Eigenschaften des Fermat-Tests

Wir stellen fest, dass wenn n eine Primzahl ist, der Test keine Aussage macht. Denn dann gilt nach dem kleinen Satz von Fermat $a^{n-1} \equiv 1$ für alle wählbaren Werte von a.

Aber was passiert, wenn n eine *zusammengesetzte Zahl* ist?

Dann müssen wir zwei Fälle unterscheiden:

1) Für das gewählte a gilt *zufällig* auch $a^{n-1} \equiv 1$
2) Für das gewählte a gilt $a^{n-1} \not\equiv 1$

Leider können wir nicht *garantieren,* dass der zweite Fall, in dem der Fermat-Test das korrekte Ergebnis (n ist nicht prim) liefert, immer mit hoher Wahrscheinlichkeit eintritt.

Trotzdem liegt dieser Test allen gängigen Primzahltests zugrunde!

Durchführung des Fermat-Tests

Die Frage ist, wie man den Test *effizient* durchführen kann.

Wenn n eine 1000-stellige Binärzahl ist, müsste man mehr als 2^{1000} Multiplikationen durchführen!

(wenn man es naiv macht...)

Ein weiteres Problem besteht in der Größe der Zwischenergebnisse.

Allerdings ist die Lösung bzgl. der Zwischenergebnisse offensichtlich:

Da uns das Endergebnis ja nur *modulo* n interessiert, können wir von Anfang an *jedes* Zwischenergebnis modulo n reduzieren. Damit sind die Zwischenergebnisse immer kleiner als n.

Denn: $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$, und analog für Multiplikation.
Bitte überprüfen Sie das!

Schnelle Exponentiation

Zur Berechnung von a^{n-1} im Fermat-Test verwenden wir das folgende Programm zur Berechnung von a^b (bei Input a, b):

```
e := 1;
while b > 0 do
  if b ungerade then
      e := e · a mod n;
  a := a^2 mod n;
  b := ⌊b/2⌋;
return e
```

Als Schleifeninvariante notieren wir, dass der Wert von $e \cdot a^b$ zu Beginn und Ende jedes Durchlaufs gleich ist. (Natürlich alles immer modulo n !!!)

Nachrechnen!

Wenn wir die Inputwerte als a_0 und b_0 bezeichnen, hat die Schleifeninvariante anfangs den Wert $1 \cdot a_0^{b_0}$. Da es eine Invariante ist und am Ende $b = 0$ gilt, erhalten wir für den Ausgabewert e die Gleichung $e = e \cdot a^0 = 1 \cdot a_0^{b_0}$.

Also: Ausgabe korrekt, Laufzeit logarithmisch, Zwischenwerte klein...

Damit erfüllt der Algorithmus alle Anforderungen.

Graphen

Wir geben zunächst die allgemeinste Definition für den Begriff *Graph* an:

Definition: Ein Graph ist ein 4-Tupel (V, E, σ, τ), wobei V und E Mengen sind, und $\sigma : E \to V$ und $\tau : E \to V$ totale Abbildungen.

Im Rahmen dieser Vorlesung beschränken wir uns auf einfache ungerichtete Graphen, die wie folgt definiert werden können:

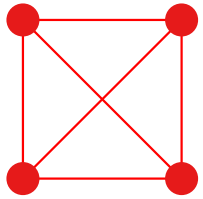
Definition: Ein einfacher ungerichteter Graph ist ein Paar (V, E), wobei V die Menge der Knoten und $E \subseteq \binom{V}{2}$ die Menge der Kanten ist.

Bitte schlagen Sie im Buch von Diekert et al. nach, um mehr über die verschiedenen Graphendefinitionen zu erfahren.

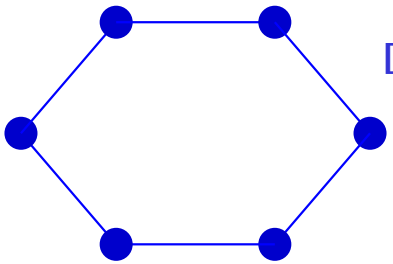
Graphische Darstellung von Graphen

Graphen können *graphisch dargestellt* werden, indem man die Elemente von V als Knoten und Elemente von E als Verbindungen der beiden zugehörigen Knoten zeichnet:

Dieser Graph heißt P_5. Die Verallgemeinerung auf P_n ist offensichtlich.



Das ist der K_4. Der K_n ist im allgemeinen der *vollständige* Graph mit n Knoten.



Dieser Graph heißt C_6.

Verallgemeinerung auf C_n naheliegend.

Als *Komplementgraph* zum Graph $G = (V, E)$ bezeichnen wir den Graph $G' = (V, \binom{V}{2} \setminus E)$. Als Beispiel für diese Begriffsbildung konstruieren wir die Komplementgraphen für die obigen Graphen.