Алгоритм Кронекера, полуавтоматиче реализация

Задача. Дан многочлен $f \in \mathbb{Z}[x]$, трубется выяснить, является ли он прост

Для решения этой задачи нужны несколько функций из комбинаторики, реа

1.) Для чего служит zip в коде Python?

B [14]:

```
L=zip(['a','b'],[1,2])
L
```

Out[14]:

```
<zip object at 0x6ffedc856410>
```

Эта функция возвращает "итератор", который можно превраить в обычный

B [15]:

```
list(L)
```

Out[15]:

```
[('a', 1), ('b', 2)]
```

Но лучше использовать функцию next. При каждом ее вызове будет возвра элемент L.

B [16]:

```
L=zip(['a','b'],[1,2])
next(L)
```

Out[16]:

```
('a', 1)
```

B [17]:

```
next(L)
```

Out[17]:

```
('b', 2)
```

В [18]:

```
1 next(L)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-18-26788c61f395> in <module>
----> 1 next(L)

StopIteration:
```

2.) Что делает функция Combinations, встроенная в Sage?

В [21]:

```
1 Combinations(range(5))
```

Out[21]:

```
Combinations of [0, 1, 2, 3, 4]
```

В [22]:

```
1 Combinations(range(5)).list()
```

Out[22]:

```
[[],
 [0],
 [1],
 [2],
 [3],
 [4],
 [0, 1],
 [0, 2],
 [0, 3],
 [0, 4],
 [1, 2],
 [1, 3],
 [1, 4],
 [2, 3],
 [2, 4],
 [3, 4],
 [0, 1, 2],
 [0, 1, 3],
 [0, 1, 4],
 [0, 2, 3],
 [0, 2, 4],
 [0, 3, 4],
 [1, 2, 3],
 [1, 2, 4],
 [1, 3, 4],
 [2, 3, 4],
 [0, 1, 2, 3],
 [0, 1, 2, 4],
 [0, 1, 3, 4],
 [0, 2, 3, 4],
 [1, 2, 3, 4],
 [0, 1, 2, 3, 4]]
```

В [25]:

```
1 Combinations(range(5),3).list()
```

Out[25]:

```
[[0, 1, 2],
 [0, 1, 3],
 [0, 1, 4],
 [0, 2, 3],
 [0, 2, 4],
 [0, 3, 4],
 [1, 2, 3],
 [1, 2, 4],
 [1, 3, 4],
 [2, 3, 4]]
```

Функция возвращает все возможные комбинации чисел из range, в которых Вторым парметром можно указать по сколько чисел будет в комбинациях.

3.) Что делает след. функция? Что на ее входе? Что на выходе?

B [26]:

```
def factors_p(n):
    ans=[]
    for (p,m)  in ZZ(n).factor():
        ans=ans+[p for mm in range(m)]
    return Combinations(ans)
```

B [34]:

```
factors_p(16)
```

Out[34]:

```
Combinations of [2, 2, 2, 2]
```

Функция возвращает список из максимального количества множителей из к
(исключая единицу, их можно написать бесконечно много). Эти множители м
комбинировать, в конце все равно получится заданное число.

B [35]:

```
[prod(a) for a in factors_p(16)]
```

Out[35]:

```
[1, 2, 4, 8, 16]
```

Замечание. Простой множитель повторяется столько раз, какова его кратно

4.) Что делает след. функция?

B [32]:

```
def factors(n):
    ans=[]
    for (p,m)  in ZZ(abs(n)).factor():
        ans=ans+[p for mm in range(m)]
    return [(-1)^k*prod(a) for a in Combinations(ans) for k
```

B [36]:

```
factors(16)
```

Out[36]:

```
[1, -1, 2, -2, 4, -4, 8, -8, 16, -16]
```

Выводит список всех возможных множителей, из которых можно составить

5.) Для чего служит функция product пакета itertools?

In [37]:

```
import itertools
L1 = [1,2,3,4]
L2 = ['a','b']
L3 = ['A','B', 'C']
list(itertools.product(L1,L2,L3))
```

Out[37]:

```
[(1, 'a', 'A'),
 (1, 'a', 'B'),
 (1, 'a', 'C'),
 (1, 'b', 'A'),
 (1, 'b', 'B'),
 (1, 'b', 'C'),
 (2, 'a', 'A'),
 (2, 'a', 'B'),
 (2, 'a', 'C'),
 (2, 'b', 'A'),
 (2, 'b', 'B'),
 (2, 'b', 'C'),
 (3, 'a', 'A'),
 (3, 'a', 'B'),
 (3, 'a', 'C'),
 (3, 'b', 'A'),
 (3, 'b', 'B'),
 (3, 'b', 'C'),
 (4, 'a', 'A'),
 (4, 'a', 'B'),
 (4, 'a', 'C'),
 (4, 'b', 'A'),
 (4, 'b', 'B'),
 (4, 'b', 'C')]
```

Замечание. Пакет itertools для Sage сторонний.

Эта функция возвращает "итератор", который можно превраить в обычный
использовать функцию next. При каждом ее вызове будет возвращаться сле

In [39]:

```
L=itertools.product(L1,L2,L3)
type(L)
```

Out[39]:

```
<class 'itertools.product'>
```

In [40]:

```
next(L)
```

Out[40]:

```
(1, 'a', 'A')
```

In [41]:

```
next(L)
```

Out[41]:

```
(1, 'a', 'B')
```

6.) Примените алгоритм Кронекера к многочлену $2x^3 - x^2 + 10x - 5$.

In [42]:

```
f=2*x^3 - x^2 + 10*x - 5
```

Если этот многочлен не является простым, то делится на многочлен g степ

$$f = gh.$$

При $x = 0$ значение $g(0)$ является фактором $f(0)$

In [43]:

```
L1=factors(f.subs(x=0))
L1
```

Out[43]:

```
[1, -1, 5, -5]
```

При $x = 1$ значение $g(1)$ является фактором $f(1)$

In [44]:

```
L2=factors(f.subs(x=1))
L2
```

Out[44]:

```
[1, -1, 2, -2, 3, -3, 6, -6]
```

Таким образом, g в точках $x = 0$ и $x = 1$ принимает одну из следующих па

В [45]:

```
1 list(itertools.product(L1,L2))
```

Out[45]:

```
[(1, 1),
 (1, -1),
 (1, 2),
 (1, -2),
 (1, 3),
 (1, -3),
 (1, 6),
 (1, -6),
 (-1, 1),
 (-1, -1),
 (-1, 2),
 (-1, -2),
 (-1, 3),
 (-1, -3),
 (-1, 6),
 (-1, -6),
 (5, 1),
 (5, -1),
 (5, 2),
 (5, -2),
 (5, 3),
 (5, -3),
 (5, 6),
 (5, -6),
 (-5, 1),
 (-5, -1),
 (-5, 2),
 (-5, -2),
 (-5, 3),
 (-5, -3),
 (-5, 6),
 (-5, -6)]
```

Для восстановления g воспользуемся функцией

В [46]:

```
1 def ipoly(points,x=x):
2     m=1
3     f=0
4     for (xx,yy) in points:
5         f=f+ (yy-f.subs([x==xx]))*m/m.subs([x==xx])
6         m=m*(x-xx)
7     return f
```

В [47]:

```
1 L=itertools.product(L1,L2)
2 L
```

Out[47]:

```
<itertools.product object at 0x6ffedc9a3eb0>
```

Запускаем след. код 2 раза, пока не получим делитель или не кончится спи

B [48]:

```
l=next(L)
g=ipoly(zip((0,1),l))
print(g)
QQ[x](f).quo_rem(QQ[x](g))
```

```
1
```

Out[48]:

```
(2*x^3 - x^2 + 10*x - 5, 0)
```

Вопрос. Зачем нам zip?

Ответ: многочлен не является простым, он делится на $1-2x$.

7.) Примените алгоритм Кронекера к многочлену $2x^5+x-1$.

B [49]:

```
f=2*x^5 + x - 1
```

Если этот многочлен не является простым, то делится на многочлен g степ

$$f=gh.$$

При $x=0$ значение $g(0)$ является фактором $f(0)$

B [50]:

```
L1=factors(f.subs(x=0))
L1
```

Out[50]:

```
[1, -1]
```

При $x=1$ значение $g(1)$ является фактором $f(1)$

B [51]:

```
L2=factors(f.subs(x=1))
L2
```

Out[51]:

```
[1, -1, 2, -2]
```

При $x=2$ значение $g(2)$ является фактором $f(2)$.

В [38]:

```
L3=factors(f.subs(x=2))
L3
```

Out[38]:

```
[1, -1, 5, -5, 13, -13, 65, -65]
```

Таким образом, g в точках $x = 0$, $x = 1$ и $x = 2$ принимает одну из следую

В [39]:

```
itertools.product(L1,L2,L3)
```

Out[39]:

```
<itertools.product object at 0x6ffed76196e0>
```

Список большой и перебирать его руками не удобно.

В [41]:

```
L=itertools.product(L1,L2,L3)
r=1
while r!=0:
    l=next(L)
    g=ipoly(zip((0,1,2),l))
    if QQ[x](g).degree()>0 and QQ[x](g).degree()<QQ[x](f).de
        print(g)
        [u,r]=QQ[x](f).quo_rem(QQ[x](g))
```

```
-(x - 1)*x + 1
2*(x - 1)*x + 1
-3*(x - 1)*x + 1
6*(x - 1)*x + 1
-7*(x - 1)*x + 1
32*(x - 1)*x + 1
-33*(x - 1)*x + 1
2*(x - 1)*x - 2*x + 1
(x - 1)*x - 2*x + 1
4*(x - 1)*x - 2*x + 1
-(x - 1)*x - 2*x + 1
8*(x - 1)*x - 2*x + 1
-5*(x - 1)*x - 2*x + 1
34*(x - 1)*x - 2*x + 1
-31*(x - 1)*x - 2*x + 1
-(x - 1)*x + x + 1
-2*(x - 1)*x + x + 1
(x - 1)*x + x + 1
-4*(x - 1)*x + x + 1
5*(x - 1)*x + x + 1
-8*(x - 1)*x + x + 1
31*(x - 1)*x + x + 1
-34*(x - 1)*x + x + 1
3*(x - 1)*x - 3*x + 1
2*(x - 1)*x - 3*x + 1
5*(x - 1)*x - 3*x + 1
-3*x + 1
9*(x - 1)*x - 3*x + 1
-4*(x - 1)*x - 3*x + 1
35*(x - 1)*x - 3*x + 1
-30*(x - 1)*x - 3*x + 1
-(x - 1)*x + 2*x - 1
-2*(x - 1)*x + 2*x - 1
(x - 1)*x + 2*x - 1
-4*(x - 1)*x + 2*x - 1
5*(x - 1)*x + 2*x - 1
-8*(x - 1)*x + 2*x - 1
31*(x - 1)*x + 2*x - 1
-34*(x - 1)*x + 2*x - 1
(x - 1)*x - 1
3*(x - 1)*x - 1
-2*(x - 1)*x - 1
7*(x - 1)*x - 1
-6*(x - 1)*x - 1
33*(x - 1)*x - 1
-32*(x - 1)*x - 1
-2*(x - 1)*x + 3*x - 1
-3*(x - 1)*x + 3*x - 1
3*x - 1
-5*(x - 1)*x + 3*x - 1
4*(x - 1)*x + 3*x - 1
-9*(x - 1)*x + 3*x - 1
30*(x - 1)*x + 3*x - 1
-35*(x - 1)*x + 3*x - 1
2*(x - 1)*x - x - 1
(x - 1)*x - x - 1
4*(x - 1)*x - x - 1
-(x - 1)*x - x - 1
8*(x - 1)*x - x - 1
-5*(x - 1)*x - x - 1
```

```
34*(x - 1)*x - x - 1
-31*(x - 1)*x - x - 1
-----------------------------------------------------------------
-
StopIteration                             Traceback (most recent
t)
<ipython-input-41-fae2d9a2f2c1> in <module>
      2 r=Integer(1)
      3 while r!=Integer(0):
----> 4     l=next(L)
      5     g=ipoly(zip((Integer(0),Integer(1),Integer(2)),l))
      6     if QQ[x](g).degree()>Integer(0) and QQ[x](g).degree(
(f).degree()/Integer(2):

StopIteration:
```

Сообщение об ошибке (StopIteration) означает, что мы перебрали все вариа
нашли.

8.) Как обработать ошибку StopIteration?

B [ ]:

```
L=itertools.product(L1,L2,L3)
r=1
while r!=0:
    try:
        l=next(L)
        g=ipoly(zip((0,1,2),l))
        if QQ[x](g).degree()>0 and QQ[x](g).degree()<QQ[x](f
            print(g)
            [u,r]=QQ[x](f).quo_rem(QQ[x](g))
    except StopIteration as err:
        print('poly is prime')
        break
```

Вопрос. Что будет, если убрать break?

Будет бесконечный цикл

B [ ]:

```
1
```