Poznámky a odporúčania k implementovaniu programov v NTL

Obsah

1 Úvod

Tento krátky dokument slúži ako pomôcka k implementovaniu programov v NTL v zmysle vyhnúť sa chybám, ktoré sa vyskytli počas implementovania pomocných programov k predmetu *Rýchle algoritmy* a pri odskúšavaní jednotlivých funkcií v okruhoch NTL.
Než začneme implementovať nejaký problém v NTL, by sme si mali skontorlovať, či funkcie, ktoré budeme potrebovať, či vôbec existujú v NTL a keď existujú, či sú preťažené aj nad triedou/okruhom, v ktorej mienime pracovať. Ak neexistujú ešte netreba zúfať, totiž sú konverzie, ktoré môžu vyriešiť tento problém, hoci nie v každom prípade. Čiže sa odporúča oboznámiť sa s dokumentom `conversions.txt`, ktorý opisuje všetky dovolené konverzie.

2 Závažné chyby

- Ak zabudneme pridať do projektu vytvorenú NTL knižnicu **ntl.lib**. Po debugovaní sa nám objavia linkovacie chyby typu:

  ```
  error LNK2019: unresolved external symbol ...
  error LNK2019: ...
  ```

- Ak zabudneme do `main.cpp` pod hlavičky napísať **NTL_CLIENT**. Po debugovaní sa nám objavia chyby typu:

  ```
  error C2065: 'x' : undeclared identifier
  error C2065: 'y' : undeclared identifier
  ```

  lebo sa nerozoznali typy knižnice NTL, ako `ZZ, ZZ_p, atď.`, ktorými boli premenné `x` a `y` deklarované.

- Podobná predošlej chybe je keď zabudneme zavolať potrebný hlavičkový súbor nejakého okruhu, ako napríklad:

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

  potom dostaneme, chybové hlásenia typu:

  ```
  error C2065: 'x' : undeclared identifier
  error C2065: 'y' : undeclared identifier
  ```

  ale tieto premenné sú zadeklarované v okruhoch, ktorých hlavičkové súbory neboli zavolané.

- Keď nie je pridaná cesta k hlavičkovým súborom v **Addional include directories** dostaneme chybové hlásenie typu:

```
fatal error C1083: Cannot open include file:
'NTL/ZZ.h': No such file or directory
```

3 Odporúčania a poznámky

- V prípade použitia niektorých funkcií z okruhu faktorizácie, ak nedodržíme predpísané predpoklady týchto funkcií na vstupe, môže nastať prípad, že sa nedočkáme výsledku.

  Príklad:

  Napríklad pri použití nasledujúcej funkcie, ak je vstupný polynóm ireducibilný alebo obsahuje ireducibilný faktor.

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

- Môže nastať prípad, že potrebujeme funkciu, ktorá nie je preťažená v triede v ktorej mienime pracovať a ani premenné sa nedajú skonvertovať z typu s ktorým pracujeme, na taký, ktorý daná funkcia vyžaduje.

  Príklad:

  Začali sme implementovať problém v `ZZ_p` a potrebujeme použiť funkciu na odmocnenie

```
SqrRootMod(ZZ& x, const ZZ& a, const ZZ& n)
```

  ktorá nie je preťažená v `ZZ_p` a ani nič podobné v nej neexistuje a navyše sa nedá skonvertovať `ZZ_p` na `ZZ`. Keďže sme nenašli iné riešenie museli sme upustiť pracovanie v triede `ZZ_p` a namiesto toho sme museli pracovať v `ZZ` a na potrebných miestach modulovať.

- Ak použijeme funkciu

```
void inv(ZZ& d, mat_ZZ& X, const mat_ZZ& A, long deterministic=0)
```

  v okruhu `mat_ZZ` na vypočítanie inverznej matice, by sme chceli upozorniť na skutočnosť, napriek tomu, že je to v popise funkcie napísané, že ak `d` determinant matice `A` sa nerovná nule $d! = 0$, potom $A * X = d * I$, kde matica `X` je vypočítaná inverzná matica k matici `A` a matica `I` je jednotková matica.

  Príklad:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} -12 & 0 & 0 & 0 & 0 \\ -6 & -4 & 12 & -2 & 24 \\ 12 & -6 & -6 & 0 & 12 \\ 6 & -2 & -6 & 2 & -24 \\ 0 & 0 & 0 & 0 & -12 \end{pmatrix} = (-12) * \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

kde $d = (-12)$ je determinant matice `A`.

- Keď sme nútený pracovať v okruhu `mat_ZZ`, a chceme počítať modulo matice tak je to potrebné spočítať modulo matice po prvkoch, lebo funkcia nato neexistuje.

  Príklad: Takýto problém nastal, keď sme potrebovali počítať odmocninu z jednotlivých prvkov matice, čiže kvôli funkcii na odmocnenie sme museli zvoliť maticu typu `ZZ`. A keďže sme potrebovali spočítať modulo `p` danej matice, sme zistili, že funkcia na modulovanie matice neexistuje.

- Ak sa pracuje v okruhu `RR` s veľkými číslami, treba si uvedomiť, že je nastavená predvolená presnosť výstupu na 10 číselných miest, ktorá keď sa presiahne tak sa výsledok zaokrúhluje. Na prenastavenie tejto presnosti slúži metóda `SetOutputPrecision`.

  Príklad:

```
RR result;
result.SetOutputPrecision(20); \\predvolenu presnost vysledku
             \\sme zmenili na presnost 20-ich ciselnych miest
```

- Pri implementovaní programov často potrebujeme počítať *logaritmus* nejakej hodnoty. V NTL v okruhu `ZZ` je funkcia na počítanie *desiatkového logaritmu*, ktorú jednoduchým prepisom rádu logaritmu ľahko zmeníme, na logaritmickú funkciu ľubovoľného rádu.

  Príklad: Funkciu definovanú v NTL...

```
double Log(double x)
{
// pocita desiatkovy logaritmus ...

   static double log10 = log(10);
   return log(x)/log10;
}
```

  nasledovným spôsobom prepíšeme na dvojkový logaritmus.

```
double Log2(double x)
{
// pocita dvojkovy logaritmus ...

   static double log2 = log(2.0);
   return log(x)/log2;
}
```