Aufgabenblatt 2
IT-Security
Angewandte Informatik
WS 2018/19

Lernziele – 6 Punkte

- Generierung von Primzahlen
- Tests auf Primzahlen
- Fermat- und Euler-Test
- Miller-Rabin-Test

In diesem Aufgabenblatt werden neben den Algorithmen zur Primzahlbestimmung weitere Routinen zur Langzahlarithmetik realisiert. Diese Bibliothek wird also schrittweise aufgebaut.

Aufgaben

Basierend auf den Routinen des letzten Aufgabenblatts werden hier zufällig gewählte Primzahlen generiert. Dies basiert auf folgenden zu realisierenden Routinen:

- func BigInt getRandomOdd(nat size>0) – erzeugt eine zufällige size-lange (in Bits) BigInt-Zahl, die immer ungerade ist. Die Zahl muss in der size-Position (most significant bit) und in der niederwertigsten Position (least significant bit) eine 1 haben.

Diese generierten Zufallszahlen werden geprüft, ob sie Primzahlen sind. Dazu werden folgende Tests realisiert, die true liefern, wenn der Test bestanden wird, sonst false:

- func bool isPrimeFermat(BigInt number, base) – führt den Fermat-Test mit Probe base durch
- func bool isPrimeEuler(BigInt number, base) – führt den Euler-Test mit Probe base durch
- func bool isPrimeMR(BigInt number, base) – führt den Miller-Rabin-Test mit Probe base durch
- func bool isPrimeDiv(BigInt number, int[] bases) – führt eine Probedivision mit den Werten durch, die im Array bases vorgegeben sind; diese Werte sind Primzahlen. Wenn eine Division ohne Rest möglich ist, wird false geliefert, ansonsten true.

Basierend auf diesen Routinen werden die folgenden realisiert:

- func bool isPrimeFermat(BigInt number, BigInt[] bases) – führt den Fermat-Test mit den Proben durch, die im Array bases vorgegeben sind
- func bool isPrimeEuler(BigInt number, BigInt[] bases) – führt den Euler-Test mit den Proben durch, die im Array bases vorgegeben sind
- func bool isPrimeMR(BigInt number, BigInt[] bases) – führt den Miller-Rabin-Test mit den Proben durch, die im Array bases vorgegeben sind

Diese drei Routinen bekommen ein Array von BigInt-Zahlen als Proben geliefert. Nach [10] reicht es aus, dass als Proben alle Primzahlen bis 38 ausreichen, um alle Zahlen bis $3{,}1*10^{23}$ (78 bit) *deterministisch* zu prüfen. Daher sollten diese Primzahlen sowie weitere in dem Array vorhanden sein. Achten Sie darauf, dass es dort keine Dubletten gibt.

Zur Realisierung sind noch weitere Langzahl-Routinen erforderlich:

- func BigInt power(BigInt x, y>0) – Schnelle Exponentiation x^y
- func BigInt powerMod(BigInt x, y>0, m>0) – Exponentiation x^y mod m
- func BigInt square(BigInt x) – Schnelles Quadrieren von x
- func int eq(BigInt x, y) – liefert true, falls x = y ist
- func int zeroBits(BigInt x) - liefert die Anzahl der untersten Bits, die 0 sind, ohne dass eine 1 dazwischen ist.

Für die schnelle Exponentiation können Sie das arithmetische shiftLeft() benutzen; beachten Sie jedoch, dass das Vorzeichen expandiert wird, d.h. die Abfrage nach 0 ist nur dann richtig, wenn der Exponent positiv ist. Entsprechendes gilt auch für den Miller-Rabin- bzw. Euler-Test.

Der Miller-Rabin-Test läuft am besten, wenn die Zufallszahlen folgenden Aufbau haben: **...**100...001, mit * für 0 oder 1. Die Anzahl der *-Bits ergibt die "Stärke" der Primzahl, d.h. für die effektive Sicherheit sind nur die *-Bits von Bedeutung.

Das Quadrieren können Sie auf die Multiplikation zurückführen. Etwas schneller (40%) geht ein besserer Quadrieralgorithmus.

Auf der Website sind Dateien mit Zahlenmaterial für das Testen vorhanden. Diese Tests müssen durchgeführt werden.

Beantworten Sie empirisch folgende Fragen:

- Es werden die drei Tests (Fermat, Euler, Miller-Rabin) mit zufälligen Proben (und den Primzahlen bis 38 (siehe oben)) sowie vorher die Probedivision mit allen Primzahlen unter 100 durchgeführt. Dies wird ca. 100 Mal mit (a) echten Primzahlen, (b) Pseudoprimzahlen und (c) zusammengesetzten Zahlen durchgeführt. Bestimmen Sie in wie vielen Prozent der Fälle die jeweiligen Tests irren bzw. richtig liegen.
- Müssen tatsächlich 50 Runden beim Miller-Rabin-Test durchgeführt wer-

den? Lohnen sich die letzten 30 überhaupt? Bestimmen Sie bei ca. 100 Versuchen, in wie vielen Fällen das Feststellen einer Nicht-Primzahl in den Bereichen 1-19, 20-29, 30-39 und 40 bis 50 Runden erfolgt. Dazu realisieren Sie auch die Fälle (a) bis (c).

Bitte beachten Sie folgendes Prinzip: **Es kommt auf Korrektheit und nicht auf Performanz an**.

Links

1. https://de.wikipedia.org/wiki/Pseudoprimzahl
2. http://www.mathe-schule.de/download/pdf/Primzahl/PSP.pdf
3. https://de.wikipedia.org/wiki/Fermatsche_Pseudoprimzahl
4. https://de.wikibooks.org/wiki/Pseudoprimzahlen:_Die_fermatsche_Pseudoprimzahl_im_allgemeinen
5. http://www.mathepedia.de/Fermatsche_Pseudoprimzahlen.aspx
6. https://de.wikipedia.org/wiki/Eulersche_Pseudoprimzahl
7. https://de.wikipedia.org/wiki/Starke_Pseudoprimzahl
8. http://mathworld.wolfram.com/EulerPseudoprime.html
9. https://de.wikibooks.org/wiki/Pseudoprimzahlen:_Tabelle_Carmichael-Zahlen
10. https://de.wikipedia.org/wiki/Miller-Rabin-Test#Deterministische_-Varianten
11. https://de.wikibooks.org/wiki/Primzahlen:_Tabelle_der_Primzahlen_(2_-_100.000)
12. https://www.cryptool.org/de/ct1-downloads

Abnahme

Zur Abnahme des gehören folgende Dateien:

- Testfälle mit Reports,
- Source-Code und
- Projektdateien, z.B. nbproject oder make-Dateien etc.
- Beantwortung der beiden obigen Fragen

Alle Tests vom Dozenten müssen minimal benutzt werden; es können noch weitere Test durchgeführt werden. Die Vorführung der Lösung besteht in folgenden Ablauf: (1) Source-Code-Begutachtung, (2) Übersetzung und (3) Testlauf.

Wer mit vorgefertigten Langzahl-Bibliotheken arbeitet erhält 2 Punkte Abzug.

Auch diese Aufgabe kann per Email abgegeben werden, dann mit allen Sour-

cen, Übersetzungsdateien, z.B. make bzw. eclipse/netbeans-Projekte, einschließlich der Tests. Alles muss ohne weiteres auf dem Rechner des Dozenten laufen können (Java, C++, Linux, CentOS 6.10). Programmieren Sie also portabel.