УДК 511.34

Студ. Н.В. Чистяков
Науч. рук. доц. Е.И. Ловенецкая
(кафедра высшей математики, БГТУ)

АДДИТИВНЫЕ ЦЕПОЧКИ

Рассмотрим следующую задачу: необходимо возвести число C в степень N наименьшим количеством действий.

Очевидно, что с помощью обычного последовательного возведения в степень потребуется $N-1$ действие. Если количество данных операций в алгоритме велико, то это уменьшит производительность программы, поэтому желательно найти решение получше.

При решении задачи будем считать простыми операциями возведение числа в квадрат, а также умножение текущего значения на уже известное.

Рассмотрим рекурсивный алгоритм возведения в степень. В нем используется возведение в квадрат для получения четной степени и умножение на исходное основание C – для нечетной, поэтому потребуется всего 2 новые переменные: результат возведения в степень и текущая степень. Обозначим через $F_R(N)$ количество простых операций, необходимых для возведения исходного числа в степень N. Легко видеть, что

$F_R(1) = 0$; $F_R(2k) = F_R(k) + 1$; $F_R(2k+1) = F_R(2k) + 1$.

Данный алгоритм не требует большого количества памяти и операций, однако он не дает наилучшего результата в данной задаче. Первым примером является $N = 15$. Рекурсивный метод дает решение за 6 действий:

$C^{15} = ((C^2*C)^2 *C)^2*C$,

однако возведение числа в 15-ю степень можно выполнить за 5 операций:

$C^{15} = ((C^2)^2*C)^3$

(возведение в 3-ю степень представляет собой умножение исходного выражения на квадрат самого себя).

Для поиска лучшего результата будем использовать понятие аддитивной цепочки[1]. Аддитивность – свойство величин, состоящее в том, что значение величины, соответствующее целому объекту, равно сумме значений величин, соответствующих его частям.

Аддитивной цепочкой для числа N называется последовательность натуральных чисел 1, 2, a_2, a_3, …, $a_l = N$, где каждый следующий член получается путем сложения каких-либо двух предыдущих:

$a_i = a_j + a_k, \quad 0 \le k \le j < i, \quad i = 1, 2, \ldots, l,$
при этом число l называется длиной аддитивной цепочки. Будем обозначать $F(N)$ минимальное из чисел l, т.е. наименьшую длину аддитивной цепочки для числа N.

Как видно из примера для $N = 15$, рекурсивный метод не всегда позволяет получить цепочку наименьшей длины, т. е. $F(N) \le F_R(N)$. Рассмотрим в качестве альтернативы рекурсивному методу бинарный (двоичный) метод решения поставленной задачи:

1) записать число N в двоичном представлении;

2) если стоит цифра 1, то вместо нее в итоговую строчку пишется SX, а если 0, то пишется S;

3) далее идет проход по получившейся строке: если X, то текущее значение умножается на исходное, а если S, то число возводится в квадрат.

Метод можно легко обосновать, проанализировав последовательность степеней при вычислении: если рассмотреть S как умножение на 2, а X как прибавление единицы, то получившаяся цепочка дает из единицы нужную степень N.

Несложно рассчитать количество операций бинарного метода $F_B(N)$, введя две вспомогательные функции: $\lambda(n) = [\log_2 N]$ – уменьшенная на единицу длина двоичной записи числа N, $\nu(n)$ – количество единиц в двоичном представлении N, которые могут быть вычислены рекурсивно:

$\lambda(1) = 0; \quad \lambda(2N) = \lambda(2N+1) = \lambda(N) + 1;$

$\nu(1) = 1; \quad \nu(2N) = \nu(N); \quad \nu(2N+1) = \nu(N) + 1.$

Тогда $F_B(N) = \lambda(N) + \nu(N) - 1$.

Бинарный метод требует память только для хранения C и текущего промежуточного результата, поэтому хорошо подходит для использования в программе. Однако можно убедиться в том, что $F_B(15) = F_R(15) > F(15)$, поэтому для уменьшения количества операций требуется рассмотреть другой метод.

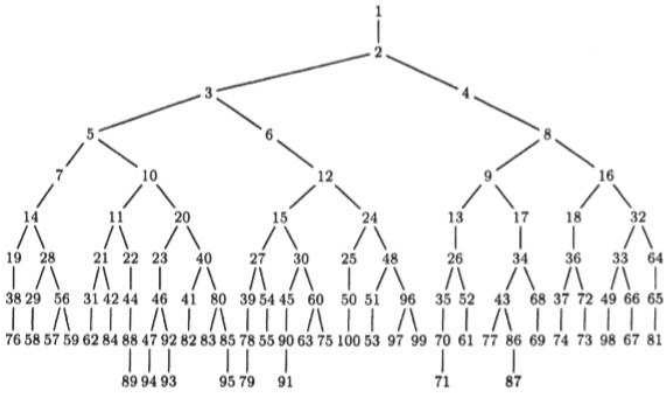
Метод множителей основан на разложении числа N на множители. Пусть $N = pq$. Тогда возведение числа C в степень N примет вид $C^N = (C^p)^q$. Например, если нужно возвести число в степень $N = 55$, то сначала вычисляем $y = C^5 = (C^2)^2 * C$ и $z = y^2$, а затем $C^{55} = y^{11} = (y^2)^5 * y = ((z^2)^2 * z) * y$.

Всего потребуется 8 действий против 9-ти у двоичного метода.

Метод множителей во многих случаях (но не всегда!) позволяет улучшить результат бинарного метода[2]. Таким образом, метод

множителей не гарантирует получение аддитивной цепочки наименьшей длины.

В [2] приводится следующее минимизирующее дерево для возведения числа в степень $N \leq 100$:



Это дерево позволяет для каждого значения N определить некоторую цепочку наименьшей длины и таким образом получить экономный способ вычисления C^N. Например, к числу $N = 15$ из вершины дерева ведет цепочка 1, 2, 3, 6, 12, 15. Поэтому для расчета C^{15} сначала вычисляем $y = C^3 = C^2 * C$, а затем

$C^{15} = C^{12} * C^3 = (y^2)^2 * y$.

Цепочка для $N = 55$ имеет вид 1, 2, 3, 6, 12, 15, 27, 54, 55. Следовательно,

$C^{55} = (C^{12} * C^3 * C^{12})^2 * C = (z * y * z)^2 * C$,

где $y = C^3 = C^2 * C$, $z = (y^2)^2$.

Приведенные примеры показывают, что аддитивная цепочка наименьшей длины для заданного N не единственна. При этом неясно, как гарантированно получить цепочку минимальной длины за разумное время.

Поиск наиболее экономичного способа вычисления C^N и расчета $F(N)$ представляет собой нетривиальную теоретическую и алгоритмическую задачу, возникающую при изучении оптимальных методов вычислений. Проблема определения $F(N)$ впервые была поставлена в 1894 году Х. Деллаком и частично решена Э. де Жонкиэресом методом множителей. В своей работе Э. де Жонкиэрес перечислил значения $F(N)$ для всех простых чисел $N < 200$, но значения, полученные для $N = 107, 149, 163, 179$, были завышены.

Как указано в [1], задача о точном вычислении $F(N)$ до сих пор в общем случае не решена. Легко видеть, что $F(2^n) = n$. Методом множителей можно доказать, что

$F(N) \leq F(p) + F(q)$, если $N = pq$;

бинарный алгоритм обеспечивает оценку

$F(N) \le F_B(N) = \lambda(N) + \nu(N) - 1$.

С другой стороны, можно доказать, что $F(N) \ge \lambda(N)$.

Неизвестно также, существует ли алгоритм полиномиальной сложности для расчета $F(N)$. Гарантировать точное значение может только полный перебор возможностей.

Целью данной работы является сравнение значений $F_R(N)$, $F_B(N)$ и $F(N)$ путем разработки программной реализации поиска необходимого для возведения числа в степень N количества операций тремя методами: рекурсивным, бинарным, полного перебора. Значения $F_R(N)$ и $F_B(N)$ вычисляются, как описано выше. Особенностью реализации алгоритма полного перебора является проверка всех путей к данной степени. Строится дерево, притом ветви друг о друге «не знают». Выписываются все возможные варианты перемножения текущей степени в вершине на известные. Если получается степень больше N, то дальше цепочка не строится. В итоге дерево остается как бы незаконченным в большинстве мест, но в одном или нескольких случаях получается нужная степень. В результате выбирается ветвь с минимальным количеством шагов.

Алгоритм полного перебора должен запоминать все возможные оптимальные способы получения каждой степени, а при проверке оптимальности новой цепочки – определять количество необходимых действий для возведения в степень N с учетом уже найденных аддитивных цепочек наименьшей длины для степеней n и m, где $N = n + m$. Например, для степени 5 есть 2 варианта получения: $N = 2 \cdot 2 + 1$ и $N = 2 + (2 + 1)$. В обоих случаях для получения степени требуется 3 операции. В первом случае в дальнейшем будут известны 1, 2, 4 степени, а во втором – 1, 2, 3 степени. Для возведения числа в 15-ю степень уже существует 4 оптимальных варианта, количество действий для каждого из вариантов равно 5. Так как при наращивании дерева используется умножение на уже известную степень, то необходимо проверять оба варианта получения базовой степени.

Реализация указанных алгоритмов для $N \le 100$ показала, что при всех этих значениях $F_R(N) = F_B(N)$. Для N = 15, 23, 27, 30, 31, …, 99 (всего 33 значения) $F_R(N)$ и $F_B(N)$ на единицу больше, чем $F(N)$; при $N = 63$ и $N = 95$ значения $F_R(N)$ и $F_B(N)$ превышают $F(N)$ на 2. Ожидаемо, что полный перебор даст лучшее значение операций, однако он требует больше памяти и времени. Однако если в программе часто используются операции возведения в различные степени, имеет смысл изначально создать минимизирующее дерево операций, и использование полного перебора для составления данного дерева будет являться оптимальным решением.

ЛИТЕРАТУРА

1. Гашков, С.Б. Задача об аддитивных цепочках и ее обобщения / С.Б. Гашков // Математическое просвещение, 2011, выпуск 15. С. 138-153.

2. Кнут, Д.Э. Искусство программирования / Д.Э. Кнут – 3-е изд. – М.: Вильямс, 2012. – 788 с.

УДК 519.171

Студ. М. Ю. Радченко, А. М. Карпач
Науч. рук. Е. В. Терешко
(кафедра высшей математики, БГТУ)

ЧТО ТАКОЕ ГРАФ? ПРИМЕНЕНИЕ ГРАФОВ В ПРОГРАММИРОВАНИИ

Если студент имеет цель стать хорошим специалистом в сфере своей специализации, ему необходимо разбираться во всех дисциплинах, преподаваемых в университете, в не зависимости, являются они предметами по специальности или нет. Именно поэтому проводить параллели и находить нечто общее между, на первый взгляд, ничем не схожими дисциплинами — одна из важнейших задач обучения. Тема «Графов» широко представлена в программировании и большинство программ, которые создаются студентами уже на 1 курсе, основаны, в том числе, и на математических графах.

Граф G — это упорядоченная пара G:=(V,E) , где V — это непустое множество вершин или узлов, а E — множество вершин, называемых рёбрами.

Вершины и рёбра графа называются также элементами графа, число вершин в графе — порядком, число рёбер — размером графа.

Граф может быть ориентированным или неориентированным. В ориентированном графе, связи являются направленными (то есть пары в E являются упорядоченными, например пары (1, 3) и (3, 1) это две разные связи). В свою очередь в неориентированном графе, связи ненаправленные, и поэтому если существует связь (1, 3) то значит что существует связь (3, 1).

Перед нами были поставлены вопросы: для чего нужны графы и что нужно знать, чтобы воспользоваться ими на практике при создании компьютерных программ?

Существует два способа представления графа: в виде списков смежности и в виде матрицы смежности. Оба способа подходят для представления ориентированных и неориентированных графов. И без труда представляются при помощи таких популярных языков программирования, как C++ или же JS.