Project 1: Factoring Algorithms

Olle Tervalampi-Olsson
Daniel Jankovic

November 17, 2015

1 Exercise 1

Assume 10^6 tests kan be performed on the following algorithm:

$$N mod p = 0 \tag{1}$$

for all p = 2,3,4,...,$\lfloor\sqrt{N}\rfloor$. If we have one 25-digit number we need to perform $\sqrt{10^{25}} \approx 10^{12}$ tests. This takes $\frac{10^{12}}{10^6} = 10^6$ seconds, which is approximately 11.5 days.

2 Exercise 2

Prime Number Theorem,

$$\pi(n) \approx \frac{n}{ln(n)} \tag{2}$$

gives an approximated solution to how many prime numbers will be trailing up to a certain number, n. With $n = 10^{12}$, the amount of primes will be $\approx 3.62 * 10^{10}$. Given this result the test will take $\frac{3.62*10^{10}}{10^6} = 3.62 * 10^4$ seconds, which is approximately 10 hours and 3 minutes.

Assume that the mean digit of the primes is six digits, and that each digit takes up 4 bits of space. Thus, we need $3.62^{10} * 4 * 6$ bits, which is ≈ 100 gigabytes. If we do not want to store all of that in memory at once, we want a pretty fast (solid state) disk to store them on. Looking at prisjakt.nu, we can find a 120 gigabyte SSD for 400 SEK, which is a pretty small budget and affordable for a student.

3 Exercise 3

We were given $N = 148042268393964514407317$. Our software found P and Q to be 428502845143 respectively 345487247219 with a total running time of < 1 minute. The program prints out the following:

```
Started clock
Generated primes in 0 seconds
Generated matrices in 39 seconds
Generated gauss in 5 seconds
Found the factors as P = 428502845143, Q = 345487247219
Found solution in 0 seconds
```

In total we spent approximately 9 hours with this project (code and report).

4 Exercise 4

```
Started clock
Generated primes in 0 seconds
Generated matrices in 489 seconds
Generated gauss in 4 seconds
Found the factors as P = 727123456789451, Q = 127123456789451
Found solution in 0 seconds
```

5 Code

```
import java.math.*;
import java.util.*;

public class PrimeGenerator {

    public static ArrayList<Integer>
      generatePrimeNumbers(long size) {
        double k;
        int j;
        ArrayList<Integer> primes = new ArrayList
          <Integer>();
        int i = 2;
        while (i < size) {
            k=Math.sqrt((double)i)+1.;
            for(j=0; j<primes.size() &&
              primes.get(j)<=k; ++j) {
                if(i%primes.get(j)==0) {
                    j=0;
                    break;
                }
            }
            if(j != 0  || primes.isEmpty())
                primes.add(i);
            i += 1;
        }
        return primes;
    }

    /**calculate the square root of a biginteger in
      logarithmic time*/
    public static BigInteger squareroot(BigInteger x)
      {
        BigInteger right = x, left = BigInteger.
          ZERO, mid;
        while(right.subtract(left).compareTo(
          BigInteger.ONE) > 0) {
            mid = (right.add(left)).
              shiftRight(1);
            if(mid.multiply(mid).compareTo(x)
              > 0)
                right = mid;
            else
                left = mid;
        }
        return left;
    }
}
```

```
import java.math.BigInteger;
import java.util.*;
import java.io.*;

public class QuadraticSieve {

  private final BigInteger N;
  private BigInteger P;
  private BigInteger Q;
  private final int L;
  private final int B;
  private ArrayList<Integer> F;
  private int[][] exponents;
  private BigInteger[] r;

  public QuadraticSieve(BigInteger N, int L, int B) {
    this.B = B;
    this.N = N;
    this.L = L;
    long start = System.currentTimeMillis();
    System.out.println("Started␣clock");
    F = PrimeGenerator.generatePrimeNumbers(B);
    System.out.println("Generated␣primes␣in␣" + (System.
        currentTimeMillis() - start)/1000 + "␣seconds");
    exponents = new int[L][F.size()];
    start = System.currentTimeMillis();
    findSmoothNumbers();
    System.out.println("Generated␣matrices␣in␣" + (System
        .currentTimeMillis() - start)/1000 + "␣seconds");
    start = System.currentTimeMillis();
    HashSet<BitSet> solutions = gauss();
    System.out.println("Generated␣gauss␣in␣" + (System.
        currentTimeMillis() - start)/1000 + "␣seconds");
    start = System.currentTimeMillis();
    for (BitSet b : solutions) {
      if (isSolution(b)) {
        System.out.println("Found␣solution␣in␣" + (System
            .currentTimeMillis() - start)/1000 + "␣seconds
            ");
        System.exit(0);
      }
    }
    System.out.println("No␣solution␣found␣=(");
  }

  public boolean isSolution(BitSet b) {
    BigInteger LHS = new BigInteger("1");
    BigInteger RHS = new BigInteger("1");
    int[] solExponents = new int[F.size()];
    for (int i = 0; i < b.length(); i++) {
```

```
    if (b.get(i)) {
      LHS = LHS.multiply(r[i]);
      for (int k = 0; k < F.size(); k++) {
        solExponents[k] += exponents[i][k];
      }
    }
  }
  for (int i = 0; i < solExponents.length; i++) {
    solExponents[i] /= 2;
    BigInteger val = BigInteger.valueOf(F.get(i));
    RHS = RHS.multiply(val.pow(solExponents[i]));
  }
  LHS = LHS.mod(N);
  RHS = RHS.mod(N);
  P = N.gcd(RHS.subtract(LHS));
  if (!P.equals(BigInteger.ONE)) {
    Q = N.divide(P);
    System.out.println("Found␣the␣factors␣as␣P␣=␣" + P
        + ",␣Q␣=␣" + Q);
    return true;
  }
  return false;
}

public void findSmoothNumbers() {
  int count = 0;
  int k = 1;
  int j;
  r = new BigInteger[L];
  BigInteger current, temp;;
  HashSet<BitSet> M = new HashSet<BitSet>();
  while(count < L) {
    k++;
    temp = PrimeGenerator.squareroot(N.multiply(new
        BigInteger(Integer.toString(k))));
    for(j = 2; j <= k; j++) {
      if (count >= L)
        break;
      current = temp.add(new BigInteger(Integer.
          toString(j)));
      int[] expVector = getExponentVector(current);
      if (expVector != null) {
        BitSet b = convertToBits(expVector);
        if (M.add(b)) {
          exponents[count] = expVector;
          r[count] = current;
          count += 1;
        }
      }
    }
  }

}

public BitSet convertToBits(int[] exps) {
  BitSet b = new BitSet(F.size());
  for (int i = 0; i < exps.length; i++) {
    if (exps[i] % 2 != 0)
      b.flip(i);
  }
  return b;
}

private int[] getExponentVector(BigInteger current) {
  BigInteger rSqr;
  int[] exponents = new int[F.size()];
  rSqr = current.pow(2).mod(N);
  int i=0;
  BigInteger temp;
  while (!rSqr.equals(BigInteger.ZERO) && !rSqr.equals(
      BigInteger.ONE)) {
    temp = new BigInteger(Integer.toString(F.get(i)));
    if (rSqr.mod(temp).equals(BigInteger.ZERO)) {
      exponents[i] += 1;
      rSqr = rSqr.divide(temp);
    } else {
      i += 1;
      if(i == F.size())
        return null;
    }
  }
  return exponents;
}

public HashSet<BitSet> gauss() {
  try {
    BufferedWriter writer = new BufferedWriter(new
        OutputStreamWriter(new FileOutputStream("input")
        ));
    writer.write(L + "␣" + F.size());
    writer.newLine();
    for(int i = 0; i < L; i++) {
      for(int j = 0; j < F.size(); j++) {
        writer.write(exponents[i][j] + "␣");
      }
      writer.newLine();
    }
    writer.close();
    Process p = Runtime.getRuntime().exec("./
        GaussSolver␣input␣output");
      p.waitFor();
      BufferedReader reader = new BufferedReader(new
          InputStreamReader(new FileInputStream(”output”))
          );
      int numSolutions = Integer.valueOf(reader.readLine
          ());
      HashSet<BitSet> possibleSolutions = new HashSet<
          BitSet>();
      String solution;
      BitSet b;
      for (int i = 0; i < numSolutions; i++) {
        b = new BitSet(F.size() + 1);
        solution = reader.readLine().replaceAll(”␣”, ””);
        for (int j = 0; j < solution.length(); j++) {
          if (solution.charAt(j) == ’1’)
            b.flip(j);
        }
        possibleSolutions.add(b);
      }
      reader.close();
      return possibleSolutions;
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException io) {
      io.printStackTrace();
    } catch (InterruptedException ie) {
      ie.printStackTrace();
    }
    return null;
  }

  public static void main(String[] args){
    //BigInteger N = new BigInteger
        (”148042268393964514407317”);
    BigInteger N = new BigInteger(”
        9243444733977001554854488 1401”);
    int L = 1000;
    int B = 8000;
    QuadraticSieve qs = new QuadraticSieve(N, L, B);
  }
}
```