Primzahlen und Programmieren

Primzahlen

Wir wollen heute gemeinsam einen (sehr grundlegenden) Zusammenhang zwischen Programmieren und Mathematik herstellen. Die Zeiten in denen Mathematiker nur mit Zettel und Stift (oder Tafel und Kreide) gearbeitet haben, sind vergangen. Programmierung ist für weite Teile der Mathematik unabdingbar und wird auch in Bereichen verwendet, wo man es zunächst gar nicht vermuten würde. Im heutigen Workshop wollen wir uns näher mit Programmieren und Zahlentheorie auseinandersetzen.
Vom besonderen Interesse in der Zahlentheorie sind die **Primzahlen**. Ganz kurz zur Wiederholung. Was ist überhaupt eine Primzahl?

> Eine Primzahl p ist eine natürliche Zahlmit $p > 1$, die keine natürlichen Teiler außer 1 und p hat. (Zum Beispiel: $2, 3, 5, 7, \ldots$)

Wie ihr sicherlich schon gehört habt, werden Primzahlen (insbesondere sehr große Primzahlen) in der Verschlüsselung (*Kryptographie*) verwendet. Ziel des Workshops ist es das sog. *Sieb des Eratosthenes* zu implementieren und Anwendungen für komplexere (und vielleicht sogar ungelöste) Probleme zu entdecken. Als erstes wollen wir uns damit auseinandersetzen, wie man ein Programm erstellt, dass entscheiden kann, ob eine gegebene Zahl n eine Primzahl ist. Nun funktioniert ein PC (bzw. ein Programm) nicht wie ein Mensch, d.h. wir müssen die Überprüfung, ob eine Primzahl vorliegt, in Schritte umformulieren, die ein Programm umsetzen kann. Häufig trifft man Vorüberlegungen für ein Programm in **Pseudocode**. Um ein Gefühl dafür zu bekommen, was ein Programm umsetzen kann und wie Pseudocode aussieht, kann man sich die Arbeitsaufträge 1-8 angucken.
Sobald die Programmiergrundlagen erarbeitet sind, setzen wir uns mit einigen grundlegenden Fragen zu Primzahlen auseinander. Zunächst beweisen wir gemeinsam, dass es in der Tat unendliche viele Primzahlen gibt (ansonsten könnten wir sehr einfache Primzahltests implementieren). Beim Sieb des Eratosthenes beantworten wir die Frage, ob eine Primzahl vorliegt nicht nur für eine bestimmt Zahl, sondern erstellen eine ganze Liste mit Primzahlen. Diese können wir für mehrere Probleme nutzen: Zum einen können wir uns bis zu einer gewissen Zahl n davon überzeugen, dass Primzahlen mit Abstand 2 (Primzahlzwillinge), Primzahlen mit Abstand 4, Primzahlen mit Abstand 6 oder sogar 8 immer wieder auftreten. Um die Häufigkeit dieses Phänomens zu untersuchen, können wir wiederum Listen mit diesen Primzahlzwillingen erstellen. Bis heute ist nicht bewiesen, ob es tatsächlich unendlich viele Primzahlzwillinge gibt, obwohl die eisten Mathematiker fest davon überzeugt sind.

Primzahlen und Programmieren

Arbeitsauftrag 1 Was macht dieses Programm?

Algorithmus 1

Input: Ganze Zahlen $a, b, c \in \mathbb{P} \setminus \{2\}$
Output: Summe $s \in \mathbb{N}$
1: **Setze** $s = a + b + c$.
2: **Gebe** s aus.

Antwort:

Arbeitsauftrag 2 *If-Else* Bedingungen sind wichtige Elemente beim Programmieren. Hier ist ein Pseudocode gegeben. Fülle die Lücken im untenstehenden Programm, sodass der zum Pseudocode von Algorithmus 2 passt.

Algorithmus 2

Input: natürliche Zahl $n \in \mathbb{N}$
Output: ob n eine potenzielle Primzahl ist (d.h. nicht durch zwei teilbar und größer als zwei)
1: **Eingabe** einer natürlichen Zahl n .
2: **Prüfe** ob n zwei ist.
3: **Gebe aus**, ob n eine potenzielle Primzahl ist.
4: **Prüfe** ob n kleiner zwei ist.
5: **Gebe aus**, ob n eine potenzielle Primzahl ist.
6: **Sonst**:
7: **Prüfe** ob n von zwei geteilt wird.
8: **Gebe aus**, ob n eine potenzielle Primzahl ist.

```
1  #include<stdio.h>
2
3  int main(){
4    int n;
5    printf("Gebe eine Zahl ein, von der du wissen willst,  ob sie
         eine potenzielle Primzahl ist:  ");
6    scanf("%i",&n);
7    if(_______________){
8      printf("%i ist eine Primzahl\n.", n);
9    }
10   if (_______________){
11     printf("Es kann keine Primzahl kleiner zwei geben.\n");
12   }
13   else{
14     if(_____________){
```

Primzahlen und Programmieren

```
15        printf("%i kann keine Primzahl sein, da %i von zwei geteilt
              wird.\n", n, n);
16      }
17      else{
18        printf("%i ist eine potentielle Primzahl.\n", n);
19      }
20    }
21  return 0;
22  }
```

Arbeitsauftrag 3 Das folgende Programm sollte die Summe der ersten n Zahlen berechnen. Allerdings enthält es 5 Fehler. Finde sie alle.

```
1  #include<stdio.h>
2
3  int main(){
4    int n=10;    /*Addiere bis zu dieser Zahl*/
5    int i;
6    int summe;     /*speichert Zwischenergebnis*/
7    i=0;
8    while(i < n){
9      summe =+ i     /* addiere ite Zahl auf Summe*/
10   }
11   printf("Das Ergebnis ist %f.\n", summe);
12   return 0;
13 }
```

Arbeitsauftrag 4 Schreibe ein Programm im Pseudocode, welches für alle Zahlen $< n$ testet, ob diese Zahlen n teilen. (*Wenn du einen Code findest, der länger ist als sieben Zeilen, ist dies natürlich auch in Ordnung* ☺.) Implementiere den Pseudocode und teste folgende Zahlen auf Primalität: $131071, 524287, 10.561.327$, $2.147.483.645$ und $2.147.483.647$.

Algorithmus 3

Input: Ganze Zahl $n \in \mathbb{N}$
Output: `Primzahl` oder `keine Primzahl`

1:
2:
3:
4:
5:
6:
7:

Die Primzahlen die du hier gefunden hast, sind *Mersenne-Primzahlen*, das heißt es sind Zahlen der Form $2^n - 1$.

Primzahlen und Programmieren

Arbeitsauftrag 5 Schaffst du es, Algorithmus 3 effizienter zu gestalten? Ist es wirklich nötig die *While*-Schleife bis n laufen zu lassen? Ergänze den Pseudocode. Teste erneut 2.147.483.647 auf Primalität. Was fällt dir auf?

Algorithmus 4

Input: Ganze Zahl $n \in \mathbb{N}$
Output: `Primzahl` oder `keine Primzahl`

01: **Setze** $i = 2$.
02: **if** _____ **then**
03: Gebe `Primzahl` aus.
04: **return** 0
05: _____
06: **if** _____ **then**
07: Gebe `Keine Primzahl` aus.
08: _____
09: **end if**
10: **Setze** _____
11: **Gehe zu** 2.

Arbeitsauftrag 6 Implementiere Algorithmus 4 und finde alle Primzahlen zwischen 1 und 200. Trage Sie in die Tabelle ein.

Primzahlen und Programmieren

Arbeitsauftrag 7 Bevor wir uns nun der Implementierung des Siebes von Erathostenes widmen, gilt es im Umgang mit Arrays und Pointern sicherer zu werden. Wir haben im folgenden Code 7 Fehler gemacht. Finde sie alle.

```
1  #include <stdio.h>
2
3  void printarray (int zahlen[10]){
4    for(i=0; i < 10; i--) printf("%i", array[i]);
5    pritnf ("\n");
6  }
7
8  int main(){
9    int array[10];
10   int *i;
11   for (i=0; i<=10; i++){
12     array[i]=i;        /*schreibe 1 bis 10 ins Array*/
13   }
14   printarray (array);      /*gebe das array auss*/
15
16   return 0;
17 }
```

Arbeitsauftrag 8 Erstelle nun ein Programm, dass eine Liste mit den Zahlen von 0 bis n füllt und implementiere es für $n = 200$) (*Um dir das Array ausgeben zu lassen, kannst du die Funktion „printarray“* aus der vorherigen Aufgabe nutzen und leicht anpassen.)

Algorithmus 5

Input: Ganze Zahl $n \in \mathbb{N}$
Output: Liste mit den Zahlen von 0 bis n

01:
02:
03:
04:
05:
06:

Primzahlen und Programmieren

Wie viele Primzahlen gibt es eigentlich?

Sicherlich hast du festgestellt, dass diese recht einfach wirkenden Fragen zum Teil schon gar nicht so leicht zu beantworten sind. Viele Jahre zerbricht man sich schon den Kopf darüber, wie man möglichst geschickt überprüfen kann, wann eine Zahl prim ist bzw. wie man möglichst schnell möglichst viele Primzahlen finden kann. Doch wie viele Primzahlen gibt es überhaupt?

Mit dieser Frage setzte sich bereits Euklid auseinander. Er beanwortete die Frage wie folgt:
"Zu jeder vorgegebenen Menge Primzahlen, kann ich eine weitere Primzahl konstruieren, die noch nicht in der vorgegebenen Menge enthalten war."
Wir wollen uns überlegen, wie er das getan haben könnte.

Arbeitsauftrag 9

(1) 2 und 3 sind beides Primzahlen. Welche Teiler hat $(2 \cdot 3) + 1$?

(2) 2 und 11 sind Primzahlen. Welche Teiler hat $(2 \cdot 11) + 1$?

(3) 3 und 11 sind Primzahlen. Welche Teiler hat $(3 \cdot 11) + 1$?

(4) 2 und 3 und 11 sind Primzahlen. Welche Teiler hat $(3 \cdot 11 \cdot 2) + 1$?

(5) 3 und 5 und 13 sind Primzahlen. Welche Teiler hat $(3 \cdot 13 \cdot 5) + 1$

(6) Fällt dir etwas auf? Wie könnte Euklid aus einer Menge Primzahlen eine weitere kontruiert haben?

Primzahlen und Programmieren

Sieb des Eratosthenes

Nun, da wir festgestellt haben, dass es unendlich viele Primzahlen gibt, können wir uns die nächste Frage stellen: Gibt es eine Methode möglichst schnell zu erkennen, ob eine Zahl prim ist? Gibt es regelmäßige Abstände in denen sie auftreten? Wie häufig sind Primzahlen? Mit Sicherheit habt ihr schon festgestellt, dass es sehr mühselig ist, eine einzelne Zahl vorgegeben zu bekommen und zu überprüfen, ob diese prim ist. Tatsächlich sind bis heute keine "effizienten" Algorithmen bekannt, um das zu tun, d.h. selbst Computer brauchen sehr lange um sehr große Zahlen zu testen, ob sie prim sind oder nicht und, falls sie nicht prim sind, in ihre Primfaktoren zu zerlegen. Daher wäre es nützlich Listen mit Primzahlen zu erstellen um ggf. voraussagen zu können, wann die nächste Primzahl kommt. Wir wollen uns überlegen, wie man eine solche Liste erstellen könnte.

Arbeitsauftrag 10

(1) Wie viele Primzahlen gibt es, die ohne Rest durch 2 teilbar sind?

(2) Wie viele Primzahlen gibt es, die ohne Rest durch 3 oder 7 oder 11 teilbar sind?

(3) Angenommen eine Zahl ungleich 3, ist durch 3 ohne Rest teilbar. Kann sie dann noch prim sein?

(4) Angenommen eine Zahl ungleich 13, ist durch 13 ohne Rest teilbar. Kann sie dann noch prim sein?

(5) Auf der nächsten Seite findest du eine Tabelle mit allen Zahlen von 1 bis . Versuche alle Primzahlen mit System zu finden.

Primzahlen und Programmieren

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

Primzahlen und Programmieren

Arbeitsauftrag 11 Vergleiche nun die Primzahlen der Tabelle mit den Primzahlen von Arbeitsauftrag 4.

Arbeitsauftrag 12 Wir möchten nun das Sieb des Eratostenes zu implementieren. Da dies gar nicht so trivial ist, haben wir euch die Implementierung etwas vereinfacht.
Die grobe Idee des Programmes sollte sein:

- Wir haben eine Liste mit den Zahlen von 0 bis n.
- Wir fangen bei der zwei an und geben diese aus, da diese prim ist.
- Wir überprüfen die restlichen Einträge der Liste darauf, ob sie von der zwei geteilt werden. Wenn ein Wert von der zwei geteilt wird, überschreiben wir diesen mit einer 0, wenn nicht überprüfen wir den nächsten Eintrag in der Liste und fahren fort, bis wir die Liste komplett durchgegangen sind.
- Wir nehmen nun den nächsten Eintrag der Liste (mit dem Wert 3) und geben den Eintrag aus, sofern er ungleich null ist. Wir fahren fort wie in der ersten „Runde“ bis wir alle Zahlen getestet haben.

Pseudocode

Algorithmus 6

Input: Ganze Zahl $n \in \mathbb{N}$
Output: Liste mit den Zahlen von Primzahlen von 0 bis n

```
01: Setze j = 0.
02: Erstelle eine Liste mit den Zahlen von 0 bis n (siehe Algorithmus 5)
03: for (i = _____) do
04:     if (_____) do
05:         Gebe i aus.
06:         for (j = i to n/i) do
07:             _____
08:         endfor
09:     endif
10: endfor
```

Primzahlen und Programmieren

Code

```
1  /* sieve.C (Erathosthenes' Sieve)*/
2
3  #include<stdio.h>
4  #include<math.h>
5
6
7  void write_number(int n){              /* Diese Funktion gibt n
       aus. */
8    __________________;
9  }
10
11
12 void sieb(int n){
13   ____________                        /* Erstellen eine Liste mit n
         Eintraegen*/
14   int j=0;
15   while(_________){                   /*Fuelle Liste mit den Zahlen von
         0 bis n*/
16     _________;
17     _________;
18   }
19   int i;
20   for (___________________){
21     if (_________________){
22       ________________;
23       for (int j = i; j <= n/i; j++){     /*Wir gehen bis n:i,
             weil wir sonst aus dem array rausspringen wuerden bei p
             [(i*j)]*/
24          p[(i*j)]= ______;
25          }
26     }
27     }
28 }
29
30
31 int main(){
32   int n;
33   /*Dieses Programm berechnet die Primzahlen bis n*/
34   printf("Gebe die Zahl ein, bis zu der du die Primzahlen wissen
         willst: ");
35   scanf("%i",&n);
36   printf("Dieses Programm berechnet alle Primzahlen bis %i\n",n);
37     if (_____________){
38     printf("Es gibt keine Primzahlen kleiner 2.\n");
39     }
40     else{
41     printf("Die Primzahlen bis %i sind:\n", n);
42         sieb(n+1);
43     printf("\n.");
44     }
45     return 0;
46 }
```

Primzahlen und Programmieren

Arbeitsauftrag 13 Zum Abschluss möchten wir uns *Primzahlzwillingen* widmen. Ein Primzahlzwilling ist ein Paar von Primzahlen (p_1, p_2) für das gilt: $p_1 - p_2 = 2$. Wir wollen nun ein Programm schreiben, welches uns Primzahlzwillinge ausgibt. Wir nutzen als Grundlage Algorithmus 6 und erweitern diesen entsprechend. In Worten müsste das Programm wie folgt ergänzt werden.

- Wir müssen zuerst die Anzahl der Primzahlen bestimmen, und die Primzahlen in eine extra Liste einspeichern.
- Wir brauchen eine Funktion, die immer zwei aufeinanderfolgende Einträge der Liste mit dem Primzahlen auf ihre Differenz überprüft. Beträgt diese Differenz zwei, so sollen die entsprechenden Werte ausgegeben werden.

```
1  #include<stdio.h>
2  #include<math.h>
3
4  void vergleich (int *array, int laenge){
5    int index1=_____;
6    int index2=_____;
7    int diff=0;
8    while(index1<=laenge){
9      diff=____________________________;
10     if(_____________){
11       printf("Die Zahlen (%i, %i) sind Primzahlzwillinge.\n",
             array[index1], array[index2]);
12     }
13     index1++;
14     index2++;
15   }
16 }
17
18 void sieb(int n){
19   int p[n];                  /* Erstellen eine Liste der Laenge n
       */
20   int j=0;
21   while(j<=n){
22     p[j]=j;
23     j++;
24   }
25   int i;
26   int anzahlderprimzahlen=0;
27     for (i = 2; i < n; i++){
28     if (p[i]!=0){
29       anzahlderprimzahlen____;
30       for (int j = i; j <= n/i; j++){
31         p[(i * j)] = 0;
32             }
33     }
34     }
35   int liste[anzahlderprimzahlen];
36   int index=0;
37   for(i=2; i<n; i++ ){
```

Primzahlen und Programmieren

```
38      if(______________){
39        liste[index]=_____________;   /*Fuelle Liste mit
              Primzahlen */
40        index++;
41      }
42    }
43    vergleich(liste, anzahlderprimzahlen);
44  }
45
46
47
48  int main(){
49    int n;
50    /*Dieses Programm berechnet die Primzahlen bis 100*/
51    printf("Gebe die Zahl ein, bis zu der du die Primzahlzwillinge
          bestimmen willst: ");
52    scanf("%i",&n);
53    printf("Dieses Programm berechnet alle Primzahlzwillinge bis %i
          \n",n);
54      if (n < 2){
55      printf("Es gibt keine Primzahlzwillinge kleiner 2.\n");
56      }
57      else{
58      sieb(n+1);
59      printf("\n.");
60      }
61      return 0;
62  }
```