Der Primzahltest

Die Schülerinnen und Schüler sollen dieses Skript selbstständig durcharbeiten. Die Aufträge und die dadurch entwickelten Algorithmen werden im Unterrichtsgespräch zwischen den einzelnen Arbeitsphasen besprochen.

Voraussetzungen

- Die SuS kennen die Modulo-Rechnung.
- Die SuS können Python-Code lesen und durchführen.

Einstieg

Primzahlen sind ein faszinierendes mathematisches Konzept, das eine wichtige Rolle in der modernen Technologie spielt. Wenn man versteht, wie Primzahlen funktionieren und wie sie verwendet werden, kann man ein tieferes Verständnis für Verschlüsselungstechnologien, Datensicherheit und Algorithmen entwickeln. In der Mathematik gibt es viele offene Fragen und Vermutungen im Zusammenhang mit Primzahlen. Zum Beispiel ist es noch immer unbekannt, ob es eine endliche Anzahl von Primzahlzwillingen (zwei Primzahlen, deren Abstand 2 ist) gibt. Doch als Erstes stellen wir uns die Frage, wie wir Primzahlen finden bzw. wie wir bestimmen können, ob eine (grosse) Zahl eine Primzahl ist.

629’572’916’353 = prim?

Lernziele

- Du kannst einfache, aber rechenaufwändige Algorithmen entwickeln, mit deren Hilfe eine bestimmte Zahl als Primzahl oder zusammengesetzte Zahl identifiziert werden kann.
- Du kennst einen randomisierten Primzahltest und dessen Vor- und Nachteile.
- Du kannst das Konzept der häufigen Zeugen und die Bedeutung der Amplifikation erklären.

Erste Versuche

Auftrag 1

Erkläre, was eine Primzahl ist, beziehungsweise wodurch sie definiert ist!

_____

_____

Auftrag 2

Notiere alle Primzahlen zwischen 0 und 25 und stoppe die Zeit, die du dafür brauchst. Vergleiche dein Resultat mit den Lösungen. Pro Fehler werden fünf Sekunden zusätzlich angerechnet.

Auftrag 3

Auftrag 2 ist dir vermutlich leichtgefallen. Notiere nun alle Primzahlen zwischen 25 und 50. Stoppe nochmals die Zeit.

Auftrag 4

Wie bist du bei der Suche nach den Primzahlen vorgegangen? Notiere deine Überlegungen oder anders ausgedrückt: Beschreibe deinen Algorithmus in eigenen Worten:

_____

_____

_____

Die Probedivision und das Sieb des Eratosthenes

Eine einfache Methode, um zu überprüfen, ob eine gegebene Zahl eine Primzahl ist oder nicht, ist die *Probedivision*. Dabei wird die Zahl *n* nacheinander durch alle Zahlen von 2 bis *n*-1 geteilt. Wenn die gegebene Zahl durch eine dieser Zahlen ohne Rest geteilt werden kann, dann ist sie keine Primzahl.

Diese Methode folgt der Definition der Primzahlen und kann bei kleineren Zahlen schnell durchgeführt werden. Bei grösseren Zahlen kann dieser Test aber zeitaufwändig sein und andere effektivere Methoden müssen eingesetzt werden.

Auftrag 5

Gesucht sind alle Primzahlen zwischen 1 und 200!
Deine Primzahlsuche ist bisher so verlaufen, dass du die Primzahlen direkt bestimmt hast. Eratosthenes, ein griechischer Gelehrter, ist bereits 3 Jahrhunderte v. Chr. bei der nach ihm benannten *Siebmethode* genau umgekehrt vorgegangen. Kommst du auf die Idee, wie er verfahren ist?
Überlege dir dabei eine Strategie, mit der du du nicht alle 200 Zahlen überprüfen musst.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170
171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190
191	192	193	194	195	196	197	198	199	200

Auftrag 6

Das Sieb des Eratosthenes ist eine Methode zur Suche von Primzahlen, bei der für jedes $a > 1$ dessen Vielfache aus der Liste gestrichen werden. Es gilt: n ist eine zusammengesetzte Zahl (d.h. keine Primzahl), wenn $n = a*b$ *für* $a, b \in \{2, \ldots, n-1\}$.

Ab welcher Zahl a konntest du bei Auftrag 5 keine weiteren Felder durchstreichen? Erkläre dein Vorgehen!

_____

_____

Naive Primzahltests in Python

Hier abgebildet sind mögliche Algorithmen für die Probedivision sowie das Sieb des Eratosthenes. Probiert beide Codes aus.

```python
import math

def probediv(n):
    if n <=1:
        return False
    else:
        for i in range(2, math.sqrt(n)+1):
            if n % i == 0:
                return False
        return True
```

```python
import math

def sieb(n):
    primes = []
    numbers = list(range(2,n))
    c = 2
    while c < math.sqrt(n)+1:
        for i in range(c,n,c):
            if i in numbers:
                numbers.remove(i)
        primes.append(c)
        c = numbers[0]
    return primes + numbers
```

Auftrag 7

Könnte man die zwei Algorithmen kombinieren? Beschreibe deine Überlegungen!

_____

_____

Erinnerst du dich an die blaue Abbildung mit der Frage zu Beginn unserer Lektion? Diese Frage kannst du nun beantworten:_____

Diese Methoden sind einfach und können bei kleineren Zahlen schnell durchgeführt werden. Bei grösseren Zahlen können diese naiven Tests aber äusserst zeitaufwändig sein und andere effektivere Methoden müssen verwendet werden. Zum Beispiel werden für aktuelle kryptographische Anwendungen Primzahlen mit über 100 Dezimalstellen verwendet und diese mittels Probedivision oder mit dem Sieb des Eratosthenes zu bestimmen, wäre hoffnungslos.

Randomisierter Primzahltest nach Fermat

Neben den behandelten deterministischen Primzahltests, die immer ein korrektes Ergebnis liefern, gibt es auch randomisierte Primzahltests, die viel schneller arbeiten, aber dafür nur mit einer bestimmten Wahrscheinlichkeit ein korrektes Ergebnis zurückgeben.

Einer dieser randomisierten Ansätze basiert auf *dem kleinen Satz von Fermat*, den wir mit den Erkenntnissen aus Auftrag 9 vervollständigen werden.

"Für jede Primzahl p und jedes $a \in \{1, 2, \ldots, p-1\}$ gilt: $a^{p-1} \pmod p = \_\_\_\_\_$ "

Auftrag 8

Gegeben ist der nebenstehende Algorithmus, der für eine Zahl p eine Liste der Ergebnisse für jedes $a \in \{1, 2, \ldots, p-1\}$ liefert. Führe den Algorithmus mit den Zahlen p = 6, 7, 8, 11 & 16 aus und notiere die Ergebnisse.

```
1 def fermatList(p):
2     results = []
3     for a in range(1,p):
4         results.append(a**(p-1) % p)
5     return results
```

	a									
p	1	2	3	4	5	6	7	8	9	10
2										
3										
4										
5										
6										
7										
8										
9										
11										

Was fällt dir auf und welche Schlüsse kannst du daraus ziehen?

_____

_____

Der kleine Satz von Fermat lässt uns darauf schliessen, dass eine Zahl *p* keine Primzahl ist, wenn wir mindestens ein $a \in \{1, 2, \ldots, p-1\}$ finden, für welches $a^{p-1} \pmod p$ nicht 1 ergibt. Bei den randomisierten Algorithmen beschreibt man ein solches *a* als **Zeugen**, welcher "bezeugt", dass *p* keine Primzahl ist. Selbstverständlich muss man nicht alle $a < p$ ausprobieren, sondern man kann bereits nach dem ersten Zeugen die Überprüfung beenden.

Wieso spricht man eigentlich von randomisierten Primzahltests? Man möchte gerne eine Zeugenart finden, die für jede Zahl *p* viele Zeugen hat und somit eine zufällige Wahl eines Zeugenkandidaten mit hoher Wahrscheinlichkeit einen Zeugen generiert.

Auftrag 9

Erstelle eine Funktion in Python, die eine Zahl *p* als Argument entgegennimmt, ein zufälliges $a \in \{1, 2, \ldots, p-1\}$ mittels random Modul bestimmt und nach der Überprüfung von $a^{p-1} \pmod p = 1$ False oder True zurückgibt.

Welche Schlüsse kannst du bei der Ausgabe von False bzw. True ziehen?

False:

True:

In der Tabelle von Auftrag 8 sehen wir, dass es z.B. für $p = 6$ vier mögliche Zeugen *a* gibt, die beweisen, dass $p \notin prim$ und ein *a*, bei welchem in der obigen Funktion True zurückgegeben würde. Da der Satz von Fermat bei $a = 1$ immer eine 1 ergibt, kann man sich auf $a \in \{2, \ldots, p-1\}$ beschränken und so die relative Häufigkeit der Zeugen für $p \notin prim$ steigern.

Um die Frage zu beantworten, wie viele Zeugen des Nichtprimzahlseins für eine zusammengesetzte Zahl *p* existieren, ergänzen wir den Algorithmus von Auftrag 8 mit einem Zähler für Anzahl Listenelemente und Anzahl Elemente ≠ 1.

```
def fermatList(p):
    results = []
    for a in range(2,p):
        results.append(a**(p-1) % p)
    return results

def anzahlElemente(p):
    return len(fermatList(p))

def anzahlZeugen(p):
    return anzahlElemente(p)-fermatList(p).count(1)

def ausgabe(p):
    print("Anzahl möglicher Zeugen: ", anzahlElemente(p))
    print("Tatsächliche Anzahl Zeugen: ", anzahlZeugen(p))
```

Auftrag 10

1. Bestimme für die folgenden zusammengesetzten Zahlen *p* die Anzahl möglicher Zeugen $a \in \{2, \ldots, p-1\}$.
2. Bestimme die Häufigkeit der Zeugen $a \in \{2, \ldots, p-1\}$, die zeigen, dass p ∉ prim.
3. Berechne die Wahrscheinlichkeit in %, dass dein Algorithmus von Auftrag 9 True zurückgibt, obwohl es sich offensichtlich nicht um eine Primzahl handelt.

p	Anzahl a ∈ {2, …, p-1}	Anzahl Zeugen p ∉ prim	Wahrsch. für Resultat p ∈ prim = True
8			
9			
25			
42			
69			
111			
522			

Anhand der hintersten Spalte wirst du erkannt haben, dass z.B. bei der zusammengesetzten Zahl 9 durchschnittlich bei ca. jeder siebten Ausführung des Primzahltests fälschlicherweise True zurückgegeben wird. Nehmen wir an, der Test wird wiederholt und als Ergebnis wird dieses Mal False ausgegeben. Für welche definitive Antwort entscheidest du dich?

Auftrag 11

Wie gross ist die Wahrscheinlichkeit, dass bei zweifacher Ausführung des Primzahltests zwei falsche Resultate ausgegeben werden? Ergänze die Tabelle von Auftrag 10 mit einer zusätzlichen Spalte und der neu errechneten Wahrscheinlichkeit für das Resultat p ∈ prim für die gegebenen zusammengesetzten Zahlen.

Amplifikation: Ein Experiment wird mehrfach wiederholt und die Wahrscheinlichkeit eines bestimmten Ergebnisses wird mit der Anzahl Wiederholungen exponentiell kleiner bzw. grösser. So kann auch die Wahrscheinlichkeit eines "falschen" Resultats mit linearem Aufwand exponentiell schnell gegen null verschoben werden.

Auftrag 12

Optimiere deinen randomisierten Primzahltest nach Fermat aus Auftrag 9, indem du angibst, wie oft der Test bei $a^{p-1} \pmod p = 1$ mit zufälligen $a \in \{2, \ldots, p-1\}$ wiederholt werden soll, damit du mit der Wahrscheinlichkeit eines korrekten Resultats zufrieden bist.

Auftrag 13 - freiwillig

Kombiniere die Algorithmen von Auftrag 10 und 12 so, dass beim Primzahltest der Zahl *p* folgendes Resultat ausgegeben wird:

"Die Zahl … ist eine/keine Primzahl. (Fehlerwahrscheinlichkeit: …%)"

Leider hat auch dieser Algorithmus zur Primzahlbestimmung eine Schwäche. Diese liegt darin, dass es sogenannte Carmichael-Zahlen (z.B. 561, 1105, 1729, …) gibt, die offensichtlich keine Primzahlen sind, aber dennoch die Fermat-Bedingung für alle Zahlen a mit ggT(a,n)=1 erfüllen. Somit gibt es zusammengesetzte Zahlen n, für die die Anzahl von Zeugen unzureichend gross ist.

.

Die Zeugenart kann man so verbessern, dass sie bei allen Nicht-Primzahlen funktioniert und man die Wahrscheinlichkeit, eine falsche Antwort zu bekommen, mit der Amplifikation in effizienter Weise beliebig nah zu null schrauben kann. Das ist heute die gängige Praxis bei der Generierung von grossen Primzahlen z.B. in der Kryptographie. Hinter der Bestimmung dieser Zeugenarten steckt jedoch eine sehr anspruchsvolle Mathematik, auf welche wir in diesem Rahmen nicht genauer eingehen werden.

Zusammenfassung / Fazit

Mit der Definition von Primzahlen im Kopf hast du intuitiv einen Algorithmus entworfen, um Primzahlen zu bestimmen bzw. um zu entscheiden, ob es sich bei einer bestimmten Zahl um eine Primzahl handelt oder nicht. Anschliessend hast du mit der *Probedivision* und *dem Sieb des Eratosthenes* zwei naive Algorithmen kennengelernt, die einfach zu verstehen, bei grossen Zahlen aber zu zeitintensiv sind. Beim randomisierten Primzahltest nach *dem kleinen Satz von Fermat* nutzt du die Methode der häufigen Zeugen, welche das Problem des hohen Rechenaufwandes löst. Durch mehrmaliges Bestimmen eines zufälligen Zeugens wird die Wahrscheinlichkeit eines korrekten Ergebnisses amplifiziert.
Der Primzahltest nach Fermat ist kein vollständig zuverlässiger Test für die Primzahlprüfung. Deshalb spricht man heute bei diesem Ansatz auch von einem Pseudoprimzahltest. Weitere Primzahltests auf Basis von randomisierten Algorithmen sind unter anderem der Solovay-Strassen-Test sowie der Miller-Rabin-Test, die mit der Wahrscheinlichkeit beliebig nah an 1 korrekt bestimmen, ob eine Zahl eine Primzahl ist oder nicht.

Bemerkung zu diesen Unterlagen

Dieses Skript ist für ca. vier Lektionen vorgesehen. Wenn das Thema noch vertiefter behandelt werden soll, kann auf weitere Primzahltests eingegangen werden, die jedoch fortgeschrittene mathematische Kenntnisse voraussetzen.

Lösungen

Auftrag 1

Eine Primzahl ist eine natürliche Zahl, die nur durch 1 und sich selbst teilbar ist.

Auftrag 2

Die Primzahlen von 1 bis 25 sind:2, 3, 5, 7, 11, 13, 17, 19, 23

Auftrag 3

Die Primzahlen von 26 bis 50 sind:: 29, 31, 37, 41, 43, 47

Auftrag 5

Die Lösung ist:

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~
101	~~102~~	103	~~104~~	~~105~~	~~106~~	107	~~108~~	109	~~110~~
~~111~~	~~112~~	113	~~114~~	~~115~~	~~116~~	~~117~~	~~118~~	~~119~~	~~120~~
~~121~~	~~122~~	~~123~~	~~124~~	~~125~~	~~126~~	127	~~128~~	~~129~~	~~130~~
131	~~132~~	~~133~~	~~134~~	~~135~~	~~136~~	137	~~138~~	139	~~140~~
~~141~~	~~142~~	~~143~~	~~144~~	~~145~~	~~146~~	~~147~~	~~148~~	149	~~150~~
151	~~152~~	~~153~~	~~154~~	~~155~~	~~156~~	157	~~158~~	~~159~~	~~160~~

Auftrag 6

Wenn eine Zahl *n* keine Primzahl ist, besteht sie aus mindestens zwei Faktoren *a* und *b.* Wenn a <= b, dann gilt auch, dass *a* <= $\sqrt{n}$. Für das Sieb bis 200 bedeutet das, dass nur die Vielfachen von *a* ∈ {2,.., $\sqrt{200}$} gestrichen werden müssen. $\sqrt{200}$ = 14.14 und da 14 bereits ein Vielfaches von 2 ist, gibt es nach *a* = 13 keine Streichungen mehr.

Auftrag 7

Wenn man bereits mit dem Sieb des Eratosthenes eine Liste *primes* bestimmt hat, die aus Primzahlen bis $\sqrt{n}$ besteht, muss man bei der Probedivision nicht durch $i \in \{2, ..., \sqrt{n}\}$ teilen, sondern nur durch $i \in$ primes <= $\sqrt{n}$.

Auftrag 8

	a									
p	1	2	3	4	5	6	7	8	9	10
2	1									
3	1	1								
4	1	0	3							
5	1	1	1	1						
6	1	2	3	4	5					
7	1	1	1	1	1	1				
8	1	0	3	0	5	0	7			
9	1	4	0	7	7	0	4	1		
11	1	1	1	1	1	1	1	1	1	1

Es ist ersichtlich, dass bei den Primzahlen für jedes $a \in \{1, 2, ..., p-1\}$ das Ergebnis 1 ergibt. Jedoch kann auch bei einigen zusammengesetzten Zahlen für zu viele *a* gelten: $a^{p-1} \pmod p = 1$ und somit wird es nicht hinreichend viele Zeugen der Zusammengesetztheit solcher Zahlen geben.

Auftrag 9

```python
import random

def fermat(p):
    a = random.randint(2, p-1)
    if a ** (p-1) % p != 1:
        return False
    else:
        return True
```

False: Wir haben einen Zeugen *a* gefunden, der zeigt, dass *p* keine Primzahl ist.
True: Es ist noch immer möglich, dass es sich bei *p* um eine zusammengesetzte Zahl oder eine Primzahl handelt.

Auftrag 10

Die Lösung ist:

p	Anzahl $a \in \{2, \ldots, p-1\}$	Anzahl Zeugen $p \notin$ prim	Wahrsch. für Resultat $p \in$ prim = True
8	6	6	0.0%
9	7	6	14.3%
25	23	20	13.0%
52	50	48	4.0%
69	67	64	4.5%
111	109	106	2.8%
964	962	960	0.2%

Auftrag 11

Mehrmaliges Durchführen des Tests. In Spalte 5 sind die Wahrscheinlichkeiten dargestellt, dass der Test für die konkrete zusammengesetzte Zahl *p* bei einem zufälligen *a* zweimal True und somit die falsche Antwort "Primzahl" zurückgibt.

p	Anzahl $a \in \{2, \ldots, p-1\}$	Anzahl Zeugen $p \notin$ prim	Wahrsch. für Resultat $p \in$ prim = True	Wahrsch. für **2x** Resultat $p \in$ prim = True
8	6	6	0.0%	0.0%
9	7	6	14.3%	1.96%
25	23	20	13.0%	1.70%
52	50	48	4.0%	0.16%
69	67	64	4.5%	0.20%
111	109	106	2.8%	0.0784%
964	962	960	0.2%	0.0004%

Auftrag 12

Wenn der Algorithmus einen Zeugen gefunden hat, ist bewiesen, dass die Zahl p zusammengesetzt ist und die entsprechende Antwort wird ausgegeben. Nur wenn alle Versuche, eine Zusammengesetztheit zu zeigen, gescheitert sind, wird die Antwort p ∈ prim = True angezeigt.

```python
import random

def multipleFermat(p, k):
    for i in range(k):
        a = random.randint(2, p)
        if a ** (p-1) % p != 1:
            return False
    return True
```

Auftrag 13

Die Kombination der Algorithmen aus den Aufträgen 10 und 12 gemäss Auftrag 13 sieht folgendermassen aus:

```python
import random

def fermatList(p):
    results = []
    for a in range(2,p):
        results.append(a**(p-1) % p)
    return results

def anzahlElemente(p):
    return len(fermatList(p))

def anzahlZeugen(p):
    return anzahlElemente(p)-fermatList(p).count(1)

def multipleFermat(p, k):
    for i in range(k):
        a = random.randint(2, p)
        if a ** (p-1) % p != 1:
            return False
    return True

def primzahltest(p,schleifen):
    isPrim = multipleFermat(p, schleifen)
    wahrscheinlichkeit = ((1-(anzahlZeugen(p) / anzahlElemente(p)))** schleifen)*100

    if isPrim == False:
        print("Die Zahl ", p, " ist keine Primzahl. (Fehlerwahrscheinlichkeit: 0%)")
    else:
        print("Die Zahl ", p, " ist eine Primzahl. (Fehlerwahrscheinlichkeit: ", wahrscheinlichkeit, "%)")
```