Математика и её приложения: ЖИМО. 2008. Выпуск 1 (5). С. 97 – 104.

УДК 512.54

С. И. Хашин[1]

Реализация алгоритма кодирования и декодирования поля кратности при булевом сжатии файлов

Ключевые слова: булева алгебра, сжатие информации

Разработан и реализован на языке *Java* алгоритм, позволяющий для разложения заданного натурального числа в сумму натуральных с фиксированным числом слагаемых вычислить номер этого разложения в лексикографически упорядоченной таблице всех таких разложений. Разработан и реализован также обратный алгоритм.

We develop the algorithm that calculate for representation of any natural number as a sum of fixed quantity of natural addends, it's number in the table of all such representations, which is lexicographically ordered. We realize this algorithm in computer language *Java*. Also we develop and realize the return algorithm.

1. Введение

Рассмотрим алгоритм сжатия информации, подробно описанный в статьях [1, 2]. Согласно этому алгоритму, код сжимаемого файла содержит три поля, условно называемые "поле принадлежности", "поле кратности" и "поле порядка". Алгоритмы построения этих полей довольно сложны и допускают самые различные реализации [3, 4]. Как всегда, приходится выбирать между более простыми алгоритмами, но требующими много машинных ресурсов (памяти, времени), и более сложными в реализации, но и более экономно расходующими ресурсы. Важный показатель, от которого зависит приемлемый вариант алгоритма, — это численные характеристики рассматриваемых объектов.

Как и в других комбинаторных задачах, здесь приходится сталкиваться с так называемым "комбинаторным взрывом": в начале при росте параметра до некоторого критического значения объем требуемых ресурсов увеличивается, но не слишком значительно. А начиная с некоторого значения рост становится катастрофическим и для перехода к следующем значению параметра уже любых имеющихся ресурсов будет недостаточно.

Таким образом, для каждого рассматриваемого алгоритма имеется довольно четко обозначенная граница, для проникновения за которую тре-

[1] Ивановский государственный университет; E-mail: khash2@mail.ru. Работа выполнена при финансовой поддержке РФФИ (проект 07-07-00155).

© Хашин С. И., 2008

буются не дополнительные вычислительные ресурсы, а принципиальная переработка самого алгоритма.

В работе [2] предложен некоторый подход к построению поля порядка. Однако, он имеет тот недостаток, что требует последовательного прохождения всей лексикографически упорядоченной таблицы перестановок, что делает этот алгоритм весьма медленным, и критическое значение размера файла, за которым наступает "комбинаторный взрыв", составляет порядка 20 бит.

В работе [4] рассматривается другой подход к задаче о построении кода поля порядка, основанный на том, что вся таблица перестановок разбивается на отдельные блоки, соответствующие значениям первой, второй и т. д. цифрам в номере перестановки, и вычисления ведутся путем последовательного прохождения блоков, а не отдельных перестановок. Это позволяет рассматривать построение кода поля для более длинных буферов за приемлемое время.

В настоящей статье рассматриваются способы реализации этого алгоритма, возможные пределы его приложения, границы, за которыми наступает "комбинаторный взрыв".

2. Основные определения

Определение 1. *Пусть* $D(m,s)$ — *количество способов, которыми можно представить натуральное число* m *в виде суммы* s *натуральных слагаемых. При этом разложения* $m = m_1 + \cdots + m_s$, *различающиеся только порядком слагаемых, считаем за одно. Поэтому, в записи будем полагать* $m_1 \geq m_2 \geq \cdots \geq m_s$.

Пример. $D(6,3) = 3$, т. к. $6 = 4+1+1 = 3+2+1 = 2+2+2$.

Справедливы следующие **свойства функции** $D(m,s)$:

(1) $D(m,s) = 0$ при $s < 1$ и при $s > m$;

(2) $D(m,1) = D(m,m) = 1$.

Для удобства вычислений, введем еще одно понятие.

Определение 2. *Пусть* $D(m,s,k)$ — *количество способов, которыми можно представить натуральное число* m *в виде суммы* s *натуральных слагаемых, наибольшее из которых равно* k. *Представления, различающиеся только порядком слагаемых, считаем за одно.*

Пример. $D(6,3,6) = D(6,3,5) = 0$, $D(6,3,4) = 1$, $D(6,3,3) = 1$, $D(6,3,2) = 1$, $D(6,3,1) = 0$.

Свойства функции $D(m,s,k)$

(1) $D(m,s,k) = 0$ при $k > m - s + 1$.

(2) $D(m,s,k)=0$ при $ks<m$.

(3) Значение $D(m,s)$ может быть выражено через $D(m,s,k)$:

$$D(m,s)=\sum_{k=1}^{m-s} D(m,s,k).$$

На самом деле, из предыдущего свойства следует, что суммирование в формуле можно начинать не с 1, а с наименьшего целого числа $\geq m/s$.

(4) Рекуррентная формула для $D(m,s,k)$:

$$D(m,s,k)=\sum_{r=1}^{m-k-s+2} D(m-k,s-1,r).$$

Дополнительные свойства функции $D(m,s,k)$

Пусть $D(m,s,k)>0$ для некоторых натуральных (m,s,k). Тогда

(1) $s\leq m$ и $k\leq m$;

(2) если $s=m$, то $k=1$ и $D(m,m,1)=1$;

(3) если $s=1$, то $k=m$ и $D(m,1,m)=1$;

(4) $m/s\leq k\leq m-s+1$.

3. Обоснование выбора языка реализации

Для реализации этой идеи был выбран язык *Java*. Отметим основные черты языка, повлиявшие на этот выбор.

- *Открытость языка.* Распространение *Java*-программ вместе с исходными текстами является обычной практикой. Если даже их нет, *Java*-классы легко декомпилируются.

- *Ориентированность на Интернет.* Подобную *Java*-систему можно адаптировать для использования в качестве апплета, т. е. вызывать непосредственно из интернет-броузера. Можно представить себе ситуации, когда такая возможность окажется очень полезной.

- *Бесплатность.* И компилятор, и виртуальная *Java*-машины распространяются бесплатно [5].

- *Наличие класса BigInteger.* Среди методов этого класса, помимо стандартных арифметических операций, выделим следующие:

  преобразование из 10-ричного и 16-ричного представления и обратно,

  mod — взятие числа по модулю,

  modInvers — нахождение обратного по модулю ($1/x \mod p$),

  modPow — возведение в степень по модулю ($x^y \mod p$).

  Эффективная и удобная реализация массивов, включая операции сортировки, поиска и т. д. Важно так же то, что многомерные массивы не обязаны быть прямоугольными, каждый подмассив может иметь свою размерность.

Реализация этих операций в языке тщательно оптимизирована. Так как все стандартные библиотеки языка *Java* поставляются вместе с исходными текстами, их реализацию можно подробно изучить (см., напр., *src.zip\java\math\BigInteger.java*).

4. Вычисление функции $D(m, s, k)$ для малых m

Описанный выше алгоритм является рекурсивным, довольно легко реализуемым. С помощью такой простейшей реализации можно найти все значения функции при малых m. В таблице 1 приведены значения $D(m, s, k)$ при $m = 3, \ldots, 6$: в i-й строке приведены значения $D(3, i, k)$.

Таблица 1. Значения функции $D(m, s, k)$ при $m = 3, \ldots, 6$

D(3,s,k)			D(4,s,k)				D(5,s,k)					D(6,s,k)					
1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
2	0	1	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1	0
3	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1	1	0	0
				1	0	0	0	0	1	0	0	0	0	1	1	0	0	0
								1	0	0	0	0	0	1	0	0	0	0
													1	0	0	0	0	0

D(7,s,k)							D(8,s,k)								D(9,s,k)								
1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
2	0	0	0	1	1	1	0	2	0	0	0	1	1	1	1	0	2	0	0	0	1	1	1	1	0
3	0	0	2	1	1	0	0	3	0	0	1	2	1	1	0	0	3	0	0	1	2	2	1	1	0	0
4	0	1	1	1	0	0	0	4	0	1	2	1	1	0	0	0	4	0	0	2	2	1	1	0	0	0
5	0	1	1	0	0	0	0	5	0	1	1	1	0	0	0	0	5	0	1	2	1	1	0	0	0	0
6	0	1	0	0	0	0	0	6	0	1	1	0	0	0	0	0	6	0	1	1	1	0	0	0	0	0
7	1	0	0	0	0	0	0	7	0	1	0	0	0	0	0	0	7	0	1	1	0	0	0	0	0	0
							8	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
															9	1	0	0	0	0	0	0	0	0

Однако, такое прямое вычисление функции $D(m, s, k)$ не очень эффективно из-за очень большого количества рекурсивных вызовов функции. В таблице 2 показана зависимость объема вычислений (количества вызовов функции $D(m, s, k)$ в процессе рекурсии) для некоторых значений (m, s, k).

Таблица 2. Число вызовов функции $D(m, s, k)$ в процессе рекурсии

m	s	k	объем вычислений
20	7	8	32
40	13	16	312
60	19	24	1 929
80	25	32	9 233
100	31	40	37 190
120	37	48	132 166
140	43	56	426 498
160	49	64	1 274 088
180	55	72	3 571 638
200	61	80	9 490 192
220	67	88	24 084 496
240	73	96	58 728 769
260	79	104	138 258 240
280	85	112	315 464 072
300	91	120	699 886 737

Выполнение 700 млн. вычислений функции уже требует очень заметного времени. Фактически, вычислить таким путем значение функции при $m > 300$ практически нереально.

5. Оптимизация вычислений

Наиболее прямой метод оптимизации вычислений в данной ситуации — предварительное вычисление и запоминание значений функции $D(m, s, k)$ при малых значениях m, s, k. Ситуация осложняется тем, что массив данных должен быть трехмерным, а значит он будет занимать весьма большой объем памяти. Например, для запоминания всех значений функции при $0 < m, s, k < 200$ требуется $200^3 = 8\,000\,000$ чисел, что уже недалеко от верхнего предела возможностей компьютера. Для уменьшения требуемого объема памяти можно использовать не прямоугольные массивы, а именно запоминать только значения $D(m, s, k)$ при

- $7 \leq m \leq 200$,
- $3 \leq s \leq m - 4$,
- $3 \leq k \leq m - s + 1$.

Благодаря такой организации массива количество требуемой памяти можно существенно сократить. Например, если предвычислять значения функции при $m \leq 200$, то на это потребуется около 10 Мбайт памяти; если до $m \leq 300$, то около 35 Мбайт памяти и несколько секунд времени. После такой предварительной подготовки объем вычислений существенно сократится (см. Табл. 3). При этом последнее значение $D(500, 151, 102)$ уже оказывается равным примерно $2 \cdot 10^{14}$, и на его вычисление тратится лишь несколько секунд.

Таблица 3. Число вызовов функции $D(m, s, k)$ после оптимизации

m	s	k	объем вычислений
200	61	42	1
220	67	46	1
240	73	50	1
260	79	54	1
280	85	58	1
300	91	62	63
320	97	66	67
340	103	70	71
360	109	74	75
380	115	78	3 157
400	121	82	5 600
420	127	86	42 857
440	133	90	433 126
460	139	94	3 735 111
480	145	98	27 296 149
500	151	102	173 461 999

6. Время вычисления индекса

Рассмотрим теперь, как сказывается введение предвычисления на времени выполнения основной функции — нахождения номера “типа повторов” [4].

Пусть $n_1, n_2, \ldots, n_s$ — тип повторов, т. е. последовательность натуральных чисел, показывающих, сколько раз каждый из s различных кортежей, входящих в буфер из m кортежей, встречается в этом буфере. При этом выполнено соотношение

$$\sum_{k=1}^{s} n_k = m. \tag{1}$$

Упорядочим лексикографически все типы повторов считая, что

$$n_k \leq n_{k+1} \quad (k = 1, 2, \ldots, s-1).$$

В результате получается таблица

$$\begin{array}{lll} 1. & n_k = 1, & n_s = n - s + 1; \\ & & k = 1, 2, \ldots, s-1, \\ 2. & n_k = 1, & n_{s-1} = 2,\ n_s = m - s, \\ & & k = 1, 2, \ldots, s-2, \\ & \ldots & \\ D^s_{m-s}. & n_k = \left[\frac{m}{s}\right], & n_{p+l} = \left[\frac{m}{s}\right] + 1, \\ & & k = 1, 2, \ldots, p, \\ & & p = \left(\left[\frac{m}{s}\right] + 1\right) s - m, \\ & & l = 1, 2, \ldots, s-p, \end{array} \qquad (2)$$

где через D^s_{m-s} обозначено число всех типов повторов при заданных m и s. Задача заключается в том, чтобы по заданному типу повторов $n_1, n_2, \ldots, n_s$ вычислить номер этого типа в таблице (2) — это задача кодирования поля кратности. Вторая, решаемая в этой работе задача, обратная к первой, — по заданному номеру в таблице (2) вычислить тип повторов $n_1, n_2, \ldots, n_s$ — это задача декодирования поля кратности.

Для вычисления номера типа повторов используется функция $D(m, s, k)$. В табл. 4 показано время вычисления индекса для данного типа повторов в двух случаях: прямого вычисления и с использованием предвычисления некоторых значений функции $D(m, s, k)$.

Таблица 4. Время вычисления индекса для данного типа повторов

$(n_1, \ldots n_k)$	время вычислений, мсек.	с предвычислением
(2,7,8,11,14,15)	< 1	< 1
(3,10,11,15,19,21)	< 1	< 1
(5,14,15,21,26,29)	15	< 1
(7,19,21,29,35,39)	16	< 1
(10,26,29,39,47,53)	63	15
(14,35,39,53,63,71)	234	22
(19,47,53,71,85,95)	891	25
(26,63,71,95,114,127)	3515	101
(35,85,95,127,153,170)	13657	687
(47,114,127,170,205,227)	55312	19328

7. Выводы

- Язык *Java* адекватен рассматриваемой ситуации и вполне может быть использован в дальнейшей реализации.

- Рекурсивные алгоритмы для вычисления рассматриваемых функций дают в целом вполне приемлемую скорость вычислений.

- Оптимизация с помощью предвычисления, а также некоторые другие методы показывают достаточно успешные для наших целей результаты.

Список использованной литературы

1. *Толстопятов А. А.* О структуре дискретной информации и общих условиях ее сжатия // Вестник ИвГУ. – 2002. – Вып. 3. – С. 80–82.
2. *Толстопятов А. А., Хашин С. И.* Алгоритм построения поля порядка при булевом сжатии // Вестник ИвГУ. – 2004. – Вып. 3. – С. 139–143.
3. *Толстопятов А. А.* Вычисление длины поля кратности при булевом сжатии файлов // Вестник ИвГУ. – 2004. – Вып. 3. – С. 71–76.
4. *Толстопятов А. А.* Быстрый алгоритм кодирования и декодирования поля порядка при булевом сжатии файлов // Математика и ее приложения: Журн. Иванов. матем. об-ва. – 2007. – Вып. 1 (4). – С. 35–46.
5. Java(TM) Platform Standard Ed. 6. // `http://java.sun.com`.

Поступила в редакцию 12.12.2008.