4.3. Классические алгоритмы умножения и деления чисел и многочленов

Продолжим изучение основных арифметических операций над целыми числами многократной точности и полиномами. Рассмотрим основные алгоритмы выполнения операций умножения и деления чисел и полиномов.

Для понимания работы с целыми числами многократной точности разберем сначала классические алгоритмы умножения и деления чисел. Следует заметить, что во многих системах компьютерной алгебры данные алгоритмы используются в основных режимах их функционирования.

Умножение целых чисел многократной точности “столбиком”

Предположим, что мы имеем дело с неотрицательными целыми числами.

Алгоритм 1. Умножение неотрицательных целых чисел. Заданы два целых числа по основанию b - первое число $u_1u_2u_3\cdots u_n$, второе число $v_1v_2v_3\cdots v_m$. Алгоритм вырабатывает их произведение $w_1w_2w_3\cdots w_{m+n}$.

Шаг 1. Начальная установка.

Установить все значения $w_{m+1}w_{m+2}w_{m+3}\cdots w_{m+n}$ равным нулю.

Установить $j = m$ (индекс цифры второго сомножителя).

Шаг 2. Учет нулевого множителя.

Если $v_j = 0$, установить $w_j = 0$ и передать управление на **Шаг 6**.

Шаг 3. Начальная установка для индекса цифры первого сомножителя.

Установить $i = n$ (индекс цифры второго числа), $k = 0$ (цифра переноса).

Шаг 4. Умножить и сложить.

Установить $t = u_i * v_j + w_{i+j} + k$, затем установить $w_{i+j} = t \bmod b$, $k = \lfloor t / b \rfloor$.

Шаг 5. Цикл по индексу i.

Уменьшить i на единицу. Если $i > 0$, то вернуться в **Шаг 4;** в противном случае установить $w_j = k$.

Шаг 6. Цикл по индексу j.

Уменьшить j на единицу. Если $j > 0$, то вернуться в **Шаг 2;** в противном случае закончить выполнение алгоритма.

Алгоритм умножения двух чисел повторяет обычные действия, производимые при умножении чисел вручную «столбиком».

Теорема. Если считать, что перемножаемые числа имеют одинаковую длину, состоят из n цифр, то трудоемкость приведенного алгоритма умножения чисел можно оценить как $O(n^2)$.

Учет знаков целых чисел производится обычным способом.

Умножение многочленов “столбиком”

Операции сложения, вычитания и умножения полиномов определяются естественным образом.

Для плотного представления полиномов могут использоваться видоизмененные алгоритмы арифметики многократной точности, в том числе, и алгоритм умножения столбиком, с учетом приведенных замечаний.

Для разреженных представлений обычно используются алгоритмы, учитывающие особенности канонической формы хранения, или рекурсивные алгоритмы.

Допустим, что мы работаем с полиномами от одной переменной в разреженном представлении в канонической форме с упорядочиванием мономов по убыванию степени переменной.

Алгоритм 2. Умножение двух полиномов.

Заданы два полинома в виде циклических списков - первый u, второй v.

Шаг 1. Начальная установка. Формируем новый (нулевой) полином r – результат работы программы.

Если один из сомножителей – нулевой полином, то закончить алгоритм. Берем первый, старший моном полинома v, делаем его текущим мономом t.

Шаг 2. Сложение результата с произведением полинома, первого сомножителя, на текущий моном.

Для этих целей используется специальная процедура сложения `AddPoly`, описанная в предыдущей лекции и видоизмененная в процедуру `AddPolyT` с помощью введения умножения второго слагаемого на текущий моном t, который задается в виде параметра процедуры.

Шаг 3. Цикл по мономам полинома v.

Если полином v не исчерпан, то переходим к следующему моному в этом полиноме и делаем его текущим мономом t, после чего передаем управление на **Шаг 2;** в противном случае закончить алгоритм.

Приведенный алгоритм умножения полиномов будет работать и для полиномов от нескольких переменных, если в процедуре `AddPolyT` реализовано умножение на одночлен для полинома, зависящего от нескольких переменных, а в процедуре `InsertMonom` делается вставка такого же типа одночлена. Пример программ умножения полиномов от одной переменной на разных языках программирования приводится ниже.

Теорема. Для полинома *A*, содержащего m одночленов, и полинома *B*, содержащего n одночленов, трудоемкость приведенного алгоритма имеет порядок $O(m^2 * n)$.

Деление “столбиком”

Предположим, что мы имеем дело с неотрицательными целыми числами.

Алгоритм 3. Деление неотрицательных целых чисел. Заданы два целых числа по основанию b; первое число - $u_1u_2u_3\cdots u_{m+n}$, второе число - $v_1v_2v_3\cdots v_n$. Алгоритм вырабатывает частное $q_0q_1\cdots q_m$ от деления этих чисел и остаток $r_1r_2r_3\cdots r_n$.

Шаг 1. Нормализация числа.

Установить $d = \lfloor b/(v_1+1) \rfloor$.

Установить число $u_0u_1u_2\cdots u_{m+n}$ равным числу $u_1u_2\cdots u_{m+n}$, умноженному на число d.

Шаг 2. Начальная установка для индекса j.

Установить $j = 0$.

Шаг 3. Вычисление множителя Q.

Если $u_j = v_1$, установить $Q = b-1$, в противном случае установить $Q = \lfloor (u_j * b + u_{j+1}) / v_1 \rfloor$. Теперь проверить выполняется ли неравенство $v_2 * Q > (u_j * b + u_{j+1} - Q * v_1) * b + u_{j+2}$. Если оно выполнено уменьшить Q на единицу и повторить проверку.

После всех проверок установить $q = Q$.

Шаг 4. Умножить и вычесть.

Заменить $u_j u_{j+1} u_{j+2} \cdots u_{j+n}$ на $u_j u_{j+1} u_{j+2} \cdots u_{j+n}$ минус число $v_1 v_2 \cdots v_n$, умноженное на q.

Шаг 5. Проверка остатка.

Установить $q_j = q$. Если результат предыдущего шага был отрицателен, то перейти на **Шаг 6**, в противном случае перейти на **Шаг 7**.

Шаг 6. Компенсирующее сложение.

Уменьшить q_j на единицу и сложить $0 v_1 v_2 \cdots v_n$ и $u_j u_{j+1} u_{j+2} \cdots u_{j+n}$.

Шаг 7. Цикл по индексу j.

Увеличить j на единицу.

Если теперь $j \le m$, передать управление на **Шаг 3.**

Шаг 8. Денормализация. Теперь $q_0 q_1 \cdots q_n$ есть искомое частное, и для получения искомого остатка достаточно разделить $u_{m+1} \cdots u_{m+n}$ на d.

Алгоритм деления двух чисел повторяет обычные действия, производимые при делении чисел вручную «уголком» с тем отличием, что делитель предварительно делается нормализованным, т.е. достаточно большим, чтобы при делении пробное число (очередная пробная цифра результата) отличалось от числа – результата незначительно, не более чем на единицу. Тогда распознавание этого случая с целью корректировки не представляет большого труда.

Теорема. Если считать, что делимое и делитель имеют одинаковую длину, состоят из n цифр, то трудоемкость приведенного алгоритма деления чисел можно оценить как $O(n^2)$.

4.4. Использование приема "разделяй и властвуй" для умножения чисел и многочленов

Так как арифметические операции, такие как умножение и деление, используются на одном из самых нижних уровней иерархии систем аналитических вычислений, то к ним могут применяться повышенные требования по эффективности вычислений. Разработаны алгоритмы, имеющие лучшие скоростные характеристики по сравнению с классическими алгоритмами. Рассмотрим некоторые из них.

Метод А.Карацубы умножения целых чисел

Для очень больших целых чисел A и B можно построить алгоритм умножения более быстрый, чем классический. Идея этого способа принадлежит А. Карацубе. Она заключается в разбиении исходных чисел A и B на две части.

При этом получим

$$A = A_1 * b^k + A_0,\ B = B_1 * b^k + B_0,$$

где $k = \max(m,n)/2$, m - число цифр по основанию b в числе A, а n - число цифр по основанию b в числе B.

Теперь произведение $C = A * B$ можно вычислить с помощью лишь трех умножений целых чисел длиной k или меньше плюс несколько сдвигов и сложений, используя формулу

$$C = A * B = (A_1 * B_1) * b^{2*k} + (A_1 * B_0 + A_0 * B_1) * b^k + (A_0 * B_0),$$

где $A_1 * B_0 + A_0 * B_1 = (A_1 + A_0) * (B_1 + B_0) - A_1 * B_1 - A_0 * B_0$.

Выигрыш в трудоемкости получается за счет замены «трудоемких» операций умножения операциями сложения и сдвига.

При этом если k само оказывается слишком велико, то можно применить тот же метод для вычисления этих трех меньших произведений. Таким образом, получаем рекурсивный метод – алгоритм умножения больших целых чисел.

Алгоритм 4. Умножение двух целых чисел по методу А. Карацубы. Пусть A и B – два целых числа. Ищется число $C = A * B$.

Шаг 1. Произведение двух «коротких» чисел. Если числа A и B представляются с помощью чисел длиной меньше N цифр (N - число цифр для *обменной точки* алгоритма, когда классический алгоритм становится менее эффективным чем быстрый алгоритм), то ищется произведение чисел A и B классическим способом.

Шаг 2. Соревнование в скорости с классическим алгоритмом. Разбить каждое из сомножителей на две части, старшую и младшую $A = A_1 * b^k + A_0$, $B = B_1 * b^k + B_0$,. После этого вычислить частичные произведения $A_1 * B_1$, $A_1 * B_0 + A_0 * B_1$, $A_0 * B_0$, рекурсивно обращаясь к данному алгоритму.

Шаг 3. Получить результат $C = A * B$, комбинируя для частичных результатов операции сложения и сдвига. Конец алгоритма.

Теорема. Трудоемкость рассмотренного алгоритма умножения можно оценить величиной $O(n^{\log_2 3})$, где n - количество цифр в перемножаемых числах.

Алгоритмы быстрого умножения целых чисел многократной точности

Приведенный выше алгоритм представляет собой просто первый ($r = 1$) модифицированный алгоритм из бесконечной последовательности алгоритмов $M_1, M_2, M_3, \ldots,$.

В алгоритме M_r разбиваем сомножители на r частей таким образом, что

$$A = \sum_{l=0}^{r} A_l * b^{l*k}, \; B = \sum_{l=0}^{r} B_l * b^{l*k},$$

где $k = \max(m,n) / (r+1)$.

Нетрудно получить обобщение вычислительных формул для предлагаемых алгоритмов.

Теорема. Трудоемкость рассмотренного алгоритма умножения можно оценить величиной $O(n^{\log_{r+1}(2*r+1)})$, где n - количество цифр в перемножаемых числах.

Известны и другие более быстрые алгоритмы (и более сложные) умножения целых чисел многократной точности. Из них необходимо отметить класс модулярных алгоритмов и алгоритмов, основанных на быстром преобразовании Фурье. Однако «быстрые алгоритмы» являются быстрыми лишь в случае огромных чисел.

Существует понятие *обменной точки*, т.е. того значения длины числа (количества цифр в числе), при котором быстрый алгоритм начинает выигрывать у классического алгоритма. Эти величины обычно значительны и часто зависят от реализации алгоритма на компьютере.

Алгоритмы быстрого умножения полиномов. Оценка их трудоемкости

В случае плотных представлений разработаны алгоритмы «быстрого» умножения полиномов. Время выполнения рекурсивной версии классического метода умножения двух плотных полиномов (степени n по каждой из v переменных) пропорционально величине n^{2*v}.

Первый быстрый метод умножения плотных полиномов основан на методе Карацубы. Чтобы умножить два полинома от одной переменной степени n, каждый из них разбивают на две части: старшую (степень в которой больше или равна $n/2$) и младшую (степень в которой меньше $n/2$). Легко получить расчетные формулы.

Время выполнения умножения двух плотных полиномов степени n от v переменных с помощью алгоритма, основанного на методе Карацубы, мажорируется величиной $n^{v*\log_2 3}$.

Второй быстрый алгоритм основан на быстром преобразовании Фурье. Время выполнения умножения двух плотных полиномов степени n от v переменных с помощью алгоритма, использующего быстрое преобразование Фурье, пропорционально величине $v*n^v*\log_2 n$.

Обменные точки для всех «быстрых» алгоритмов имеют довольно большое значение.

Реализация операции умножения полиномов в разных языках программирования: LISP, C++, PASCAL

Рассмотрим реализацию алгоритма умножения полиномов от одной переменной в известных языках программирования **LISP и PASCAL**. Необходимо заметить, что при рассмотрении схем реализации обработки полиномов, мы ориентировались, как и в предыдущей лекции, на представление полинома в виде односвязного циклического списка и предположили, что коэффициенты полинома есть целые числа однократной точности, полином хранится в разреженном представлении с упорядочиванием одночленов по убыванию степени одночлена.

В этом пункте мы будем считать, что программы сложения полиномов уже реализованы. (см. программы в предыдущей лекции).

Язык программирования LISP

В языке программирования **LISP** наиболее легко использовать рекурсивные программы. Рассмотренная программа умножения полиномов является продолжением примера рекурсивной работы с полиномами на языке **LISP**, который содержится в книге Дж. Дэвенпорта, И. Сирэ, Э. Турнье.[1]. Если программа сложения полиномов уже реализована в виде функции **ADD-POLY**, тогда программа умножения полиномов будет выглядеть следующим образом

```
(DE MULTIPLY-POLY (A B)
 (COND ((OR (NULL A) (NULL B)) NIL)
       (T (CONS (CONS (PLUS (CAAR A) (CAAR B))
                      (TIMES (CDAR A) (CDAR B)))
                (ADD-POLY (MULTIPLY-POLY (LIST (CAR A))
```

[1] **Дж. Дэвенпорт, И. Сирэ, Э. Турнье. Компьютерная алгебра. - М.: Мир, 1991.**

```
                                       (CDR B))
                         (MULTIPLY-POLY (CDR A) B))))))
```

В приведенной программе умножения полиномов используются следующие функции языка **LISP**, неописанные ранее:

`(OR A B)` – функция, принимающая значение «истина» (`T`), если оба аргумента `A` и `B` принимают значение «истина» (`T`), и принимающая значение «ложь» (`NIL`) – в противном случае;

`(NULL A)` – функция, принимающая значение «истина» (`T`), если аргумент `A` является пустым списком, и значение «ложь» (`NIL`) – в противном случае;

`(TIMES A B)` – функция, результатом которой является *атом*, равный произведению двух чисел – *атомов* А и В.

Теорема. Для полинома A, содержащего m одночленов, и полинома B, содержащего n одночленов, время счета алгоритма **`MULTIPLY-POLY`** имеет порядок $O(m^2 * n)$.

Существуют и более быстрые алгоритмы, но для наглядности приемов программирования на языке **LISP** мы остановились на данной программе.

Особенностью этого алгоритма является широкое использование рекурсии. Полиномы представляются в виде $A = a_0 + a_1$ и $B = b_0 + b_1$, где a_0 и b_0 - старшие одночлены своих полиномов. Вычисления производятся в соответствии с формулой $A * B = (a_0 + a_1) * (b_0 + b_1) = a_0 * b_0 + a_0 * b_1 + a_1 * B$.

Таким образом, произведение двух полиномов переопределяется как сумма термов со старшими одночленами, вычисляемыми по соответствующим формулам, с произведением полиномов более низкой степени по заданной переменной.

При использовании рекурсии необходимо аккуратно относиться к порядку проводимых вычислений, так как от порядка вычислений существенно зависят потребности программы в ресурсах оперативной памяти.

В данной программе для эффективности программы существенен порядок следования слагаемых в формуле вычислений.

Язык программирования C++

Примеры программирования на языке **C++** подробно рассматриваются на практических занятиях (см. программу практикума).

Язык программирования PASCAL

При программировании на языке **PASCAL** необходимо выбрать базовую структуру данных для представления одного звена, одного монома. Допустим, что мономы имеют целочисленные коэффициенты однократной точности, а степень монома целое неотрицательное число. Тогда можно использовать структуру данных для хранения мономов, предложенную в предыдущей лекции.

```
Type TCoeff=Integer;
     TDeg=Integer;
     PMonom=^TMonom;
     TMonom=record
            Coef:TCoeff;
            Deg:TDeg;
            Next:PMonom;
            End;
```

В отдельном модуле (**Unit**) реализуются арифметические операции над полиномом, представимом в виде односвязного циклического списка со звеном, описанном выше. Предлагаем один из вариантов реализации алгоритма умножения двух полиномов с учетом реализации программ **NewPoly**, **DublMonom**, **InsertMonom**, **AddPoly** (см. программы в предыдущей лекции).

```
Procedure AddPolyT (H1, H2, T:PMonom);
  Var P1 ,P2:Pmonom;
  Begin
  P1:=H2^.next;
  While P1^.deg>=0 do
    Begin
    DublMonom (P1,P2);
    P2^.deg:=P2^.deg + T^.deg;
    P2^.coef:=P2^.coef * T^.coef;
    InsertMonom (H1, P2);
    If P2^.next=Nil then Dispose(P2);
    End;
End;

Procedure MultiplyPoly(H1, H2:Pmonom; var R:PMonom);
  Var T:Pmonom;
  Begin
  T:=H2^.next;
  NewPoly(R);
  While T^.deg>=0 do
    Begin
    AddPolyT (R, H1, T);
    T:=T^.next;
    End;
End;
```

В приведенной программе **AddPolyT** (построенной по аналогии уже рассмотренной программы **AddPoly)** производится сложение двух полиномов (первый из которых берется без изменения, а второй умножается дополнительно на моном **T),** результат операции образуется на месте первого слагаемого. Замечание о повышении эффективности расчетов, если в процедуре **InsertMonom** сделать первый параметр выходным, т.е. запоминать последнее место вставки одночлена в первый полином, остается в силе.

Приведенные примеры программ на разных языках показывают относительную сложность программирования структур данных базового уровня и базовых алгоритмов, а также использование рекурсивных алгоритмов и специфики канонических форм хранения.

Теорема. Для полинома A, содержащего m одночленов, и полинома B, содержащего n одночленов, время счета алгоритма **MultiplyPoly** имеет порядок $O(m^2 * n)$.

Резюме

- Основой для построения систем компьютерной алгебры являются выбор представления полиномов от нескольких переменных, целых и рациональных чисел произвольной точности и знание классических алгоритмов базовых операций над этими структурами данных.
- В системах компьютерной алгебры применяются как классические, так и быстрые алгоритмы операций над полиномами и целыми числами многократной точности.
- Реализация быстрых алгоритмов оправдана для операций над очень большими числами и полиномами, содержащих большое число одночленов. В многих системах компьютерной алгебры используются классические алгоритмы умножения.
- Приведенные примеры программ на разных языках показывают относительную сложность программирования структур данных базового уровня и базовых алгоритмов, их трудоемкость, а также использование рекурсивных алгоритмов и специфики канонических форм хранения.

Контрольные вопросы и упражнения:

1. Представление целых чисел.
2. Представление вещественных чисел.
3. Целые числа многократной точности.
4. Позиционные и смешанные системы счисления.
5. Классические операции над числами произвольной точности.
6. Алгоритмы перевода из одной системы счисления в другую систему счисления.
7. Алгоритмы сложения и вычитания целых чисел произвольной точности.
8. Алгоритмы сложения и вычитания полиномов.
9. Реализация операции сложения полиномов в одном из языков программирования.
10. Умножение целых чисел многократной точности “столбиком”.
11. Умножение многочленов “столбиком”.
12. Деление целых чисел многократной точности “столбиком”.
13. Алгоритмы быстрого умножения целых чисел многократной точности.