Faktorizácia

Obsah

1 Úvod **3**

2 Úvodné pojmy **3**

3 Moduly na faktorizáciu polynómov **4**

4 ZZXFactoring **5**
4.1 factor 5
4.1.1 SFFactor 6
4.1.2 SquareFreeDecomp 6
4.2 MultiLift 7
4.3 mul 7

5 ZZ_pX,lzz_pX,ZZ_pEX,lzz_pEX, GF2X,GF2EXFactoring **8**
5.1 FindRoots 8
5.2 FindRoot 8
5.3 berlekamp 9
5.3.1 SFBerlekamp 9
5.4 CanZass 9
5.4.1 SquareFreeDecomp 10
5.4.2 SFCanZass 10
5.5 Funkcie a podfunkcie na zistenie ireducibility pre dané triedy 13
5.5.1 ProbIrredTest 13
5.5.2 DetIrredTest 14
5.5.3 IterIrredTest 14
5.5.4 BuildIrred 14
5.5.5 BuildRandomIrred 14
5.5.6 ComputeDegree 15
5.5.7 ProbComputeDegree 15
5.5.8 TraceMap 16
5.5.9 PowerCompose 17

6 Prídavné funkcie v triedach ZZ_pEX,lzz_pEX,GF2EXFactoring 19
6.1 IterComputeDegree 19
6.2 RecComputeDegree 20

7 Prídavná funkcia v GF2EXFactoring 21
7.1 FrobeniusMap 21

8 Prídavná funkcia v GF2XFactoring 21
8.1 BuildSparseIrred 22

9 Vizuálny prehľad funkcií v daných okruhoch-poliach 22

10 Podrobnosti implemetácie 22

1 Úvod

Táto príručka slúži na oboznámenie sa s faktorizačnými funkciami, ich podfunckciami a funkciami na testovanie nerozložiteľnosti polynómov v NTL. Najdôležitejšie faktorizačné funkcie v NTL sú **factor**, **CanZass** a **berlekamp**. Faktorizačné metódy sú komplexné, takže sa zjednodušujú na podúlohy. Napr. Faktorizácia s funkciou **CanZass** sa delí na 3 podúlohy, kde na každú podúlohu existuje funkcia.

1. Square free faktorizácia
   spraví z ľubovoľného polynómu square-free polynóm, čiže sa odstráni mnohonásobnosť ireducibilných faktorov

2. Distinct degree faktorizácia
   rozdelí square-free polynóm na equal-degree polynómy, ktorých ireducibilné faktory sú rovnakého stupňa,

3. Equal degree faktorizácia
   faktorizuje square-free polynóm, ktorého ireducibilné faktory sú rovnakého stupňa

Všetky hore uvedené faktorizačné funkcie majú podfunkciu SFFactor, SFCanZass, SFBerlekamp, ktoré vyžadujú za vstup *square-free polynóm* 2, inak sa výstupu nedočkáme. A tieto podfunkcie majú ďalšie podfunkcie, ktoré faktorizujú špecifické polynómy špeciálnymi predpokladmi na vstupe. V prípade nedodržania týchto predpokladov natrafíme na chybové hlásenie, alebo v horšom prípade sa nedočkáme výsledku. Väčšina funkcií vyžaduje, aby vstupný polynóm bol monický. V prípade nesplnenia tejto požiadavky, narazíme na chybové hlásenie typu: **bad args**.

2 Úvodné pojmy

- **Euklidovská vzdialenosť** x je rovná: $|x| = \sqrt{x_1^2 + ... + x_n^2}$.

- **Primitívny polynóm** je polynóm $a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$, s koeficientami $a_i \in GF(p)$, a $GCD(a_n, a_{n-1}, \ldots, a_1, a_0) = 1$

- **Monický polynóm** je polynóm, ktorého koeficient pri najväčšej mocnine sa rovná jednej.

  Príklad v GF3: 1x$^8 + 2x^4 + 2x + 1$

- **Square free polynóm** je polynóm, ktorý nemá mnohonásobné ireducibilné faktory

Príklad v GF2: $x^5 + 1 = ((x^4 + x^3 + x^2 + x + 1)^{\mathbf{1}}) * ((x + 1)^{\mathbf{1}})$

- **Equal degree polynóm** je taký square-free polynóm, ktorý má ireducibilné faktory rovnakého stupňa

  Príklad v GF2:

  $$x^8 + x^7 + x^6 + x^4 + 1 = ((x^{\mathbf{4}} + x^3 + x^2 + x + 1)^1) * ((x^{\mathbf{4}} + x + 1)^1)$$

- **Distinct degree polynóm** je taký square-free polynóm, ktorý je súčinom (equal degree) polynómov

  Príklad v GF3:

  $$x^{14} + 2x^{13} + x^{12} + 2x^{11} + x^{10} + 2x^9 + x^7 + 2x^5 + 2x^3 + x^2 + 2 =$$

  $$= ((x^6+x^5+x^4+2x^2+x+2)^1)*((x^8+x^7+2x^6+2x^5+x^4+2x^3+2x^2+x+1)^1)$$

  kde

  $$((x^6+x^5+x^4+2x^2+x+2)^1) = ((x^3+2x^2+x+1)^1)*((x^3+2x^2+2x+2)^1)$$

  a

  $$((x^8+x^7+2x^6+2x^5+x^4+2x^3+2x^2+x+1)^1) = ((x^4+x^2+x+1)^1)*((x^4+x^3+x^2+1)^1)$$

  sú (equal degree polynómy).

- **Lineárny faktor** je polynóm v tvare: $(a_1x + a_0)$, ktorý má koreň v $GF(p)$ a $a_i \in GF(p)$

- **Ireducibilný faktor** je ireducibilný polynóm v tvare: $(a_nx^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0)$, ktorý bol získaný faktorizáciou,pričom $a_i \in GF(p)$

3 Moduly na faktorizáciu polynómov

Na faktorizáciu polynómov je možné použiť nasledovné moduly.

- `ZZXFactoring` faktorizácia polynómov s jednou premennou nad ZZ
- `ZZ_pXFactoring` faktorizácia polynómov s jednou premennou nad ZZ_p

- `zz_pXFactoring` faktorizácia polynómov s jednou premennou nad zz_p
- `ZZ_pEXFactoring` faktorizácia polynómov s jednou premennou nad ZZ_pE
- `zz_pEXFactoring` faktorizácia polynómov s jednou premennou nad zz_pE
- `GF2XFactoring` faktorizácia polynómov s jednou premennou nad $GF2$
- `GF2EXFactoring` faktorizácia polynómov s jednou premennou nad $GF2E$

4 ZZXFactoring

V tomto moduli sú programy, ktoré slúžia na faktorizáciu polynómov v ZZX.
Nevyhnutné hlavičky potrebné na správny chod programov v danom moduli.

```
#include <NTL/ZZX.h>
#include <NTL/pair_ZZX_long.h>
#include <NTL/pair_ZZXFactoring.h>
//bez tejto hlavičky program bude vypisovať chybové hlášky
```

4.1 factor

```
void factor(ZZ& c,
            vec_pair_ZZX_long& factors,
            const ZZX& f,
            long verbose=0,
            long bnd=0);
```

- **Vstupy:** polynóm `f`, `verbose`, `bnd`
- **Výstupy:** `factors`, konštanta `c`

Vstupný polynóm `f` je ľubovoľný, `c` je `GCD` koeficientov polynómu `f` a `factors` sú primitívne a ireducibilné časti faktorizovaného polynómu `f`. Nenulová hranica `bnd`, znamená že polynóm `f` delí polynóm `h`, ktorého Euklidovská norma 2 je ohraničená absolútnou hodnotou 2^{bnd} a `verbose` je voľba, ktorá umožňuje podrobný výpis krokov faktorizácie, ak je nastavená na 1. Daná funcia využíva podfunkcie `SFFactor` a `SquareFreeDecomp`.

4.1.1 SFFactor

```
void SFFactor (vec_ZZX& factors,
                const ZZX& f,
                long verbose=0,
                long bnd=0);
vec_ZZX SFFactor(const ZZX& f,
                long verbose=0,
                long bnd=0);
```

- **Vstupy:** polynóm `f`, `verbose`, `bnd`
- **Výstupy:** `factors`

Vstupný polynóm `f` musí byť square free, s kladným vedúcim koeficientom. Hranica `bnd` a voľba `verbose` 4.1 má podobný význam, ako vo funkcii `factor`. V moduli `zz_pXFactoring` funkcia `SFCanZass` používa túto funkciu a funkciu `MultiLift`, na hľadanie faktorov pomocou hrubej sily.

4.1.2 SquareFreeDecomp

```
void SquareFreeDecomp(vec_pair_ZZX_long& u, const ZZX& f);
const vector(pair_ZZX_long SquareFreeDecomp(const ZZX& f);
```

- **Vstup:** polynóm `f`
- **Výstup:** `u`

Táto funkcia vykonáva (square-free) rozklad polynómu `f` s kladným vedúcim koeficientom. $f = \prod_i g_i^i$, kde `u` je reprezentované pármi (g_i, i). Square-free delitele vypisuje vzostupne podľa ich mocnín `i`. **Square-free deliteľ nemusí, ale môže byť ireducibilný!**

Príklad: Nech f je reprezentovaná polynómom v zápise:

```
[0 0 0 0 0 1 0 0 0 0 0 0 0 4]
```

potom `u (square-free) polynómy f` budú:

```
[[[1 0 0 0 0 0 0 0 4] 1] [[0 1] 5]]
```

kde ako vidíme polynómy

```
[1 0 0 0 0 0 0 0 4] a [0 1]
```

sú square-free, ale ešte prvý z nich nie je ireducibilný, čiže sa s ďalšími funkciami bude faktorizovať na:

```
[[[1 0 2 0 2] 1] [[1 0 -2 0 2] 1]
```

4.2 MultiLift

```
void MultiLift(vec_ZZX& A,
               const vec_zz_pX& a,
               const ZZX& f, long e,
               long verbose=0);
```

- **Vstupy:** polynóm `f`, faktory `a`, `verbose`, `e`
- **Výstup:** `A`

Pre daný polynóm `f mod p`, pre ktoré sú známe faktory `(a mod p)`, sa dá pomocou tejto funkcie faktorizovať vo väčšom module (p^e),pričom sa faktorizácia `f (a mod p)` povýši na faktorizáciu `f (A mod` p^e`)`, kde polynóm `f` a všetky polynómy v `a` sú monické.

Príklad: Nech f je reprezentovaná polynómom:

```
[1 0 0 0 0 0 0 0 0 0 0 0 1]
```

vektor `a`, ktorý reprezentuje faktory `f` modulo `p` je:

```
[[1 0 0 0 1] [1 0 0 0 1 0 0 0 1]]
```

V tomto príklade vektor `a` bol získaný faktorizáciou `f` modulo `2`. A posledný potrebný parameter `e` nech je rovné `3`. Potom výsledok `A` z hore uvedených parametrov bude:

```
[[1 0 0 0 1] [1 0 0 0 7 0 0 0 1]]
```

Poznámka: Daná funkcia očakáva na vstupe vektor `a` typu `zz_p`, čiže treba zadefinovať pole `GF(p)` príkazom `zz_p::init(p)`. Ak sa nezadefinuje, debugger vráti chybové hlásenie: **ZZ_p constructor called while modulus undefined**

4.3 mul

```
void mul(ZZX& x, const vec_pair_ZZX_long& a);
ZZX mul(const vec_pair_ZZX_long& a);
```

- **Vstup:** vectorový pár `a`
- **Výstup:** polynóm `x`

Funkcia `mul` slúži na vypočítanie súčinu polynómov z vektorového páru `a`.

5 ZZ_pX,lzz_pX,ZZ_pEX,lzz_pEX, GF2X,GF2EXFactoring

Moduly: `ZZ_pX`, `zz_pX`, `ZZ_pEX`, `zz_pEX`, `GF2X`, `GF2EXFactoring`

Súhrn: Programy sú poskytované na faktorizáciu polynómov nad **`ZZ_pX`**, **`zz_pX`**, **`ZZ_pEX`**, **`zz_pEX`**, **`GF2X`**, **`GF2EX`**, a na s ňou súvisiace problémy, ako testovanie nerozložiteľnosti (ireducibility) a vytváranie nerozložiteľných polynómov daného stupňa. V nasledujúcich funckiách sa parametre vzťahujú na triedu **`ZZ_pX`, ale všetky sú preťažené aj nad uvedenými triedami tejto sekcie, kde nie je uvedené inak.**

Nasledujúce hlavičky sú potrebné pre dané funkcie na správny chod programu.

```
#include <NTL/*.h>
#include <NTL/pair_*_long.h>
#include <NTL/*Factoring.h>
\\* môže byť ZZ_pX,zz_pX,ZZ_pEX,zz_pEX,GF2EX
```

5.1 FindRoots

```
void FindRoots(vec_ZZ_p& x, const ZZ_pX& f);
vec_ZZ_p FindRoots(const ZZ_pX& f);
```

- **Vstup:** polynóm `f`
- **Výstup:** korene vo vektore `x`

Funkcia `FindRoots` vracia korene reducibilného polynómu `f`, pričom sa predpokladá, že ak je stupňa n, potom má aj n koreňov. Daná funkcia v GF2X nie je definovaná, ale v `ZZ_pX` s voľbou $p = 2$ je možné pracovať ako v `GF2X`.

Poznámka: Keď je vstupný polynóm ireducibilný, alebo má ireducibilný faktor, tak sa výsledku nedočkáme.

5.2 FindRoot

```
void FindRoot(ZZ_p& root, const ZZ_pX& f);
ZZ_p FindRoot(const ZZ_pX& f);
```

- **Vstup:** polynóm `f`
- **Výstup:** koreň `root`

Nájde jeden koreň polynómu `f`, pri predpoklade, že `f` je monický, ktorý má aspoň jeden lineárny faktor. Ak nemá ani jeden, potom sa výsledku nedočkáme.

5.3 berlekamp

```
void berlekamp(vec_pair_ZZ_pX_long& factors,
               const ZZ_pX& f,
               long verbose=0);
```

- **Vstupy:** polynóm `f`, `verbose`
- **Výstup:** vektorový pár `factors` s mocninami ireducibilných faktorov

Berlekampov algoritmus vracia faktory polynómu `f` nad hore uvedenými triedami, okrem **`ZZ_pEX`, `zz_pEX`, `GF2X`**, pri predpoklade, že `f` je monický polynóm. Voľba `verbose` slúži na výpis postupu faktorizácie v NTL. K porozumeniu algoritmu nájdi reprezentačný program v materiáloch k predmetu *Rýchle algoritmy*.

Príklad:
Nech `f` je [1 0 0 0 0 0 0 0 0 0 0 0 1] v GF3, potom faktorizovaný polynóm `f` bude mať nasledujúce faktory: [[[2 1 1]3][[2 2 1]3]].

5.3.1 SFBerlekamp

```
void SFBerlekamp(vec_ZZ_pX& factors,
                 const ZZ_pX& f,
                 long verbose=0);
vec_ZZ_pX  SFBerlekamp(const ZZ_pX& f, long verbose=0);
```

- **Vstupy:** polynóm `f`, `verbose`
- **Výstup:** vektor `factors` s ireducibilnými faktormi

Funkcia je podfunkciou `berlekamp`-a. Vyžaduje, aby vstupný polynóm bol square-free, inak nedôjde k faktorizácii bez upozornenia chyby.

5.4 CanZass

```
void CanZass(vec_pair_ZZ_pX_long& factors, const ZZ_pX& f,
             long verbose=0);
vec_pair_ZZ_pX_long CanZass(const ZZ_pX& f, long verbose=0);
```

- **Vstupy:** polynóm `f`, `verbose`
- **Výstup:** vektorový pár `factors` s mocninami ireducibilných faktorov

Funkcia využívajúca funkcie SquareFreeDecomp a SFCanZass vracia ireducibilné faktory s počtom ich výskytu. Jediným predpokladom pre túto funkciu je, aby bol monický.

Poznámka:
Príklad pre danú funkciu nájdeš v príkladoch pre modulárnu aritmetiku.

5.4.1 SquareFreeDecomp

```
void SquareFreeDecomp(vec_pair_ZZ_pX_long& u,
                      const ZZ_pX& f);
vec_pair_ZZ_pX_long SquareFreeDecomp(const ZZ_pX& f);
```

Táto funkcia vykonáva (square-free) rozklad polynómu `f` s kladným vedúcim koeficientom. $f = \prod_i g_i^i$, kde `u` je reprezentované pármi (g_i, i). Square-free delitele vypisuje vzostupne podľa ich mocnín `i`. **Square-free deliteľ nemusí, ale môže byť ireducibilný!**

5.4.2 SFCanZass

```
void SFCanZass(vec_ZZ_pX& factors,
               const ZZ_pX& f,
               long verbose=0);
vec_ZZ_pX SFCanZass(const ZZ_pX& f,long verbose=0);
```

- **Vstupy:** polynóm `f`, `verbose`
- **Výstup:** vektor `factors` s ireducibilnými faktormi

Predpokladaný polynóm `f` je monický a (square free). SFCanZass funkcia použitím Cantor-Zassenhausovej metódy, ktorá využíva ďalej uvedené funcie NewDDF a EDF vracia zoznam ireducibilných faktorov polynómu `f`.

- **DDF(Distinct Degree Factorization)**2

```
void NewDDF(vec_pair_ZZ_pX_long& factors,
            const ZZ_pX& f,
            const ZZ_pX& h,
            long verbose=0);

vec_pair_ZZ_pX_long NewDDF(const ZZ_pX& f,
                           const ZZ_pX& h,
                           long verbose=0);
```

- **Vstupy:** polynóm `f`, `verbose`
  polynóm `h` sa vyžaduje tiež ako vstup, ale sa vypočítava po zadaní polynómu `f`
- **Výstup:** (equal-degree) `factors` s ich stupňami

Pre vstupný monický polynóm `f`, faktorizuje na (equal-degree)2 polynómy. Je využiteľná nad triedami tejto sekcie okrem nad triedou `GF2X`. Rozložené faktory sú uložené do zoznamu v tvare (g, d), kde `g` sú (equal-degree) faktory polynómu `f` so stupňami `d`, ktorý je stupňom všetkých ireducibilných polynómov v jednotlivých (equal-degree) faktoroch.

V zozname sú obsiahnuté len netriviálne (equal-degree) faktory polynómu `f`. (t.j. ak g!=1). Pre polynóm `h` sa prepokladá, že pri `ZZ_pX a zz_pX` sa rovná x^p mod f pri `ZZ_pEX a zz_pEX` sa rovná $x^{ZZ\_pE::cardinality()}$ mod f a pri `GF2EX` sa rovná $x^{2^{GF2::degree()}}$ mod f.

Príklad v GF3: Nech `f` je reprezentovaná polynómom:

$$x^{14} + 2x^{13} + x^{12} + 2x^{11} + x^{10} + 2x^9 + x^7 + 2x^5 + 2x^3 + x^2 + 2$$

```
[2 0 1 2 0 2 0 1 0 2 1 2 1 2 1]
```

polynóm `h` je x^3 `mod f`

```
[0 0 0 1]
```

potom (equal-degree) faktory polynómu sú:

$$(x^6 + x^5 + x^4 + 2x^2 + x + 2)$$

s ireducibilnými faktormi stupňa 3 a

$$(x^8 + x^7 + 2x^6 + 2x^5 + x^4 + 2x^3 + 2x^2 + x + 1)$$

s ireducibilnými faktormi stupňa 4

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

Poznámka:
Pri faktorizácii veľkých (large) polynómov funkcia používa vonkajší súbor na uloženie medzivýpočtov, ktoré sa vymažú po skončení programu. Tieto súbory sú uložené v dočasnom adresári s názvom:
`ddf-*-baby-* and ddf-*-giant-*`
Nastavením nasledujúcej premennej je možné nastaviť hodnotu (large).

```
extern double ZZ_pXFileThresh
\\počiatočná hodnota je 256
```

Vonkajšie súbory sa použijú len pri prekročení nastavenej veľkosti premennej `ZZ_pXFileThresh`.

- **EDF(Equal Degree Factorization)**3

```
void EDF(vec_ZZ_pX& factors, const ZZ_pX& f,
         const ZZ_pX& h, long d, long verbose=0);

vec_ZZ_pX EDF(const ZZ_pX& f, const ZZ_pX& h,
              long d, long verbose=0);
```

  - **Vstupy:** polynóm `f`, stupeň ired. polynómov `d`, `verbose` polynóm `h` sa vyžaduje tiež ako vstup, ale sa vypočítava po zadaní polynómu `f`
  - **Výstup:** ireducibilné `factors` so stupňami `d`

Predpokladajme polynóm `f`, ktorý je monický, (square free) a (equal-degree), potom daná funkcia faktorizuje na irreducibilné polynómy rovnakého stupňa.

Pre polynóm `h` sa prepokladá, že pri `ZZ_pX a zz_pX` sa rovná x^p mod f pri `ZZ_pEX a zz_pEX` sa rovná $x^{ZZ\_pE::cardinality()}$ mod f, pri `GF2X` sa rovná x^2 mod f a pri `GF2EX` sa rovná $x^{2^{GF2::degree()}}$ mod f.

Táto funkcia implementuje algoritmus z článku:
[von zur Gathen and Shoup, Computational Complexity 2:187-224, 1992].

Príklad v GF3: Nech `f` je reprezentovaná polynómom:

$$(x^8 + x^7 + 2x^6 + 2x^5 + x^4 + 2x^3 + 2x^2 + x + 1)$$

```
[1 1 2 2 1 2 2 1 1]
```

stupeň `d` ired. polynómov je 4 a polynóm `h` je x^3 `mod f`

```
[0 0 0 1]
```

potom ireducibilné faktory sú:

$$(x^4 + x^2 + x + 1)$$

```
[1 1 1 0 1]
```

a

$$(x^4 + x^3 + x^2 + 1)$$

```
[1 0 1 1 1]
```

```
void RootEDF(vec_ZZ_pX& factors,
             const ZZ_pX& f,
             long verbose=0);
vec_ZZ_pX RootEDF(const ZZ_pX& f, long verbose=0);
```

- **Vstup:** polynóm `f`, `verbose`
- **Výstup:** korene vo vektore `factors`

Táto funkcia vracia korene (equal-degree) polynómov stupňa $d = 1$.

Poznámka:
Tieto funkcie používajú modulárnu aritmetiku. Veľkosť pamäte požadovanej tabuľky, môže byť nastavené pomocou príslušnej premennej pre danú triedu:

```
ZZ_pXArgBound (pozri ZZ_pX.txt).
zz_pXArgBound (pozri zz_pX.txt).
ZZ_pEXArgBound (pozri ZZ_pEX.txt).
zz_pEXArgBound (pozri zz_pEX.txt).
GF2XArgBound (pozri GF2X.txt).
GF2EXArgBound (pozri GF2EX.txt).
```

5.5 Funkcie a podfunkcie na zistenie ireducibility pre dané triedy

5.5.1 ProbIrredTest

```
long ProbIrredTest(const ZZ_pX& f, long iter=1);
```

- **Vstup:** polynóm `f`, `iter`
- **Výstup:** návratová hodnota 0 alebo 1

vykonáva rýchly pravdepodobnostný test irreducibility v každej triede okrem v triede `GF2X`. Pravdepodobnosť chyby testu je p^{-iter}, ak polynóm `f` je rozložiteľný. Implementácia algoritmu je z článku: [Shoup, J. Symbolic Comp. 17:371-391, 1994].

Poznámka: Na presnejší odhad irreducibility než 50% zadaj `iter>1`.

5.5.2 DetIrredTest

```
long DetIrredTest(const ZZ_pX& f);
```

- **Vstup:** polynóm `f`
- **Výstup:** návratová hodnota 0 alebo 1

vykonáva rekurzívny deterministický test irreducibility v každej triede okrem v triede `GF2X`. Rýchly aj v najhoršom prípade, keď vstup je irreducibilný. Implementácia algoritmu je z článku: [Shoup, J. Symbolic Comp. 17:371-391, 1994].

5.5.3 IterIrredTest

```
long IterIrredTest (const ZZ_pX& f);
```

- **Vstup:** polynóm `f`
- **Výstup:** návratová hodnota 0 alebo 1

vykonáva iteračno-deterministický test irreducibility na základe DDF. Priemerne rýchly test, keď polynóm `f` má malý faktor.

5.5.4 BuildIrred

```
void BuildIrred (ZZ_pX& f, long n);
ZZ_pX BuildIrred_ZZ_pX(long n);
```

- **Vstup:** zvolený stupeň `n`
- **Výstup:** ireducibilný polynóm `f`

zostrojí monický irreducibilný polynóm stupňa `n`

5.5.5 BuildRandomIrred

```
void BuildRandomIrred (ZZ_pX& f, const ZZ_pX& g);
ZZ_pX BuildRandomIrred(const ZZ_pX& g);
```

- **Vstup:** ireducibilný polynóm `g`
- **Výstup:** ireducibilný polynóm `f` rovnakého stupňa ako `g`

Polynóm `g` je monický irreducibilný polynóm, ku ktorému príslušná funkcia vygeneruje náhodný monický irreducibilný polynóm rovnakého stupňa.

5.5.6 ComputeDegree

```
long ComputeDegree (const ZZ_pX& h, const ZZ_pXModulus& F);
```

- **Vstupy:** polynóm `F` polynóm `h` sa vyžaduje tiež ako vstup, ale sa vypočítava po zadaní polynómu `F`
- **Výstup:** návratová hodnota funkcie

Návratová hodnota je deg(f) ak ireducibilné polynómy sú rôzneho stupňa alebo niektorý z nich je mnohonásobný, **alebo spoločný stupeň ireducibilných polynómov**, ak každý ireducibilný polynóm je rôzny a zároveň jedennásobný rovnakého stupňa.

Funkcia sa nachádza iba v triedach `ZZ_pX` a `zz_pX`. Na vstupe očakáva (equal-degree) 2 polynóm a polynóm h je reprezentovaný, ako $h = x^p \bmod f$.

Príklad1:

```
p:  2
f: [1 0 0 0 0 0 0 0 0 0 0 0 1]
\\\\\\vedĺajšia informácia\\\\\\\\\\
factors: [[[1 1] 4] [[1 1 1] 4]]
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
h: [0 0 1]
návratová hodnota: 12, lebo faktory sú mnohonásobné
a nie sú ani rovnakého stupňa
```

Príklad2:

```
p: 11
f: [1 0 0 0 0 0 0 0 0 0 0 0 1]
\\\\\\\vedĺajšia informácia\\\\\\\\\
factors: [[[10 8 1] 1] [[10 3 1] 1] [[10 5 1] 1]
          [[10 2 1] 1] [[10 6 1] 1] [[10 9 1] 1]]
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
h: [0 0 0 0 0 0 0 0 0 0 0 1]
návratová hodnota: 2, lebo každý z faktorov má rovnaký stupeň
```

5.5.7 ProbComputeDegree

```
long ProbComputeDegree (const ZZ_pX& h,
                        const ZZ_pXModulus& F);
```

- **Vstupy:** polynóm `F`
  polynóm `h` sa vyžaduje tiež ako vstup, ale vypočítava sa pomocou zadefinovaného poľa a polynómu `F`
- **Výstup:** návratová hodnota funkcie

Funkcia nachádzajúca sa iba v triedach `ZZ_pX` a `zz_pX` je podobná predošlej funkcii, pričom používa trochu rýchlejší pravdepobnostný algoritmus.

Návratová hodnota je nula ak ireducibilné polynómy nie sú rovnakého stupňa alebo niektorý z nich je mnohonásobný, **alebo spoločný stupeň ireducibilných polynómov**, ak každý ireducibilný polynóm je jedennásobný a zároveň rôzny.

5.5.8 TraceMap

```
void TraceMap (ZZ_pX& w, const ZZ_pX& a,
               long d, const ZZ_pXModulus& F,
               const ZZ_pX& h);

ZZ_pX TraceMap(const ZZ_pX& a, long d,
                const ZZ_pXModulus& F,
                const ZZ_pX& h);
```

- **Vstupy:** polynómy `a`, `F`, `h` a konštanta `d`
  polynóm `h` sa vyžaduje tiež ako vstup, ale sa vypočítava po zadaní polynómu `F` a
- **Výstup:** polynóm `w`

$w = a + a^q + \ldots + a^{q^{d-1}}$ mod f; $d >= 0$; $h = x^q$ mod f, kde pri (`ZZ_pX` a `zz_pX`) `q` je charakteristikou pola `GF(q)`, pri (`ZZ_pEX` a `zz_pEX`) `q` je kardinalitou pola `GF(q)` $ZZ\_pE :: cardinality()$ a pri `GF2EX` `q` je $2^{GF2::degree()}$.

Poznámka: Táto pomocná funkcia sa používa napr. pri pravdepodobnostnom testovaní ireducibility polynómu vo funkcii `ProbIrredTest`. Ak $deg(w) <= 0$ potom je testovaný polynóm ireducibilný s pravdeobodobnosťou v závislosti od počtu iterácii vo funkcii `ProbIrredTest`. 5.5

Implementáciu algoritmu funkcie pozri v článku: [von zur Gathen and Shoup, Computational Complexity 2:187-224,1992].

Príklad 1

```
Zadaj prvocislo reprezentujuce pole!:
```

```
2                           //p
Zadaj polynom f
[1 1 1]                     //F

Stupen polynomu 'F' je: 2   //d = deg(F)

Polynom 'h' je: [1 1]       //h = h^p % F

Polynom 'a' je: [1 1]       //náhodne vygenerovaný polynóm stupňa < n

Polynom 'w' je:             //w = a+a^p+...+^{p^{n-1}} mod F;
[]
Stupen polynomu 'w' je:
-1
```

Z výsledku funkcie usudzujeme 50 percentnou pravdepodobnosťou, že polynóm `F` je ireducibilný, lebo je $deg(w) <= 0$.

Príklad 2

```
Zadaj prvocislo reprezentujuce pole!:
2                           //p
Zadaj polynom f
[1 0 1]                     //F

Stupen polynomu 'F' je: 2   //d = deg(F)

Polynom 'h' je: [1]

Polynom 'a' je: [1]         //h = h^p % F

Polynom 'w' je:             //náhodne vygenerovaný polynóm stupňa < n
[1 1]
Stupen polynomu 'w' je:     //w = a+a^p+...+^{p^{n-1}} mod F;
1
```

Z výsledku funkcie usudzujeme 50 percentnou pravdepodobnosťou, že polynóm `F` nie je ireducibilný, lebo $deg(w) > 0$.

5.5.9 PowerCompose

```
void PowerCompose(ZZ_pX& w, const ZZ_pX& h,
                  long d, const ZZ_pXModulus& F);
ZZ_pX PowerCompose(const ZZ_pX& h, long d,
                   const ZZ_pXModulus& F);
```

- **Vstupy:** polynómy `F`, `h` a konštanta `d`
  polynóm `h` sa vyžaduje tiež ako vstup, ale sa vypočítava po zadaní polynómu `F`
- **Výstup:** polynóm `w`

$w = x^{q^d}$ mod f; $d >= 0$; $h = x^q$ mod f, kde kde pri (`ZZ_pX` a `zz_pX`) `q` je charakteristikou pola `GF(q)`, pri (`ZZ_pEX` a `zz_pEX`) `q` je kardinalitou pola `GF(q)` $ZZ\_pE :: cardinality()$ a pri `GF2EX` `q` je $2^{GF2::degree()}$.

Poznámka: Táto pomocná funkcia sa používa napr. pri pravdepodobnostnom testovaní ireducibility polynómu vo funkcii `DetIrredTest`. Ak $w = x$, potom testovaný polynóm `F` je ireducibilný.

Implementáciu algoritmu funkcie pozri v článku: [von zur Gathen and Shoup, Computational Complexity 2:187-224,1992].

Príklad 1:

```
Zadaj prvocislo reprezentujuce pole!:
2                              //p
Zadaj polynom F                //F
[1 1 1]

Stupen polynomu 'F' je: 2      //deg(F)

Polynom 'h' je: [1 1]          // h = h^{ZZ_p::modulus()} % F

Polynom 'w' je:                //w = X^{q^d} mod f
[0 1]
```

Keďže $w = x$, polynóm `F` je ireducibilný.

Príklad 2:

```
Zadaj prvocislo reprezentujuce pole!:
2
Zadaj polynom F
[1 0 1]

Stupen polynomu 'F' je: 2

Polynom 'h' je: [1]

Polynom 'w' je:
[1]
```

Keďže $w = 1$, polynóm F nie je ireducibilný.

6 Prídavné funkcie v triedach ZZ_pEX,lzz_pEX,GF2EXFactoring

Moduly: Prídavné funkcie na zistenie ireducibility v triedach
ZZ_pEX,zz_pEX,GF2EXFactoring

Súhrn: V nasledujúcich funckiách sa parametre vzťahujú na triedu ZZ_pEX, ale všetky sú preťažené aj nad hore uvedenými triedami danej sekcie.

6.1 IterComputeDegree

```
long IterComputeDegree(const ZZ_pEX& h,
                       const ZZ_pEXModulus& F);
```

- **Vstupy:** polynómy h a F
- **Výstup:** návratová hodnota funkcie

Návratová hodnota je deg(F) ak ireducibilné polynómy sú rôzneho stupňa alebo niektorý z nich je mnohonásobný, **alebo spoločný stupeň ireducibilných polynómov**, ak každý ireducibilný polynóm je rôzny a zároveň jedennásobný rovnakého stupňa.

Účel funkcie je rovnaký, ako napr. funkcie ComputeDegree 5.5.5, ale pre rozšírené polia. V danej funkcii sa od polynómu f očakáva, aby bol (equal-degree) polynóm pričom platí, že v triedách(ZZ_pEX a zz_pEX) $h = x^{ZZ\_pE::cardinality()} \bmod f$ a v triede GF2EX $h = x^{2^{GF2E::degree()}} \bmod f$.

Funkcia používa (baby step/giant step) algoritmus, ktorý je použitý aj vo funkcii NewDDF.

Príklad 1:

```
p: 2
P: [1 1 1] //ireducibilný polynóm 2-ho stupňa
kardinalita je: 4
```

Kardinalita je $p^{deg(P)}$.

```
F:[[1] [0 1] [1 1] [] [0 1] [] [1]]
```

polynóm F je $x^6 + (\alpha)x^4 + (\alpha + 1)x^2 + (\alpha)x + 1$

```
factors: [[[[1] [1] [] [1]] 1] [[[1] [1 1] [] [1]] 1]]
```

ktorého faktory sú: $(x^3 + x + 1)^1$ a $(x^3 + (\alpha + 1)x + 1)^1$

```
h modulo F: [[] [] [] [] [1]]
```

`h mod F` je x^4
a nakoniec návratová hodnota funkcie je 3, čo je stupeň ireducibilných polynómov polynómu `F`.

Príklad 2:

```
p: 2
P: [1 1 1] //irreducibilný polynóm 2-ho stupňa
kardinalita je: 4
```

Kardinalita je $p^{deg(P)}$.

```
[[1] [0 1] [1 1] [1 1] [] [1]]
```

polynóm `F` je $x^5 + (\alpha + 1)x^3 + (\alpha + 1)x^2 + (\alpha)x + 1$

```
factors: [[[[0 1] [1]] 1] [[[0 1] [0 1] [1]] 1] [[[1 1] [1]] 2]]
```

ktorého faktory sú: $(x + \alpha)^1$, $(x^2 + (\alpha)x + \alpha)^1$ a $(x + (\alpha + 1))^2$

```
[[] [] [] [] [1]]
```

`h mod F` je x^4
a nakoniec návratová hodnota funkcie je 5, čo je stupeň vstupného polynómu `F`, keďže ireducibilné faktrory sú rôzneho stupňa a jeden faktor je viacnásobný.

6.2 RecComputeDegree

```
long RecComputeDegree (const ZZ_pEX& h,
                       const ZZ_pEXModulus& F);
```

- **Vstupy:** polynómy `h` a `F`
- **Výstup:** návratová hodnota funkcie

V danej funkcii sa od polynómu `f` očakáva, aby bol (equal-degree) polynóm pričom platí, že v triedách (`ZZ_pEX` a `zz_pEX`) $h = x^{ZZ\_pE::cardinality()} \bmod f$ a v triede `GF2EX` $h = x^{2^{GF2E::degree()}} \bmod f$.

Funkcia vracia **spoločný stupeň irreducibilných faktorov** a používa rekurzívny algoritmus podobný algoritmu, ktorý je použitý aj vo funkcii DetIrredTest.

7 Prídavná funkcia v GF2EXFactoring

Moduly: Prídavná funkcia na zistenie ireducibility v triede `GF2EXFactoring`

7.1 FrobeniusMap

```
void FrobeniusMap (GF2EX& h, const GF2EXModulus& F);
GF2EX FrobeniusMap(const GF2EXModulus& F);
```

- **Vstup:** polynóm `F`
- **Výstup:** polynóm `h`

Nasledujúca funkcia buď iteračno-umocňovacou alebo modulárnou metódou spočíta $h = X^{2^{GF2E::degree()}} \bmod F$.

Modulárna metóda je vyvinutá prostredníctvom (Kaltofen & Shoup) - (Rýchla polynomická faktorizácia nad vysoko algebraicky rozšírenými konečnými poľami, ISAAC 1997). Táto metóda je rýchlejšia ako iteračno-umocňovacia metóda, keď stupeň polynómu `F` je veľká v porovnaní s $GF2E :: degree()$.

Príklad:

`P: [1 1 1]`

P je $x^2 + x + 1$

`F 3-ho stupňa: [[0 1] [0 1] [] [1 1]]`

`F 3-ho stupňa:` je $(\alpha + 1)x^3 + (\alpha)x + \alpha$

`monický F štvrtého stupňa: [[0 1] [0 1] [] [1 1] [1]]`

`monický F:` je $x^4 + (\alpha + 1)x^3 + (\alpha)x + \alpha$

`h: [[0 1] [0 1] [] [1 1]]`

a `h` je $(\alpha + 1)x^3 + (\alpha)x + \alpha$.

8 Prídavná funkcia v GF2XFactoring

Moduly: Prídavná funkcia na zistenie ireducibility v triede `GF2XFactoring`

8.1 BuildSparseIrred

```
void BuildSparseIrred (GF2X& f, long n);
GF2X BuildSparseIrred_GF2X(long n);
```

- **Vstup:** konštanta `n` reprezentujúci stupeň ired. polynómu `h`
- **Výstup:** ireducibilný polynóm `h`

Funkcia na vytvorenie monického ireducibilného polynómu stupňa `n`. Z ireducibilných trojčlenných polynómov v tvare $X^n + X^k + 1$ sa vyberie polynóm s minimálnou hodnotou k, alebo z ireducibilných päťčlenných polynómov v tvare $X^n + X^{k_3} + X^{k_2} + X^{k_1} + 1$ sa vyberie polynóm s najmenšími hodnotami k_i v poradí od i rovné 3 až po i rovné 1. Keď neexistuje ani trojčlenný ani päťčlenný ireducibilný polynóm, potom sa vygeneruje nejaký iný pomocou funkcie BuildIrred, avšak sa domnieva aj keď to nie je dokázané, že existujú pre každé $n > 1$. Pre stupeň $n <= 2058$, polynóm je vytvorený tabuľkovým vyhľadávaním v predpočítanej tabuľke. Zaobstarané výstupy pre triedy `GF2X` a nepriamo pre `GF2EX` sú optimalizované pomocou funkcie BuildSparseIrred.

9 Vizuálny prehľad funkcií v daných okruhoch-poliach

10 Podrobnosti implemetácie

Pri faktorizácii polynómu sa najprv extrahuje jeho obsah, ktorým dosiahneme, aby bol square-free.

Druhým krokom je pokus o jednoduché rozsekanie (simple hack). Ak je polynóm v tvare $g(x^l)$, tak sa pokúsime prepísať na tvar $g(k^m)$, kvôli získaniu deliteľov `l`. Tento krok v niektorých prípadoch značne uľahčuje faktorizačnú úlohu. Nastavením nasledujúcej premennej na 1/0 je možné túto voľbu zapnúť/vypnúť.

```
extern long ZZXFac_PowerHack;
// počiatočná hodnota = 1
```

Tretím krokom je modulárne delenie s nejakými malými prvočíslami, z ktorých sa zvolí jedno ako najlepšie. Nasledujúcim príkazom je možné nastaviť počet spomenutých malých prvočísel.

```
extern long ZZXFac_InitNumPrimes;
// počiatočná hodnota = 7
```

Štvrtým krokom je zvýšenie faktorizácie z `mod p` na faktorizáciu `mod` p^k pre dostatočne veľké `k` pomocou kvadratickej Henselovej metódy zvýšenia.

Posledným krokom je rekombinančná fáza. V tejto fáze musíme zlúčiť podmnožiny modulárnych faktorov a otestovať, či sú faktormi nad číslami.

Uvedieme dve základné metódy:

- metódu Zassenhausa
- metódu van Hoeij-a

Predvolená metóda Marka van Hoeija je celkom nová, ale je lepšia ako staršia metóda Zassenhausa. Podrobný popis algoritmu sa nachádza na domovskej stránke Marka van Hoeij-a: http://www.openmath.org/ hoeij/

Tento algoritmus nie je veľmi špecifický, ale vo všeobecnom priblížení, veľmi dobre pracuje so všetkými vstupnými polynómami.
Nastavením nasledujúcej premennej sa nastavuje voľba hore uvedených dvoch metód.

- pri metóde Zassenhausa sa premenná = 0
- pri metóde van Hoeij-a sa premenná = 1

```
extern long ZZXFac_van_Hoeij;
// počiatočná hodnota = 1
```

Pri Hoeij-ovej metóde premenná `power hack` je predvolene nastavená, pokiaľ výpočet nie je veľmi časovo náročný.
Zassenhausova metóda je v podstate hľadanie hrubou silou s veľmi efektívnymi metódami popísané v článku:

[J. Abbott, V. Shoup, P. Zimmermann, (Factoring in Z[x]: the searching phase), ISSAC 2000].

Tieto heuristiky sú značne effektívne a ľahko si poradia aj s 40-imi modulárnymi faktormi, čo je oveľa viac v porovnaní so Zassenhausovou metódou.

Správanie týchto heuristík je možné nastaviť pomocou nasledujúcich globálnych premenných.

```
extern long ZZXFac_MaxNumPrimes;
 // počiatočná hodnota = 50
```

Počas rekombinačnej fázy faktorizácie, ak je priebeh pomalý, občas sa získa viac informácií s faktorizáciou modulo iné prvočíslo. Táto informácia je používaná na vyškrtnutie stupňov potenciálnych faktorov. Táto hodnota ohraničuje celkový počet prvočísel modulo ktorými je polynóm `f` faktorizovaný.

```
 extern long ZZXFac_MaxPrune;
// počiatočná hodnota = 10
```

Fáza rekombinácie na zníženie prehľadávacieho priestoru používa stratégiu podobajúcu sa na stratégiu "stretnutie v strede". Táto stratégia pre množstvo polynómov, ale nie pre všetky je schopná znížiť výpočtový čas.

Keď $t = ZZXFac\_MaxPrune$, potom výpočtový čas sa zníži približne na 2^t, pričom výpis zaberie najviac $t*(2^{t-1})$ bytov pamäte. Treba poznamenať, že táto premenná je prispôsobená hornej hranici `t`, ktorú môže faktorizačný algoritmus znížiť z rôznych dôvodov.