acm.mipt.ru

олимпиады по программированию на Физтехе

Быстрое преобразование Фурье (код на C++)

Файл test.cpp содержит простую программу, которая считывает из потока ввода два неотрицательных целых числа произвольной длины и печатает их произведение. Произведение вычисляется за время O(N*log N), где N – длина результата. Это достигается за счет использования *дискретного преобразования Фурье*. Здесь реализовано дискретное преобразование Фурье в той форме, в которой оно описано в книге Кормена. Преобразование Фурье реализовано отдельно в файле dft.cpp

Файл dft.h

```
/*
 * Fast Fourier Transform (with complex numbers)
 *
 * Daniel Shved, MIPT, 2010.
 * danshved [at] gmail.com
 */

#ifndef __DFT_H__
#define __DFT_H__

#include <complex>
using namespace std;

typedef complex<double> comp;

// Gets the complex conjugate of every element
void conjugate(comp *array, int size);

// Multiplies two vectors element by element
void multiply(comp *arr1, comp *arr2, comp *result, int size);

// Finds the convolution of two vectors
void convolution(comp *arr1, comp *arr2, comp *result, int size);

// Discrete fourier transform. size must be a power of 2
void fourier_transform(comp *array, int size);

// Inverse fourier transform. size must be a power of 2
void inverse_fourier_transform(comp *array, int size);

#endif
```

Файл dft.cpp

```
/*
 * Fast Fourier Transform (implementation)
 *
 * Daniel Shved, MIPT, 2010.
 * danshved [at] gmail.com
 */
```

Поиск
Поиск
Раздел «Алгоритмы»
Главная
Форум
Ссылки
El Judge
Инструменты:
Поиск
Изменения
Index
Статистика
Разделы
Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C
Logon>>

```
#include "dft.h"
#include <math.h>
#include <complex>
#include <algorithm>
#include <stack>
using namespace std;

/*
 * "Butterfly" transform.
 */
inline void butterfly(comp &x, comp &y, comp w)
{
   comp p = x, q = y*w;
   x = p + q;
   y = p - q;
}

/*
 * Series of butterfly transforms required by the FFT algorithm.
 */
inline void mass_butterfly(comp *array, int size, comp w)
{
   comp power(1.0, 0.0);
   int n = size/2;

   for(int i = 0; i < n; i++) {
      butterfly(array[i], array[i+n], power);
      power *= w;
   }
}

/*
 * Given a number ``x'' returns the number which has the same bits as ``x'',
 * but in the reverse order
 */
inline unsigned int backwards(unsigned int x, int length)
{
   unsigned int result = 0;
   unsigned int bit = 1u;
   unsigned int reverse = 1u<<(length-1);
   for(int i = 0; i < length && x != 0; i++) {
      if(x & bit) {
         result |= reverse;
         x &= ~bit;
      }
      bit <<= 1;
      reverse >>= 1;
   }
   return result;
}

/*
 * Moves elements of the array as required by the iterative FFT implementation.
 * ``size'' must be a power of 2.
 */
static void reposition(comp *array, int size)
{
   // Determine the bit length
   int length = 0;
   while(1u << length < (unsigned int)size)
      length++;

   // Swap elements at positions k and reverse(k)
   for(int i = 0; i < size; i++) {
      int j = backwards(i, length);
      if(i <= j)
         swap(array[i], array[j]);
   }
}
```

```
/*
 * Does the Discrete Fourier Transform.  Takes time O(size * log(size)).
 * ``size'' must be a power of 2.
 */
void fourier_transform(comp *array, int size)
{
   // Arrange numbers in a convenient order
   reposition(array, size);

   // Prepare roots of unity for every step
   int step;
   comp root = exp(comp(0.0, 2.0*M_PI/size));
   stack<comp> roots;
   for(step=size; step != 1; step /= 2) {
      roots.push(root);
      root *= root;
   }

   // Do lots of butterfly transforms
   for(step = 2; step <= size; step *= 2) {
      root = roots.top();
      roots.pop();
      for(int i = 0; i < size; i += step)
         mass_butterfly(array + i, step, root);
   }
}

/*
 * The inverse DFT.
 */
void inverse_fourier_transform(comp *array, int size)
{
   conjugate(array, size);
   fourier_transform(array, size);
   conjugate(array, size);
   for(int i = 0; i < size; i++)
      array[i] = array[i] / (double)size;
}

/*
 * Replaces every element of the vector by its complex conjugate.
 */
void conjugate(comp *array, int size)
{
   for(int i = 0; i < size; i++)
      array[i] = conj(array[i]);
}

/*
 * Multiplies two vectors element by element.
 */
void multiply(comp *arr1, comp *arr2, comp *result, int size)
{
   for(int i = 0; i < size; i++)
      result[i] = arr1[i] * arr2[i];
}

/*
 * Finds the convolution of two vectors (the product of two polynomials, given
 * that the result has power less than ``size'').  ``size'' must be a power of
 * 2.
 */
void convolution(comp *arr1, comp *arr2, comp *result, int size)
{
   fourier_transform(arr1, size);
   fourier_transform(arr2, size);
   multiply(arr1, arr2, result, size);
```

```
   inverse_fourier_transform(result, size);
}
```

Файл test.cpp

```
/*
 * Long multiplication with the FFT.
 *
 * Daniel Shved, MIPT, 2010.
 * danshved [at] gmail.com
 */
#include "dft.h"
#include <string>
#include <iostream>
using namespace std;

const int base = 10;

/*
 * Turns a string with a number into an array of complex numbers (digits)
 */
comp *make_polynomial(const string &str, int length)
{
   comp *result = new comp[length];
   for(int i = 0; i < str.length(); i++)
      result[i] = comp((double)(str[str.length() - 1 - i] - '0'), 0.0);
   for(int i = str.length(); i < length; i++)
      result[i] = comp(0.0, 0.0);
   return result;
}

/*
 * This is used to turn complex numbers which are known to be almost real
 * integers into integers.
 */
int round(comp x)
{
   return (int)(x.real()+0.5);
}

/*
 * Given a polynomial ``a'', returns the string with number a(10).
 */
string make_number(comp *a, int n)
{
   // Turn coefficients into digits (carry)
   for(int i = 0; i < n - 1; i++) {
      a[i+1] += round(a[i]) / base;
      a[i] = round(a[i]) % base;
   }

   // Determine length
   int realLength = 0;
   for(int i = 0; i < n; i++)
      if(round(a[i]) != 0)
         realLength = i+1;
   if(realLength == 0)
      return string("0");

   // Make a string
   string result(realLength, '0');
   for(int i = 0; i < realLength; i++)
      result[realLength - 1 - i] = '0' + (char)round(a[i]);
   return result;
}

/*
 * Long multiplication.
```

```
 */
string multiply(string number1, string number2)
{
   // Determine the size of polynomials
   int deg = number1.length() + number2.length();
   int n = 1;
   while(n<=deg) n*= 2;

   // Turn numbers into complex polynomials
   comp *a = make_polynomial(number1, n);
   comp *b = make_polynomial(number2, n);

   // Multiply polynomials with the FFT
   comp *product = new comp[n];
   convolution(a, b, product, n);

   // Turn the polynomial back into a number
   string result = make_number(product, n);

   // Clean up and exit
   delete a;
   delete b;
   delete product;
   return result;
}

/*
 * A test. Reads two non-negative integers from the input and prints their
 * product.
 */
int main()
{
   string a, b, res;
   cin >> a >> b;
   res = multiply(a, b);
   cout << res << endl;
   return 0;
}
```

-- DanielShved - 24 Apr 2010