を探す
PrimeNumB = previousPrimeNum(PrimeNumB, 0);
CompositeNumN = PrimeNumA.multiply(PrimeNumB);
```

```
//候補が現在のN以上か
if(CompositeNumN.compareTo(CompositeNumNLast) >=0){
//B2をBとして確定
PrimeNumALast = PrimeNumA;
PrimeNumBLast = PrimeNumB;
CompositeNumNLast = CompositeNumN;
System.out.println("PrimeNumB(2nd candidate is optimum solution) = " + PrimeNumBLast +
" PrimeNumA2 = " + PrimeNumALast );
System.out.println(" N = " + CompositeNumNLast );
}else{
//System.out.println("PrimeNumB(2nd candidate is no optimum solution) = " + PrimeNumB +
// " N = " + CompositeNumN );
}
}

//次のA、Bを探す
//Aを1加算しておく
PrimeNumA = PrimeNumA.add(BigNumber.ONE);
}

}

private static String formatNumber(BigNumber convNum){
return convNum.toString();
/*String tempStr=convNum.toString();
int len=tempStr.length();
//System.out.println("String len " + convNum + " = " + len);

StringBuffer buf = new StringBuffer();
//buf.append("1");
for (int i = 0; i < len-1; i++) {
```

```
//候補が現在のN以上か
if(CompositeNumN.compareTo(CompositeNumNLast) >=0){
//B2をBとして確定
PrimeNumALast = PrimeNumA;
PrimeNumBLast = PrimeNumB;
CompositeNumNLast = CompositeNumN;
System.out.println("PrimeNumB(2nd candidate is optimum solution) = " + PrimeNumBLast +
" PrimeNumA2 = " + PrimeNumALast );
System.out.println(" N = " + CompositeNumNLast );
}else{
//System.out.println("PrimeNumB(2nd candidate is no optimum solution) = " + PrimeNumB +
// " N = " + CompositeNumN );
}
}

//次のA、Bを探す
//Aを1加算しておく
PrimeNumA = PrimeNumA.add(BigInteger.ONE);
}

}

private static String formatNumber(BigInteger convNum){
return convNum.toString();
/*String tempStr=convNum.toString();
int len=tempStr.length();
//System.out.println("String len " + convNum + " = " + len);

StringBuffer buf = new StringBuffer();
//buf.append("1");
for (int i = 0; i < len-1; i++) {
```

```
buf.append(tempStr.substring(i,i+1));
if((len-i-1)%5==0){
buf.append(" ");
}
}
buf.append(tempStr.substring(len-1,len));
buf.append("(" + len + ")");

return buf.toString();
*/
}

//次の素数を探索するメソッド
private static BigNumber nextPrimeNum(BigNumber startNumber,int mode){
System.out.println("NextP");
//素数マスクのチェック
int maskoffset = startNumber.remainder(BigNumber.maskLenBN).words[0];
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == BigNumber.maskLen-1 ? 0 : maskoffset+1 );
startNumber = startNumber.add(BigNumber.ONE);
}
//if(!BigNumber.primeMask[maskoffset]) return 0;

while(isPrimeNum(startNumber,checkPrimeMode)==0){
maskoffset = (maskoffset == BigNumber.maskLen-1 ? 0 : maskoffset+1 );
startNumber = startNumber.add(BigNumber.ONE);
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == BigNumber.maskLen-1 ? 0 :
```

```
buf.append(tempStr.substring(i,i+1));
if((len-i-1)%5==0){
buf.append(" ");
}
}
buf.append(tempStr.substring(len-1,len));
buf.append("(" + len + ")");

return buf.toString();
*/
}

//次の素数を探索するメソッド
private static BigInteger nextPrimeNum(BigInteger startNumber,int mode){
System.out.println("NextP");
//素数マスクのチェック
int maskoffset = startNumber.remainder(BigNumber.maskLenBN).intValue();
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == BigNumber.maskLen-1 ? 0 : maskoffset+1 );
startNumber = startNumber.add(BigInteger.ONE);
}
//if(!BigNumber.primeMask[maskoffset]) return 0;

while(isPrimeNum(startNumber,checkPrimeMode)==0){
maskoffset = (maskoffset == BigNumber.maskLen-1 ? 0 : maskoffset+1 );
startNumber = startNumber.add(BigInteger.ONE);
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == BigNumber.maskLen-1 ? 0 :
```

```
maskoffset+1 );
startNumber = startNumber.add(BigNumber.ONE);
}
//startNumber = startNumber.add(BigNumber.ONE);
//System.out.println("PrimeNumB(2nd check no candidate) = " +
startNumber );
//startNumber.add(startNumber);
}
//素数を返却
return startNumber;
}

//前の素数を探索するメソッド
private static BigNumber previousPrimeNum(BigNumber
startNumber,int mode){
System.out.println("PreviousP");
//素数マスクのチェック
int maskoffset =
startNumber.remainder(BigNumber.maskLenBN).words[0];
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == 0 ? BigNumber.maskLen-1 :
maskoffset-1 );
startNumber = startNumber.subtract(BigNumber.ONE);
}
while(isPrimeNum(startNumber,checkPrimeMode)==0){
maskoffset = (maskoffset == 0 ? BigNumber.maskLen-1 :
maskoffset-1 );
startNumber = startNumber.subtract(BigNumber.ONE);
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == 0 ? BigNumber.maskLen-1 :
maskoffset-1 );
```

```
maskoffset+1 );
startNumber = startNumber.add(BigInteger.ONE);
}
//startNumber = startNumber.add(BigNumber.ONE);
//System.out.println("PrimeNumB(2nd check no candidate) = " +
startNumber );
//startNumber.add(startNumber);
}
//素数を返却
return startNumber;
}

//前の素数を探索するメソッド
private static BigInteger previousPrimeNum(BigInteger
startNumber,int mode){
System.out.println("PreviousP");
//素数マスクのチェック
int maskoffset =
startNumber.remainder(BigNumber.maskLenBN).intValue();
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == 0 ? BigNumber.maskLen-1 :
maskoffset-1 );
startNumber = startNumber.subtract(BigInteger.ONE);
}
while(isPrimeNum(startNumber,checkPrimeMode)==0){
maskoffset = (maskoffset == 0 ? BigNumber.maskLen-1 :
maskoffset-1 );
startNumber = startNumber.subtract(BigInteger.ONE);
while(BigNumber.primeMask[maskoffset]){
maskoffset = (maskoffset == 0 ? BigNumber.maskLen-1 :
maskoffset-1 );
```

```
startNumber = startNumber.subtract(BigNumber.ONE);
}
//5桁以下になったら0を返す
if(startNumber.compareTo(minPrimeNum) <= 0){
return BigNumber.ZERO;
}
}
//素数を返却
return startNumber;
}
//素数かどうかを判断するメソッド
private static int isPrimeNum(BigNumber PrimeNumber,int mode){
System.out.println("isP : " + PrimeNumber);
judgeExit();
//mode判定  0=ためし割、1=isProbablePrime関数
if(mode==0){
//与えられた整数の平方根を探索の上限とする
//long rootPrimeNumber=PrimeNumber;
/* igNumber rootPrimeNumber = sqrt(PrimeNumber);
for ( BigNumber i = new BigNumber("2");
i.compareTo(rootPrimeNumber) < 0;i = i.add(BigNumber.ONE)) {
if (PrimeNumber.remainder(i) == BigNumber.ZERO) // 割り切れると
素数ではない
return 0; // それ以上の繰返しは不要
} */
// 最後まで割り切れなかった
return 1;
}else if(mode==1){
```

```
startNumber = startNumber.subtract(BigInteger.ONE);
}
//5桁以下になったら0を返す
if(startNumber.compareTo(minPrimeNum) <= 0){
return BigInteger.ZERO;
}
}
//素数を返却
return startNumber;
}
//素数かどうかを判断するメソッド
private static int isPrimeNum(BigInteger PrimeNumber,int mode){
System.out.println("isP : " + PrimeNumber);
judgeExit();
//mode判定  0=ためし割、1=isProbablePrime関数
if(mode==0){
//与えられた整数の平方根を探索の上限とする
//long rootPrimeNumber=PrimeNumber;
/* igNumber rootPrimeNumber = sqrt(PrimeNumber);
for ( BigInteger i = new BigInteger("2");
i.compareTo(rootPrimeNumber) < 0;i = i.add(BigInteger.ONE)) {
if (PrimeNumber.remainder(i) == BigInteger.ZERO) // 割り切れると
素数ではない
return 0; // それ以上の繰返しは不要
} */
// 最後まで割り切れなかった
return 1;
}else if(mode==1){
```

```
if(PrimeNumber.isProbablePrime(50)){
//System.out.println("true");
return 1;
}else{
//System.out.println("false");
return 0;
}
}else{
//modeが指定外
return -1;

}

}

//素数リストを作成
public static int[] sosuList(int max,int num){
//ふるいを作成(maxまでチェック)
boolean sieve[]=makeSieve(max);

//素数リスト作成
int list[]=new int[num];

//ゼロで初期化
for (int i=0; i<list.length ; i++){
list[i] = 0;
}

//素数のみセット
int count=0;

for (int i=2; i<sieve.length ; i++){
if(sieve[i]){
list[count] = i;
```

```
if(BigNumber.isProbablePrime(PrimeNumber,50)){
//System.out.println("true");
return 1;
}else{
//System.out.println("false");
return 0;
}
}else{
//modeが指定外
return -1;

}

}

//素数リストを作成
public static int[] sosuList(int max,int num){
//ふるいを作成(maxまでチェック)
boolean sieve[]=makeSieve(max);

//素数リスト作成
int list[]=new int[num];

//ゼロで初期化
for (int i=0; i<list.length ; i++){
list[i] = 0;
}

//素数のみセット
int count=0;

for (int i=2; i<sieve.length ; i++){
if(sieve[i]){
list[count] = i;
```

```
count++;
if(count>=num) break;
}
}
System.out.println("Last Prime Number = " + list[count-1]);
return list;
}

//素数のふるいを実施 (numまでのエラトスネテスのふるい)
private static boolean[] makeSieve(int num){
//trueで初期化
boolean sieve[]= new boolean[num+1];
for (int i=0; i<sieve.length ; i++){
sieve[i] = true;
}

//0と1は除外
sieve[0] = false;
sieve[1] = false;

int max = (int)Math.sqrt(sieve.length);

for (int p=2; p<=max ; p++){
if(sieve[p]){
for (int i=p*2; i<sieve.length ; i += p ){
sieve[i] = false;
}
}
}
return sieve;
}

}
```

```
count++;
if(count>=num) break;
}
}
System.out.println("Last Prime Number = " + list[count-1]);
return list;
}

//素数のふるいを実施 (numまでのエラトスネテスのふるい)
private static boolean[] makeSieve(int num){
//trueで初期化
boolean sieve[]= new boolean[num+1];
for (int i=0; i<sieve.length ; i++){
sieve[i] = true;
}

//0と1は除外
sieve[0] = false;
sieve[1] = false;

int max = (int)Math.sqrt(sieve.length);

for (int p=2; p<=max ; p++){
if(sieve[p]){
for (int i=p*2; i<sieve.length ; i += p ){
sieve[i] = false;
}
}
}
return sieve;
}

}
```

文字数: 10340
空白数: 3551 空白込み文字数: 13891
改行数: 503 改行込み文字数: 14394
単語数: 1307

文字数: 10120
空白数: 3453 空白込み文字数: 13573
改行数: 491 改行込み文字数: 14064
単語数: 1282

全体を表示 | カラー1 カラー2 モノクロ