# Príklady na faktorizáciu

## Obsah



## 1 SquareFreeDecomp

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]
```

## 2 MultiLift

**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]]
```

Nech `f` je typu `ZZ_pX`, pričom `p=2` a 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 vyžaduje, aby bolo nastavené modulo `p` pre vektor `a`.

```
zz_p::init(p)
```

## 3 Berlekamp

**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]].

```
#include <NTL/zz_pX.h>
#include <NTL/zz_pXFactoring.h>
#include <NTL/pair_ZZX_long.h>
#include <conio.h>

NTL_CLIENT

int main()
{
  ZZ p;
  ZZ_pX f;

  vec_pair_ZZ_pX_long factors;

  cin >> p;
  ZZ_p::init(p);
  cin>>f;
  berlekamp(factors,f,0);
  cout<<factors;
  getch();
  return 1;
}
```

# 4 ComputeDegree

```
#include <NTL/zz_pX.h>
#include <NTL/zz_pXFactoring.h>
#include <NTL/pair_ZZX_long.h>
#include <conio.h>

NTL_CLIENT

int main()
{
  long i;
  ZZ p;
  ZZ_pX f,h;

  vec_pair_ZZ_pX_long factors;

  cin >> p;
  ZZ_p::init(p);
  cin>>f;
  CanZass(factors,f,0);
  cout<<factors<<endl;
  cin>>h;
  h=h%f;
  i=ComputeDegree(h,f);
  cout<<i;
  getch();
  return 1;
}
```

**Poznámka:** Ak faktory polynómu `f` sú mnohonásobné alebo nie sú rovnakého stupňa návratová hodnota funkcie bude `deg(f)`. Inak spoločný stupeň irreducibilných faktorov.

**Príklad1:**

```
p:  2
f: [1 0 0 0 0 0 0 0 0 0 0 0 1]
factors: [[[1 1] 4] [[1 1 1] 4]]
h: [0 0 1]
i: 12
```

**Príklad2:**

```
p: 11
```

```
f: [1 0 0 0 0 0 0 0 0 0 0 0 1]
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]
i: 2
```

## 5 TraceMap

```
#include <NTL/ZZ_pX.h>
#include <NTL/ZZ_pXFactoring.h>
#include <NTL/pair_ZZX_long.h>

NTL_CLIENT

int main()
{
long d;
ZZ p;
ZZ_pX F,h,w,a;
vec_pair_ZZ_pX_long factors;

cout<<"Zadaj prvocislo reprezentujuce pole!: "<<endl;
cin >> p;
ZZ_p::init(p);

cout<<"Zadaj polynom f"<<endl;
cin>>F;

long n = deg(F);
cout<<endl<<"Stupen polynomu 'F' je: "<<n<<endl;

PowerXMod(h, p, F); // h = h^p % F
cout<<endl<<"Polynom 'h' je: "<<h<<endl;

random(a, n); // generate a random polynomial of degree < n
cout<<endl<<"Polynom 'a' je: "<<h<<endl;

TraceMap(w,a,n,F,h); // w = a+a^p+...+^{p^{n-1}} mod F;
cout<<endl<<"Polynom 'w' je: "<<endl;
cout<<w;

cout<<endl<<"Stupen polynomu 'w' je: "<<endl;
cout<<deg(w);
```

```
return 1;
}
```

**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 $deg(w) > 0$.

## 6 PowerCompose

```
#include <NTL/ZZ_pX.h>
#include <NTL/ZZ_pXFactoring.h>
#include <NTL/pair_ZZX_long.h>

NTL_CLIENT

int main()
{
long d;
ZZ p;
ZZ_pX F,h,w;

cout<<"Zadaj prvocislo reprezentujuce pole!: "<<endl;
cin >> p;
ZZ_p::init(p);

cout<<"Zadaj polynom F"<<endl;
cin>>F;

long n = deg(F);
cout<<endl<<"Stupen polynomu 'F' je: "<<n<<endl;

PowerXMod(h, ZZ_p::modulus(), F); // h = h^{ZZ_p::modulus()} % F
cout<<endl<<"Polynom 'h' je: "<<h<<endl;

PowerCompose(w, h, n, F); // w = x^{p^n} mod F
cout<<endl<<"Polynom 'w' je: "<<endl;
cout<<w<<endl<<endl;

if(IsX(w))
cout<<"F je ireducibilny!";
else
cout<<"F nie je ireducibilny!";

return 1;
}
```

**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.

# 7 IterComputeDegree

```
#include <NTL/ZZ_pEX.h>
#include <NTL/ZZ_pXFactoring.h>
#include <NTL/ZZ_pEXFactoring.h>
#include <NTL/pair_ZZ_pEX_long.h>

NTL_CLIENT

int main()
{
long i,r;
ZZ p;

cout<<"Zadaj prvocislo reprezentujuce pole!: "<<endl;
cin >> p;
ZZ_p::init(p);
```

```
ZZ_pX P;
BuildIrred(P, 2); //vygeneruje ireducibilny polynom stupna 2
cout<<P<<endl;

ZZ_pE::init(P);
q=ZZ_pE::cardinality();
r=to_long(q);    //kardinalita pola bude p^2
cout<< r <<endl;

/////1. pripad - vygenerovany polynom F\\\\\\\\

ZZ_pEX F,h, i1,i2;
//vygeneruju sa 2 ired. polynomy rovnakeho stupna
// a vypocita sa ich sucin

BuildIrred(i1,3);
cout<<endl<<"Ired. poly. 1: "<<endl<<i1<<endl<<endl;

BuildRandomIrred(i2,i1);
cout<<endl<<"Ired. poly. 2: "<<endl<<i2<<endl<<endl;

mul(F,i1,i2);
cout<<endl<<"Sucin i1 a i2: "<<endl<<F<<endl<<endl;

CanZass(factors, F, 0);
cout<<endl<<"Faktory su: "<<endl<<factors<<endl<<endl;
PowerXMod(h,r,F);

cout <<endl<< h <<endl;
i=IterComputeDegree(h,F);
//otestovanie nami vytvoreneho polynomu,
//ci vracia spolocny stupen ired. polynomov

cout<<endl<<i;

cout<<endl<<endl;

/////2. pripad - lubovolny polynom F\\\\\\

cout<<"Pri nahodne vygenerovanom F"<<endl<<endl;
random(F, 5);
cout<<endl<<"Polynom F"<<endl<<F<<endl<<endl;
SetCoeff(F,5);
cout<<endl<<F<<endl;
```

```
CanZass(factors, F, 0);
cout<<endl<<"Faktory su: "<<endl<<factors<<endl<<endl;

PowerXMod(h,r,F);
cout <<endl<< h <<endl;

i=IterComputeDegree(h,F);
//otestovanie lubovolneho polynomu co vracia

cout<<endl<<i;
return 1;
}
```

**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.

# 8 FrobeniusMap

```
#include <NTL/GF2EX.h>
#include <NTL/GF2XFactoring.h>
#include <NTL/GF2EXFactoring.h>
#include <NTL/pair_GF2EX_long.h>
#include <conio.h>

NTL_CLIENT

int main()
{
  long q;
  vec_pair_GF2EX_long factors;

  GF2X P;
  BuildIrred(P, 2);
  cout<<P<<endl;
  GF2E::init(P);

  GF2EX F, h;
  random(F, 5); \\generuje polynóm F stupňa < 5
  SetCoeff(F,5); \\nastaví 5-ty koeficient polynómu F na 1
  FrobeniusMap(h,F);
  cout<<endl<<h;
  getch();
  return 1;
```

**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 $(\alpha + 1)x^3 + (\alpha)x + \alpha$.