

Contents

1	Classes	2
1.1	poly.hensel – ヘンゼルリフト	2
1.1.1	HenselLiftPair – ヘンゼルリフトの組	3
1.1.1.1	lift – 一段階引き上げる	4
1.1.1.2	lift_factors – a1 と a2 を引き上げる	4
1.1.1.3	lift_ladder – u1 と u2 を引き上げる	4
1.1.2	HenselLiftMulti – 複数次多項式に対するヘンゼルリフト	4
1.1.2.1	lift – 一段階引き上げる	6
1.1.2.2	lift_factors – 因数を引き上げる	6
1.1.2.3	lift_ladder – u1 と u2 を引き上げる	6
1.1.3	HenselLiftSimultaneously	7
1.1.3.1	lift – 一段階引き上げる	8
1.1.3.2	first_lift – 最初のステップ	8
1.1.3.3	general_lift – 次のステップ	8
1.1.4	lift_upto – main 関数	8

Chapter 1

Classes

1.1 poly.hensel – ヘンゼルリフト

- **Classes**
 - †**HenselLiftPair**
 - †**HenselLiftMulti**
 - †**HenselLiftSimultaneously**
- **Functions**
 - **lift_upto**

このモジュールドキュメント内では,*polynomial* は整数係数多項式を意味.

1.1.1 HenselLiftPair – ヘンゼルリフトの組

Initialize (Constructor)

```
HenselLiftPair(f: polynomial, a1: polynomial, a2: polynomial, u1: polynomial, u2: polynomial, p: integer, q: integer=p)
→ HenselLiftPair
```

このオブジェクトはヘンゼルの補題によって引き上げられていく整数係数多項式を保存.

引数は以下の前提条件を満たさなければならない:

- $f, a1$ そして $a2$ はモニック多項式
- $f == a1 * a2 \pmod{q}$
- $a1 * u1 + a2 * u2 == 1 \pmod{p}$
- p は q を割り切り, どちらも自然数

```
from_factors(f: polynomial, a1: polynomial, a2: polynomial, p: integer)
→ HenselLiftPair
```

これは `HenselLiftPair` のインスタンスを作成し返すクラスメソッド. 初期構成のために $u1$ と $u2$ を計算し直す必要はない; これらは他の引数から用意される.

引数は以下の前提条件を満たすべきである:

- $f, a1$ と $a2$ はモニック多項式
- $f == a1 * a2 \pmod{p}$
- p は素数

Attributes

`point` :
リストとしての因数 $a1, a2$.

Methods

1.1.1.1 lift – 一段階引き上げる

`lift(self) →`

いわゆる二次方程式法により多項式を引き上げる.

1.1.1.2 lift_factors – a_1 と a_2 を引き上げる

`lift_factors(self) →`

整数係数多項式 A_i たちを引き上げることににより因数を更新:

- $f == A_1 * A_2 \pmod{p * q}$
- $A_i == a_i \pmod{q} \ (i = 1, 2)$

さらに, q は $p * q$ に更新される.

† 次の前提条件は自動的に満たされる:

- $f == a_1 * a_2 \pmod{q}$
- $a_1 * u_1 + a_2 * u_2 == 1 \pmod{p}$
- p は q を割り切る

1.1.1.3 lift_ladder – u_1 と u_2 を引き上げる

`lift_ladder(self) →`

u_1 と u_2 を U_1 と U_2 に更新:

- $a_1 * U_1 + a_2 * U_2 == 1 \pmod{p^{**2}}$
- $U_i == u_i \pmod{p} \ (i = 1, 2)$

そして, p を p^{**2} に更新.

† 次の前提条件は自動的に満たされる:

- $a_1 * u_1 + a_2 * u_2 == 1 \pmod{p}$

1.1.2 HenselLiftMulti – 複数次多項式に対するヘンゼルリフト

Initialize (Constructor)

`HenselLiftMulti(f: polynomial, factors: list, ladder: tuple, p: integer, q: integer=p) → HenselLiftMulti`

このオブジェクトはヘンゼルの補題によって引き上げられていく整数係数多項式の因数を保存. もし因数の数が二つなら, **HenselLiftPair** を使うべきである.

`factors` は多項式のリスト; これらの多項式は二つのリスト `sis` と `tis` のタプルである `a1, a2, ... ladder` として表し, 両リストは多項式から成る. `s1, s2, ...` として `sis` の多項式を表し, `t1, t2, ...` として `tis` の多項式を表す. さらに, b_i を $i < j$ である a_j たちの積として定義. 引数は以下の前提条件を満たす:

- f と全ての `factors` はモニック多項式
- $f == a_1 * \dots * a_r \pmod{q}$
- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, 2, \dots, r)$
- p は q を割り切り, どちらも自然数

```
from _factors(f: polynomial, factors: list, p: integer)
    → HenselLiftMulti
```

これは `HenselLiftMulti` のインスタンスを作成し返すためのクラスメソッド. 初期構成のために `ladder` を計算し直す必要はない; これらは他の引数によって用意される.

引数は前提条件を満たすべきである:

- f と全ての `factors` はモニック多項式
- $f == a_1 * \dots * a_r \pmod{q}$
- p は素数

Attributes

`point` :

リストとしての因数 a_i たち.

Methods

1.1.2.1 lift – 一段階引き上げる

`lift(self) →`

いわゆる二次方程式法により多項式を引き上げる.

1.1.2.2 lift_factors – 因数を引き上げる

`lift_factors(self) →`

整数係数多項式 A_i たちを引き上げることににより因数を更新:

- $f == A_1 \dots A_r \pmod{p * q}$
- $A_i == a_i \pmod{q} \ (i = 1, \dots, r)$

さらに, q は $p * q$ に更新.

† 次の前提条件は自動的に満たされる:

- $f == a_1 \dots a_r \pmod{q}$
- $a_i s_i + b_i t_i == 1 \pmod{p} \ (i = 1, \dots, r)$
- p は q を割り切る

1.1.2.3 lift_ladder – u_1 と u_2 を引き上げる

`lift_ladder(self) →`

s_i たちと t_i たちを S_i たちと T_i たちに更新:

- $a_1 S_i + b_i T_i == 1 \pmod{p^{**2}}$
- $S_i == s_i \pmod{p} \ (i = 1, \dots, r)$
- $T_i == t_i \pmod{p} \ (i = 1, \dots, r)$

そして, p を p^{**2} に更新.

† 次の前提条件は自動的に満たされる:

- $a_i s_i + b_i t_i == 1 \pmod{p} \ (i = 1, \dots, r)$

1.1.3 HenselLiftSimultaneously

このメソッドは [1] で説明されている.

† 以下の不変式を保存:

- a_i たち, p_i と g_i たちはすべてモニック多項式
- $f == g_1 * \dots * g_r \pmod{p}$
- $f == d_0 + d_1 * p + d_2 * p^2 + \dots + d_k * p^k$
- $h_i == g_{(i+1)} * \dots * g_r$
- $1 == g_i * s_i + h_i * t_i \pmod{p} \quad (i = 1, \dots, r)$
- $\deg(s_i) < \deg(h_i), \deg(t_i) < \deg(g_i) \quad (i = 1, \dots, r)$
- p は q を割り切る
- $f == l_1 * \dots * l_r \pmod{q/p}$
- $f == a_1 * \dots * a_r \pmod{q}$
- $u_i == a_i * y_i + b_i * z_i \pmod{p} \quad (i = 1, \dots, r)$

Initialize (Constructor)

```
HenselLiftSimultaneously(target: polynomial, factors: list, cofactors:
list, bases: list, p: integer)
    → HenselLiftSimultaneously
```

このオブジェクトはヘンゼルの補題によって引き上げられていく整数係数多項式の因数を保存.

```
f = target, gi in factors, his in cofactors and sis and tis are in bases.
from _factors(target: polynomial, factors: list, p: integer, ubound: in-
teger=sys.maxint)
    → HenselLiftSimultaneously
```

これは、因数が **HenselLiftMulti** によって引き上げられた *HenselLiftSimultaneously* のインスタンスを作成し返すためのクラスメソッドで、**HenselLiftMulti** はもし `sys.maxint` より小さければ `ubound` と一致し、さもなければ `sys.maxint` と一致する. 初期構成を補助する多項式を計算し直す必要はない; これらは他の引数によって用意される.

```
f = target, gis in factors.
```

Methods

1.1.3.1 lift – 一段階引き上げる

`lift(self) →`

引き上げメソッド. このメソッドのみ呼び出すべき.

1.1.3.2 first_lift – 最初のステップ

`first_lift(self) →`

引き上げを開始.

`f == l1*l2*...*lr (mod p**2)`

`di` たち,`ui` たち,`yi` たちそして `zi` たちの初期化. `ai` たちと,`bi` たちを更新. そして,`q` を `p**2` に更新.

1.1.3.3 general_lift – 次のステップ

`general_lift(self) →`

引き上げを続ける.

`f == a1*a2*...*ar (mod p*q)`

`ai` たち,`ubi` たち,`yi` たちそして `zi` たちを初期化. そして,`q` を `p*q`. に更新

1.1.4 lift_upto – main 関数

`lift_upto(self, target: polynomial, factors: list, p: integer, bound: integer)`
`→ tuple`

`bound` まで `target` の `factors mod p` をヘンゼルリフト氏,`factors mod q` と the `q` それ自身を返す.

以下の前提条件は満たされるべきである:

- `target` はモニック多項式.
- `target == product(factors) mod p`

結果 (`factors`, `q`) は以下の前提条件を満たす:

- k s.t. `q == p**k >= bound` なる k が存在
- `target == product(factors) mod q`

Bibliography

- [1] G. E. Collins and M. J. Encarnación. Improved techniques for factoring univariate polynomials. *Journal of Symbolic Computation*, Vol. 21, pp. 313–327, 1996.