Number Theory Library

NTL

Выполнила Капитанюк Светлана
13604/1

Библиотека NTL была разработана Виктором Шоупом.
(Нью-Йорк, США).
Разработка библиотеки была начата в 1990 году.
Последняя версия была выпущена 13.11.2015.

30.10.2015 автор NTL был удостоен Richard Dimick Jenks Memorial Prize за выдающиеся достижения в разработке программного обеспечения, за разработку NTL.

NTL является высокопроизводительной библиотекой C++, разработанной для работы с теорией чисел.

В NTL рассматриваются:

- Структуры данных и алгоритмы для чисел произвольной длины;
- Матрицы;
- Вектора;
- Полиномы;
- Конечные поля;
- Базовая линейная алгебра;

Так же в библиотеке существует возможность работы с арифметикой произвольной точности для чисел с плавающей точкой.

Алгоритмы, работающие с полиномами в NTL являются одними из самых быстрых, по сравнению с другими библиотеками. И именно поэтому данную библиотеку использовали для установления "мирового рекорда" разложения полинома.

Части рабочего интерфейса:

- Для работы с кольцами *(Basic Ring Classes,Residue class rings and modulus switching)*;
- Для чисел с плавающей запятой *(Floating Point Classes)*;
- Для работы с векторами и матрицами *(Vectors and Matrices)*;
- Для работы с функциональными и процедурными формами *(Functional and Procedural forms)*;
- Для преобразований *(Conversions and Promotions)*;
- Для ввода/вывода *(Input and Output)*;
- Для сглаживания вывода *(Aliasing)*;
- Для конструкторов,деструкторов и работы с памятью *(Constructors, Destructors, and Memory Management)*;
- Для обработки ошибок и исключений *(Error Handling and Exceptions)*;

Некоторые классы (модули):

- **ZZ**: большие целые числа;
- **ZZ_p**: целые числа по модулю p;
- **zz_p**: большие числа *одинарной точности* по модулю p;
- **GF2**: целые числа по модулю 2;
- **ZZX**: одномерные многочлены для ZZ;
- **ZZ_pX**: одномерные многочлены для ZZ_p
- **zz_pX**: одномерные многочлены для zz_p
- **ZZ_pE**: расширенное поле/кольцо для ZZ_p

Модуль ZZ

В основном используется для записи чисел произвольной длины.

Может быть использован для работы с конструкторами, деструкторами, для вычисления арифметической точности.

Пример для вычисления: (a+1)*(b+1)

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

using namespace std;
using namespace NTL;

int main()
{
   ZZ a, b, c;

   cin >> a;
   cin >> b;
   c = (a+1)*(b+1);
   cout << c << "\n";
}
```

Присвоение значения для ZZ:

```
ZZ x;
x = 1;
```

```
ZZ x = ZZ(1);
ZZ y(1);
ZZ z{1};
```

```
ZZ x = conv<ZZ>(1);
```

```
ZZ x = conv<ZZ>("99999999999999999999");
```

Нельзя записать это в таком виде:

```
ZZ x = 1;
```

Преобразование типов из ZZ в long и обратно:

```
ZZ z1 = ZZ(2);
ZZ z2;
long a = 0;
conv(a, z1); //converts zz type to long

conv(z2,a); //converts long to zz type
```

Модуль RR

Используется для представления произвольной точности числа с плавающей точкой

Возможность округления (указания точности вычисления) числа через специальные функции:
RR::SetPrecision ();
RR::precision();

Минимальная точность может быть установлена на 53 бита, максимальная — не ограничена. В данном классе, сначала вычисляется значение, а затем уже происходит округление до указанного значения.

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

using namespace std;
using namespace NTL;

int main()
{
   RR acc, val;

   acc = 0;
   while (cin >> val)
      acc += val*val;

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

Модуль xdouble

Этот класс используется для представления чисел с плавающей запятой двойной точности; практически не отличается от обычного типа double, но с более расширенным диапазоном показателя

Модуль quad_float

Используется для представления чисел с плавающей запятой четверной точности без расширенного диапазона

Наиболее эффективны по сравнению с RR

Вектора и Матрицы

Программа для вычисления суммы чисел в векторе:

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

using namespace std;
using namespace NTL;

ZZ sum(const Vec<ZZ>& v)
{
   ZZ acc;

   acc = 0;

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

   return acc;
}
```

Для задания векторов используются следующие классы:

- **Vector -** Класс для динамического создания векторов;

  Vec<T> v — нулевой вектор по определению;
  v.SetLength(n) — вектор длины n;

- **vec_GF2 -** Вектор для работы с числами GF2;
- **vec_GF2E -** Вектор для работы над полем чисел GF2;
- **vec_RR -** Вектор для работы с числами RR;
- **vec_ZZ -** Вектор для работы с числами ZZ;
- **vec_ZZ_p -** Вектор для работы с числами ZZ_p;
- **vec_ZZ_pE -** Вектор для работы над полем чисел ZZ_pE;

Также есть возможность обозначать вектора как: **v[i]** или **v(i)**, но разница будет заключаться в том, что в v[i] отсчет будет идти с 0, а в другом случае — с 1.

Программа для перемножения матриц:

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

using namespace std;
using namespace NTL;

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();

   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;
      }
   }
}
```

Для задания матриц используют следующие классы:

- **matrix** - Класс для создания двумерных динамических массивов;

  Mat<T> M- создание нулевой матрицы;
  M.SetDims(A,B); - создание матрицы A x B, где A — строки, а B — столбцы;

- **mat_GF2** - Матрицы для работы над числами GF2; Включает в себя основные арифметические действия с матрицами и работу с методом Гаусса;
- **mat_RR** - Матрицы для работы над числами RR; Включает в себя работу с основными арифметическими операциями над матрицами, вычисление определителей и линейно независимых уравнений;
- **mat_ZZ** - то же, что и для mat_RR;
- **mat_poly_ZZ** - для вычисления многочлена из mat_ZZ;

  В NTL матрицы скорее представлены в виде Vec< Vec< T > >.

  Так же может быть использованы записи M[i][j] или M(i)(j), где для последней записи могут использовать M(i,j).

Функциональные и процедурные формы

Функциональная форма:

x = a + b;

x = conv<ZZ>(a);

Процедурная форма:

add(x, a, b);

conv(x, a);

Применение процедурной формы может быть *более эффективно*, так как она избегает создания временного объекта для хранения результата.

Однако функциональная форма, как правило, предпочтительнее, поскольку в результате код *легче понять*.

Сглаживание

Важной особенностью является то, что NTL «сглаживает» ввод и вывод объектов.
Например, если написать программу **mul(x,a,b)**, где **b** будет иметь одинаковый адрес с другим из объектов, или, например **x** уже будет задействован где-либо ещё, как переменная или самостоятельный объект (вектор/многочлен и т.д), то программа может работать корректно и в данных ситуациях.

Есть одно исключение: вход и выход у матриц и векторов не должен совпадать.

Обработка исключений и ошибок

- ErrorObject → std::runtime_error — базовый класс для ошибок;
- LogicErrorObject → ErrorObject — класс для обработки логических ошибок, выход за пределы диапазона и т. д.;
- ArithmeticErrorObject → ErrorObject — для арифметических ошибок, например, деление на 0;
- ResourceErrorObject → ErrorObject — используется для ошибок переполнения памяти;
- FileErrorObject → ErrorObject — для указания проблем с открытием/закрытием файлов;
- InputErrorObject → ErrorObject — для указания проблемы при чтении из потока.

Многочлены

Основной класс для использования — **ZZX.**

Класс ZZX используется для одномерных многочленов с целыми коэффициентами.

Класс **Pair<S,T> p** используется для вычисления пары чисел S и T от p.

Пример для чтения многочлена, его коэффициентов и разложение на множители:

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

using namespace std;
using namespace NTL;

int main()
{
    ZZX f;

    cin >> f;

    Vec< Pair< ZZX, long > > factors;
    ZZ c;

    factor(c, factors, f);

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

Выводы:

- ✔ Библиотека NTL всё ещё разрабатывается;
- ✔ Возможность самостоятельной разработки тех или иных интерфейсов для библиотеки, с возможностью последующей реализации;
- ✔ Легкость в использовании и простота установки;
- ✔ Быстрая работа с многочленами, по сравнению с другими библиотеками (MAGMA, FLINT...);
- ✔ Огромная точность вычислений, работа со сверхбольшими числами;
- ✔ Регулярное наличие обновлений;

- ✘ Некоторые стандартные функции и операторы не реализованы либо полностью, либо работают только с частью интерфейса (% и %= не реализованы для таких классов как:ZZ_p, zz_p, GF2, ZZ_pE, zz_pE, GF2E);
- ✘ Не всегда удобный синтаксис использования функций, некоторые из них не работают без других (FindRoots() — функция сама по себе будет работать, но довольно-таки медленно, быстрее с CanZass())

Ссылки:

1. www.csd.uwo.ca/~eschost/publications/fp097-li.ps - ссылка на документацию по исследованию разложения полинома;
2. http://www.shoup.net/ntl/ - официальный сайт библиотеки NTL;
3. http://www.shoup.net/ - сайт разработчика библиотеки NTL;
4. http://www.shoup.net/ntl/doc/tour.html - ссылка на документацию;