Praktikum Computeralgebra: $\mathbb{Z}[1/N]$ in Sage

0. Ziel dieses Blattes ist es, die Einheiten im Ring $\mathbb{Z}[1/N]$ zu repräsentieren und für einige dieser Einheiten die Gleichung

$$u + v = 1$$

zu überprüfen. Analoge Funktionalität existiert bereits in Sage für die S-Einheiten in Zahlköpern $K \neq \mathbb{Q}$. Für $\mathbb{Q}$ selbst existiert diese Funktionalität kurioserweise noch nicht.

Legen Sie für dieses Blatt zwei Dateien an, `units.sage` und `test.sage`. Die erste Datei sollten Sie mit `%attach units.sage` in Ihre Sitzung einbinden; diese wird dann automatisch bei Änderungen neu geladen. Die zweite Datei lassen Sie mit `%runfile test.sage` bei Bedarf laufen.

1. Wie sie wissen, sind die Einheiten in $\mathbb{Z}[1/N]$ von der Form

$$(-1)^{\epsilon_0} p_1^{\epsilon_1} \cdots p_n^{\epsilon_n},$$

wobei die ϵ_i ganze Zahlen sind und die p_i die Primzahlen sind, die N teilen.

(a) Schreiben Sie zunächst eine Funktion `prime_factors(N)`, die zu einer Zahl N die Primzahlen liefert, die diese teilen. Vervollständigen Sie also den folgenden Ansatz:

```
def prime_factors(N):
  F = N.factor()
  S = []
  ...
  return S
```

(b) Vermutlich haben Sie in der vorigen Aufgabe manuell die Korrektheit Ihrer Funktion getestet. Um sicherzustellen, dass die Funktion auch Zukunft das richtige leistet, bietet es sich an Tests zu schreiben, die diesen Prozess automatisieren. Schreiben Sie hierzu in `test.sage` eine Funktion `test_prime_factors()` nach folgendem Muster:

```
import inspect

def test_prime_factors():
  name = inspect.stack()[0][3]
  tester = sage.misc.sage_unittest.InstanceTester(name)

  tester.assertListEqual(prime_factors(-18), [2,3])
  ...

  print "Test cases in %s pass."%name
test_prime_factors()
```

Ergänzen Sie dies mit einigen eigenen Beispielen und führen Sie `test.sage` aus.

2. Ziel dieser Aufgabe ist es, von einer rationalen Zahl x zu überprüfen, ob diese eine Einheit in $\mathbb{Z}[1/N]$ ist.

(a) Um dies zu prüfen, kann man eine Faktorisierung des Nenners von x betrachten. Es ist x eine Einheit genau dann, wenn in diesem Nenner nur Primfaktoren auftauchen, die auch N teilen. Schreiben Sie eine entsprechende Funktion `is_unit(x,S)`, wobei S eine Liste aus Primzahlen ist, die Primfaktoren von N.

```
def is_unit(x,S):
  if x == 0: return False
  d = x.denominator()
  F = prime_factors(d)
  ...
  return True
```

(b) Alternativ könnte man auch über die Primfaktoren von N iterieren und den Nenner von x so oft durch diese teilen, wie möglich. Es ist x eine Einheit genau dann, wenn der resultierende Nenner 1 ist. Schreiben Sie eine entsprechende Funktion `is_unit_(x,S)`, die diese Idee umsetzt. (Hinweis: Vermutlich werden Sie zwei Schleifen verwenden. Eine äußere `for`-Schleife, die über die Elemente von S iteriert und eine innere `while`-Schleife, die einen Primfaktor so oft austeilt, wie möglich. Um zu überprüfen, ob eine Zahl eine andere teilt, können Sie den Operator `%` verwenden.)

(c) Da wir nun über zwei konkurrierende Implementierungen verfügen, können wir prüfen, ob diese stets die gleiche Ausgabe liefern. Fügen Sie folgenden Test in `test.sage` ein.

```
def test_is_unit():
  name = inspect.stack()[0][3]
  tester = sage.misc.sage_unittest.InstanceTester(name)
  N = [1, 2, 6, 30]
  for n in N:
    S = prime_factors(n)
    X = [0,1,-1] + [1/i for i in range(-1000,1000) if i
       !=0]
    for x in X:
      tester.assertIs(is_unit(x,S),is_unit_(x,S))
  print "Test cases in %s pass."%name
test_is_unit()
```

3. Die Einheiten in $\mathbb{Z}[1/N]$ von der Form

$$(-1)^{\epsilon_0} p_1^{\epsilon_1} \cdots p_n^{\epsilon_n},$$

wobei die p_i die Primzahlen sind, die N teilen. Die Einheiten bilden also eine Gruppe, die isomorph ist zu

$$(\mathbb{Z}/2\mathbb{Z}) \times \mathbb{Z}^n.$$

(a) In Sage lassen sich solche Gruppen mit `AbelianGroupWithValues` realisieren. Versuchen Sie anhand der (verbesserungswürdigen[1]) Dokumentation von `AbelianGroupWithValues` die Einheitengruppe von $\mathbb{Z}[1/6]$ zu realisieren. Geben Sie den Erzeugern die Namen `pm`, `p2` und `p3`. Sie sollten also folgende Ausgaben beobachten: (Hinweis: Der Parameter `n` ist nicht optional. Er gibt die Anzahl der Erzeuger an.)

```
sage: G = AbelianGroupWithValues(...)
sage: G
Multiplicative Abelian group isomorphic to C2 x Z x Z
sage: G.gens()
(pm, p2, p3)
```

(b) Schreiben Sie eine Funktion `units(S)`, die zu einer Liste von Primzahlen `S` die entsprechende Gruppe aus der vorigen Teilaufgabe liefert.

[1] Wer schon immer mal seinen Namen in der Autorenliste eines freien Softwareprojekts wiederfinden wollte, kann gerne versuchen eine bessere Dokumentation zu schreiben...

(c) Schreiben Sie auch eine Funktion `test_units()`, um die Korrektheit Ihrer Implementierung zu prüfen. Prüfen Sie an selbst gewählten Beispielen, jeweils mit `tester.assertTupleEqual()`, dass `G.gens_orders()` und `G.variable_names()` die richtigen Werte liefern. Testen Sie, dass Ihre Implementierung den Grenzfall `S=[]` richtig behandelt.

4. Sei $G = \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ aus der vorigen Aufgabe. Elemente von `G` lassen sich durch Angabe der Exponenten erzeugen:

```
sage: G([1,2,3])
pm*p2^2*p3^3 # = -108
```

(a) Erzeugen Sie eine Liste, die alle Elemente von `G` enthält, deren Exponenten betragsmäßig kleiner oder gleich 10 sind. (Hinweis: Ihre Liste sollte also 882 Elemente enthalten.)

(b) Schreiben Sie eine Funktion `small_elements(n, G)`, die eine Liste aller Elemente von `G` liefert, deren Exponenten betragsmäßig durch `n` beschränkt sind. (Hinweis: Wenn Sie noch keine große Programmiererfahrung haben, wird diese Aufgabe möglicherweise unlösbar erscheinen. Fragen Sie, wenn Sie nicht wissen, wie Sie vorgehen sollen.)

(c) Schreiben Sie eine Funktion `test_small_element()`, die die Korrektheit prüft. Prüfen Sie also für einige selbst gewählt Beispiele, dass die Anzahl der Elemente, die von `small_elements()` zurückgegeben wird, korrekt ist.

5. Sei G wieder die Einheitengruppe von $\mathbb{Z}[1/N]$. Die Element von G lassen sich in Sage leicht in Elemente von $\mathbb{Q}$ konvertieren:[2]

```
sage: G = units([2,3])
sage: g = G([1,2,-3])
sage: QQ(g)
-4/27
```

(a) Implementieren Sie eine Funktion `to_unit(x, S)` die eine rationale Zahl `x` in ein Element von `G` konvertiert. (Hinweis: Verwenden Sie `is_unit()` um zu prüfen, ob `x` eine gültige Eingabe ist. Sollte dies nicht der Fall sein, können Sie mit `raise ValueError(¨x is not a unit¨)` einen Fehler erzeugen.)

(b) Schreiben Sie einen Funktion `test_to_unit()`, die für einige selbst gewählte Gruppen G mit `small_units()` eine Liste von Einheiten erzeugt und für diese dann überprüft, dass `QQ(to_unit(x,S))` und `x` übereinstimmen.

6. Finden Sie alle *kleinen* Lösungen der Gleichung

$$u + v = 1$$

in $\mathbb{Z}[1/6]$. (Hinweis: Beschränkt man die Exponenten wie oben durch 10, gibt es 21 Lösungen.)

[2] Erhalten Sie die Ausgabe `-4`, fehlt bei `AbelianGroupWithValues` der Parameter `values_group`.