Vektory a matice

Obsah

1 Vektory v NTL

1.1 Prvý príklad

Prvý príklad používa triedu vec_ZZ, ktorá reprezentuje "veľké celé čísla": čísla ľubovolnej dĺžky

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

NTL_CLIENT

ZZ sum(const vec_ZZ& v)
{
   ZZ acc;

   acc = 0;

   for (long i = 0; i < v.length(); i++)
      acc += v[i];

   return acc;
```

Trieda `vec_zz` je dynamická dĺžka poľa ZZ. Obecnejšie povedané, pre vektory v NTL existuje šablóna `vec_T`, ktorá nahrádza makra, pre tvorenie dynamickej dĺžky vektorov cez nejaký typ T napr.(`RR, ZZ, ZZ_p, zz ...`). Dôvod, že makra sú používané namiesto pravých šablón je jednoduchý: v dnešnej dobe, podpora kompilátora pre šablóny nie je úplne postačujúca, a ich použitie by NTL oveľa viac znáročnilo portovanie. V budúcnu by však takáto verzia šablón mohla byť sprístupnená.

Vektory v NTL sú indexované od 0, ale vo viacerých situáciách je vhodnejšie, alebo viacej prirodzenejšie indexovať od 1. Všeobecná trieda vektora s tým počíta, preto hore uvedený príklad by mohol byť napísaný takto:

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

NTL_CLIENT

ZZ sum(ZZ& s, const vec_ZZ& v)
{
   ZZ acc;

   acc = 0;

   for (long i = 1; i <= v.length(); i++)
      acc += v(i);

   return acc;
}
```

Všimnite si, že štandardne NTL nevykonáva rozsahové kontroly na registroch vektorov. Je tam znamenie času kompilátora, ktorý aktivuje kontrolu rozsahu. Preto je osvedčená metóda vždy predpokladať, že rozsah kontroly môže byť aktivovaný, a nesprístupni elementy, ktoré sú mimo rozsahu.

1.2 Druhý príklad

Príklad ilustruje `I/O` vektora, rovnako ako výmenu dĺžky vektora. Tento program načita vektor typu `vec_ZZ`, a následne vytvorí a vypíše `"palindrome"`

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

NTL_CLIENT

int main()
{
   vec_ZZ v;
   cin >> v;

   long n = v.length();
   v.SetLength(2*n);

   long i;
   for (i = 0 ; i < n; i++)
      v[n+i] = v[n-1-i];

   cout << v << "\n";
}
```

Všimnite si zmenu dĺžky vektora pri nezmenenom obsahu.
Keď vstupom budú hodnoty

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

Na výstupe dostaneme

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

NTL vopred definuje typy vektorov (v našom prípade to bol vec_ZZ). Naviac je možné definovať si vlastne typy.

2 Matice v NTL

Trieda `mat_T` je špeciálny druh z `vec_vec_T`, kde každý riadok je vektor rovnakej dĺžky. Riadok i matice M môže byť sprístupnený ako $M[i]$ (s indexom od 0), alebo ako $M(i)$ (s indexom od 1). Stĺpec j z rady i môže byť sprístupnený ako $M[i][j]$ alebo $M(i)(j)$, pre neskorší, lepší zápis je ekvivalentn0 s $M(i, j)$.

2.1 Príklad

Nasledujúci program vykonáva násobenie matic, ktoré je aj poskytované knižnicou NTL:

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

NTL_CLIENT

void mul(mat_ZZ& X, const mat_ZZ& A, const mat_ZZ& B)
{
   long n = A.NumRows();
   long l = A.NumCols();
   long m = B.NumCols();

   if (l != B.NumRows())
      Error("matrix mul: dimension mismatch");

   X.SetDims(n, m); // make X have n rows and m columns

   long i, j, k;
   ZZ acc, tmp;

   for (i = 1; i <= n; i++) {
      for (j = 1; j <= m; j++) {
         acc = 0;
         for(k = 1; k <= l; k++) {
            mul(tmp, A(i,k), B(k,j));
            add(acc, acc, tmp);
         }
         X(i,j) = acc;
      }
   }
}
```

V prípade nezhody formátov, sa obvykle zavolá funkcia Error, ktorá je súčasťou NTL a ktorá jednoducho vytlačí správu a ukončí beh programu. Je to obecné pre NTL ako jednať s chybami.

Zavolanie obvyklého násobenia môžeme zapísať ako:

```
mul(X, A, B);
```

alebo môže byť použitý matematický operátor:

```
X = A * B;
```