Príklady polynómov

Obsah

NTL poskytuje rozsiahlu podporu pre veľmi rýchlu polynomiálnu aritmetiku. V prvej rade to v skutočnosti bola hlavná motivácia pre vytvorenie NTL, pretože existujúce počítačové algebraické systémy a knižničné software-i mali veľmi pomalú polynomiálnu aritmetiku. Trieda ZZX reprezentuje polynómy s celočíselným koeficientom.

1 Prvý príklad

Nasledujúci program číta polynóm, ten rozdelí na faktory a vypíše ich

```
#include <NTL/ZZXFactoring.h>

NTL_CLIENT

int main()
{
   ZZX f;

   cin >> f;

   vec_pair_ZZX_long factors;
   ZZ c;

   factor(c, factors, f);

   cout << c << "\n";
   cout << factors << "\n";
}
```

Keď na vstupe zadáme hodnoty:

```
[2 10 14 6]
```

ktorý predstavuje polynóm $2 + 10 * x + 14 * x^2 + 6 * x^3$, výstupom bude:

```
2
[[[1 3] 1] [[1 1] 2]]
```

Prvý riadok výstupu je obsah polynómu, ktorý je v tomto prípade 2, ako každý koeficient so vstupným polynómom rozdeleným na 2 faktory. Druhý riadok je vektor párov, kde prvý člen každého paru je irreducibilný faktor výstupu, a druhý je exponent, ktorým je vytvorený rozklad faktorov. Takže všetko vyššie uvedené jednoducho znamená:

```
2 + 10*x + 14*x^2 + 6*x^3 = 2 * (1 + 3*x) * (1 + x)^2
```

Nepochybne `I/O` v NTL nie je vlastne užívateľsky príjemný, ale zámerom NTL nebol algebraicky počítačový systém. Je to knižnica pre programátorov.

V tomto príklade sa nachádza typ vektora `vec_pair_long_ZZ`, kde jeho základnú časť v NTL tvorí typ `pair_long_ZZ`. Typ `pair_long_ZZ` je typ vytvorený pre ďalšie šablóny ako makro mechanizmu. Vo všeobecnosti platí, že typy S a T môžu vytvoriť typ `pair_S_T`, ktorý má triedu s polom a typu S a polom b typu T.

2 Druhý príklad

Výstupom nasledujúceho programu je prvých 100 cyclotomických polynómov

```
#include <NTL/ZZX.h>

NTL_CLIENT

int main()
{
   vec_ZZX phi(INIT_SIZE, 100);

   for (long i = 1; i <= 100; i++) {
      ZZX t;
      t = 1;

      for (long j = 1; j <= i-1; j++)
         if (i % j == 0)
            t *= phi(j);

      phi(i) = (ZZX(i, 1) - 1)/t;  // ZZX(i, a) == X^i * a

      cout << phi(i) << "\n";
   }
}
```

Na tomto príklade môžeme názorne ukázať iné alternatívy príkazov .
1.) namiesto:

```
   vec_ZZX phi(INIT_SIZE, 100);
```

môžeme napísať:

```
   vec_ZZX phi;
   phi.SetLength(100);
```

2.) namiesto:

```
      t *= phi(j);
```

môžeme napísať:

```
      mul(t, t, phi(j));
alebo
      t = t * phi(j);
```

taktiež je možné napísať phi[j-1] namiesto phi(j)
3.) namiesto:

```
      phi(i) = (ZZX(i, 1) - 1)/t;
```

môžeme napísať:

```
ZZX t1;
SetCoeff(t1, i, 1);
SetCoeff(t1, 0, -1);
div(phi(i), t1, t);
```

Inak by sa priamo mohol sprístupniť koeficient vektora:

```
ZZX t1;
t1.rep.SetLength(i+1);
     // všetky elementy vektora sú inicializované na 0
t1.rep[i] = 1;
t1.rep[0] = -1;
t1.normalize();
     // nie je tu potrebná,
     //ale je dobrá pri obvyklých postupoch
div(phi(i), t1, t);
```

Koeficient vektora polynómu je vždy nejaký NTL vektor cez základný okruh: v tomto prípade `vec_ZZ`.
NTL Vám dáva voľný prístup ku koeficientu vektora, ale treba vedieť, že že vedúci nenulový koeficient je *invariant*, ktorý všetky postupy, ktoré pracujú s polynómami bude zachovávať.Samozrejme, ak sa môžete vyhnúť priamemu sprístupneniu koeficienta vektora, urobte tak. Vždy totiž to môžete použiť SetCoeff, bežný pre nastavenie alebo zmenu koeficientov, a môžete vždy čítať hodnoty koeficientov použitím coeff:

```
... f.rep[i] == 1 ...
```

je ekvivalentné s

```
... coeff(f, i) == 1 ...
```

Až na to, že v neskoršom prípade je čítaný len odkaz na nulu, vrátený iba ak index i je mimo rozsahu. Okrem toho sú na čítanie jednoúčelové obvyklé prístupy $LeadCoeff(f)$ a $ConstTerm(f)$. NTL poskytuje veľké množstvo operácii pre polynómy celého čísla, ako operátora tak aj procedurálnej formy. Všetky základné operácie podporujú "*promotionlogic*" podobne pre ZZ, až na to, že vstupy oboch typov *long* a ZZ sú povýšene do ZZX.