Vektory v NTL

Obsah

1 Moduly na prácu s vektormi a ich základné funkcie

Na prácu s vektormi je možné v NTL použiť viacero modulov. Každý z nich má k dispozícii iné funkcie a je vhodný na použitie na iný účel. Základné moduly ktoré pracujú s vektormi sú *vec_ GF2, vec_ RR, vec_ ZZ.*

- `GF2`: celé čísla mod 2
- `RR`: reálne čísla
- `ZZ`: veľké celé čísla

Existujú aj ďalšie moduly s ktorými môžeme operovať v rámci vektorov: *vec_ GF2E, vec_ ZZ_ p, vec_ ZZ_ pE, vec_ lzz_ p, vec_ lzz_ pE*

- `ZZ_p`: veľké celé čísla modulo p
- `lzz_p`: celé čísla mod p s jednoduchou presnosťou (single precision)
- `ZZ_pE`: rozšírenie poľa/okruhu nad ZZ_p
- `lzz_pE`: rozšírenie poľa/okruhu nad lzz_p
- `GF2E`: rozšírenie poľa/okruhu nad GF2

1.1 Deklarácia

```
vec_T v;
```

//vytvorenie vektora dĺžky 0.

Vstupné elementy sú inicializované použitím štandardného T konštruktora

```
vec_T(INIT_SIZE_TYPE, long n);
```

1.2 SetLength

Aktuálne nastavenie dĺžky n vektora v vykonáva

```
v.SetLength(n)
```

1.3 length

Pre prístup k aktuálnej dĺžke vektora nám slúži funkcia

```
v.length()
```

, kde i-tý element vektora (počítaný od 0) je prístupný ako $v[i]$, alebo (počítaný od 1) je prístupný ako $v(i)$.

Nech n = *v.length()*. Zavolaním funkcie *v.SetLength(m)* s $m <= n$ nastaví aktuálnu dĺžku v na m (ale nevolajú sa žiadne deštruktory ani sa neuvoľni žiaden priestor). Volaním *v.SetLength(m)* s $m > n$ bude alokované miesto a inicializované ako potrebné, ale hodnoty už alokovaných elementov nezmení (aj keď ich adresy sa zmeniť môžu). Inicializácie sú vykonané použitím štandardného T konštruktora.

1.4 SetMaxLength

```
void SetMaxLength(long n);
```

v.SetMaxLength(n) vyhradí najväčšií možný priestor za inicializovanými n elementami. Zavolaním *v.length()* sa zobrazí honota len inicializovaných elementov.

1.5 MaxLength

```
long MaxLength() const;
```

v.MaxLength(), zobrazuje najväčšiu možnú hodnotu n (t.j.počet inicializovaných elementov a maximálneho prideleného miesta), ktorú je možné inicializovať volaním *v.SetLength(n)*.

1.6 deštruktor

Ak je volaný v deštruktor

```
~vec_T();
```

, všetky vytvorené elementy budú zničené a všetko miesto pola bude zrušené. Miesto pola v pamäti je riadene použitím *malloc, realloc a free*.

Pri zväčšovaní vektora sa znovu prideľuje miesto použitím *realloc*, pričom sa môže meniť aj adresovanie elementov vektora, vytvorením premenlivých vyjadrení elementov vektora. Na to musíte byť zvlášť opatrní pri použití vektorov schválených ako odkazy na parametre, ktoré môžu nahrádzať iné.

1.7 allocated

```
long allocated() const;
```

v.allocated() zobrazuje počet elementov, ktoré boli alokované. Ten môže byť väčší ako počet elementov, ktoré boli inicializované.

1.8 elts

```
const T* elts() const;
```

T elts()* vracia adresu prvého elementu vektora (alebo 0, ak pre daný vektor nie je možné vytvoriť miesto).

1.9 kill

```
void kill();
```

v.kill() uvolní pamäť a nastaví dĺžku vektora = 0.

1.10 FixLength

```
void FixLength(long n);
```

v.FixLength(long n) nastaví fixnú dĺžku vektora.
Pre zrušenie je potrebná funkcia *kill*().

```
long fixed() const;
```

fixed testuje či dĺžka vektora bola fixovaná funkciou *FixLength()*.

1.11 RawGet

```
T& RawGet(long i) const;
```

RawGet robí indexovanie bez kontroly rozsahu.

1.12 position

```
long position(const T& a) const;
```

position vráti pozíciu *a* vektora ak sa tam nachádza a -1 ak sa tam nenachádza.

1.13 swap

```
void swap(vec_T& x, vec_T& y);
```

swap - výmena x & y výmenou ukazovateľov

1.14 append

```
void append(vec_T& v, const T& a);
```

append - pripojenie *a* za koniec *v*

2 Vstup/Výstup (input/output)

I/O operátory môžu byť deklarované použitím *NTL_io_vector_decl(T,vec_T)*, a implementované použitím *NTL_io_vector_impl(T,vec_T)*. Elementy sú čítané a zapisované použitím základných I/O operátorov << a >> pre T.

I/O format pre vector v s n elementmi je:
`[v[0] v[1] ... v[n-1]]`

```
istream& operator>>(istream&, vec_T&);
ostream& operator<<(ostream&, const vec_T&);
```

3 Aritmetické operácie

Knižnica NTL poskytuje aritmetické operácie ako: sčítanie, odčítanie, negácia, násobenie, vektorový a skalárny súčin a kopírovanie vektora dĺžky n. Tieto operácie sú spoločné pre všetky moduly, ktoré v NTL slúžia na reprezentáciu vektorov.
Všetky názorné príklady budú znázornené modulom vec_RR. Funkcie, ktoré budú vykonávať odlišné operácie od názorných príkladov, budú uvedené samostatne pre daný modul.

3.1 Sčítanie, odčítanie a negácia

Na sčítanie a odčítanie vektorov slúžia klasické operátory $+, -, +=, -=$. Môžu sa použiť aj funkcie *add*, *sub*, pri negácii sa používa funkcia *negate*.

3.1.1 add

```
void add(vec_RR& x, const vec_RR& a, const vec_RR& b);
// x = a + b
```

Do premennej *vec_RR x*, priradí súčet vektorov *vec_RR a* + *vec_RR b*.

3.1.2 sub

```
void sub(vec_RR& x, const vec_RR& a, const vec_RR& b);
// x = a - b
```

Do premennej *vec_RR x*, priradí rozdiel vektorov *vec_RR a* - *vec_RR b*.
Funkcia *sub* v moduloch *vec_GF2* a *vec_GF2E* vykonáva:

```
void sub(vec_GF2E& x, const vec_GF2E& a, const vec_GF2E& b);
// x = a - b = x + a
```

3.1.3 negate

```
void negate(vec_RR& x, const vec_RR& a);
// x = -a
```

Do premennej *vec_RR x*, priradí negáciu vektora *vec_RR a*.
Funkcia *negate* v moduloch *vec_GF2* a *vec_GF2E*:

```
void negate(vec_GF2E& x, const vec_GF2E& a);
// x = - a = a
```

znaméná to, že pre daný modul nevykonáva žiadnu operáciu. Funkcia *negate* je zbytočná!

3.2 Násobenie, vektorový súčin, skalárny súčin

Na násobenie a vektorový súčin vektorov sa používa operátor $*, *=$ alebo funkcia *mul*.

3.2.1 Násobenie

```
void mul(vec_RR& x, const vec_RR& a, const RR& b);
// x = a * b
```

3.2.2 Vektorový súčin

```
vec_RR operator*(const vec_RR& a, const RR& b);
```

3.2.3 Skalárny súčin

Pri skalárnom súčine je možné použiť funkciu *InnerProduct*,

```
void InnerProduct(RR& x, const vec_RR& a, const vec_RR& b);

//dĺžka vektorov môže byť rôzna, kde tento rozdiel dĺžok
//na výstupe bude zaplnený nulou.
//x = skalárny súčin a * b
```

3.2.4 Kopírovanie vektora dĺžky n

Kopírovanie vektora sa deje na základe funkcie $VectorCopy$

```
void VectorCopy(vec_RR& x, const vec_RR& a, long n);
```

$x = a$, kopírované z vektora a dĺžky n. Ak je potrebné, vstup je skrátený, alebo zaplnený nulou.

4 Porovnávanie vektorov

Knižnica NTL poskytuje na porovnananie dvoch vektorov dva klasické operátory == (rovnosť), ! = (nerovnosť).

```
NTL_eq_vector_decl(RR,vec_RR)
// == and !=
```

Ďalej existuje funkcia $IsZero$

```
long IsZero(const vec_GRR& a);
```

funkcia $IsZero$ vracia 1, ak vektor obsahuje 0 elementy, inak vracia 0.

5 Utility routines

V tejto časti sa nachádza niekoľko užitočných funkcií, ktoré môžu byť potrebné pri práci s vektormi. Sú tu obsiahnuté rôzne funkcie, ako napr. mazanie vektora, posun vektora o n miest, náhodný vektor dĺžky n, obrátený vektor, alebo množstvo nenulových elementov vektora.

5.1 clear

```
void clear(vec_RR& x);
```

clear vymaže všetky elementy vektora, dĺžka sa nezmení.

5.2 shift

```
void shift(vec_GF2& x, const vec_GF2& a, long n);
vec_GF2 shift(const vec_GF2& a, long n);
```

$x = a$ posunuté o n miest, kde n môže byť kladné alebo záporné. Všeobecne, $x[i] = a[i - n]$, tak kladné n robí posun indexu smerom nahor. Dĺžka vektora x je rovnaká ako dĺžka vektora a, a teda ak je potrebné, elementy sú zaplnené nulou alebo skrátené.

5.3 reverse

```
void reverse(vec_GF2& x, const vec_GF2& a); // c = obrátené a
```

5.4 weight

```
long weight(const vec_GF2& a);
```

funkcia *weight* vráti množstvo nenulových elementov vektora a.

5.5 random

```
void random(vec_GF2& x, long n);  // x = náhodný vektor dĺžky n
vec_GF2 random_vec_GF2(long n);
```