УДК 004.021

ПРИМЕНЕНИЕ ДЕРЕВЬЕВ ДЛЯ РЕАЛИЗАЦИИ МАССОВЫХ ОПЕРАЦИЙ НА МНОГОМЕРНЫХ МАССИВАХ ДАННЫХ

А.Г. Банных

Предлагается метод построения структур данных для выполнения массовых операций на многомерных структурах данных для узкого класса задач. Предлагаемый метод применим в том случае, если элементы многомерной структуры данных принадлежат абелевой группе, и позволяет эффективно выполнять вычисление суммы и прибавление значения к многомерной области.

Ключевые слова: структуры данных, массовые операции, многомерные массивы данных, дерево отрезков, дерево Фенвика.

Введение

Для эффективной работы с информацией были разработаны разнообразные структуры данных. Одна из самых распространенных из них – дерево. Существует огромное множество различных типов деревьев. Эти структуры позволяют собирать статистику на отрезках и изменять отдельные элементы. Представление об этих структурах данных можно получить в классических трудах [1–4].

Практически все эти деревья предназначены для данных, на которых можно ввести линейный порядок. В данной работе рассматриваются многомерные массивы данных. Введение нескольких измерений может быть как естественным (растровые изображения и видео), так и искусственным (базы данных). В последнем случае дополнительные измерения позволяют точнее указывать интересующую выборку информации. Широко известны квадродеревья [5], кд-деревья [6], обобщения дерева отрезков [7] и дерева Фенвика [8]. Существующие структуры данных для работы с многомерными структурами данных позволяют эффективно получать статистическую информацию и изменять отдельные элементы.

Цель настоящей работы заключается в том, чтобы исследовать возможность выполнения массовых обновлений – единообразного изменения целых областей данных. Ни одна вышеперечисленная структура данных в чистом виде не предоставляет подобную функциональность. Приведем простой пример. Пусть дан двумерный массив целых чисел размера $n \times n$. Требуется эффективно выполнять две операции – вычислять сумму на прямоугольнике и прибавлять число к прямоугольнику. Оказывается, что добиться асимптотики $\mathrm{O}(\log^2 n)$ для выполнения этих операций – вовсе не тривиальная задача, как может показаться на первый взгляд. Подробнее о возникающих проблемах можно узнать в работе [9].

Постановка задачи

Рассмотрим d-мерное векторное пространство $U_d = \mathbb{Z}^d$. Элементы этого пространства будем обозначать жирным шрифтом. Компоненты векторов будем обозначать нижними индексами. Например, $\mathbf{x} = (x_1, x_2, \ldots, x_d)$. Введем частичный порядок на элементах этого пространства. Скажем, что $\mathbf{x} \le \mathbf{y} \Leftrightarrow \forall k \in \{1, 2..d\}\, x_k \le y_k$. Кроме того, будем считать, что $\mathbf{1}_d = (1,1,\ldots,1) \in U_d$.

Для любых $x, y \in U_d\ x \le y$ областью будем называть множество $P_{x,y} = [x_1..y_1] \times [x_2..y_2] \times \ldots \times [x_d..y_d]$. Для повышения читаемости формул будем опускать нижние индексы в тех случаях, когда это не приводит к неоднозначностям. Будем считать, что мы работаем с областью $W = P_{\mathbf{1},\mathbf{N}}$, где $\mathbf{N} \in U_d, \mathbf{1} \le \mathbf{N}$. Эта область – не что иное, как множество индексов некоторого d-мерного массива.

Сопоставим индексам значения. Рассмотрим множество G и два бинарных оператора: $+$ и $\circ$ $(+, \circ : G \times G \to G)$. Далее оператор $+$ будем называть сложением, а $\circ$ – умножением. Сложение используется для выполнения массовых запросов, таких как суммирование на прямоугольнике. Умножение используется при выполнении массовых обновлений, таких как умножение всех элементов подпрямоугольника на число.

Рассмотрим функцию $A : W \to G$. Можно рассматривать A как многомерный массив с множеством индексов W и множеством допустимых значений ячеек G.

Массовый запрос определим следующим образом:

$$get(A, P) = \sum_{x \in P} A(x).$$

Массовое обновление определим следующим образом:

$$update(A, P, v) = A' : W \to G,$$

где

$$A'(x) = \begin{cases} A(x) \circ v & x \in P \\ A(x) & x \notin P \end{cases}.$$

Задача состоит в том, чтобы реализовать структуру данных, которая позволила бы эффективно выполнять эти массовые операции.

Чтобы массовые операции имели смысл, на них нужно наложить некоторые ограничения. Обычно полагают, что $< G, + >$ – коммутативная полугруппа, а $< G, \circ >$ – полугруппа. Кроме того, потребуем выполнения дистрибутивного закона умножения относительно сложения.

Подобная модель описывает не все возможные массовые запросы. Тем не менее, она проста, многие естественные запросы легко описываются в терминах этой модели, и на операции наложены требования, которые учитывают особенности реализации массовых операций в одномерном случае. Это позволяет надеяться на то, что задача имеет эффективное решение.

Настоящая работа посвящена узкому случаю: $+ \equiv \circ$, $< G, + >$ – абелева группа. Стоит подчеркнуть, что у всех элементов группы существует обратный, т.е. $\forall a \in G \exists a^{-1} \in G : a + a^{-1} = 0$. Это свойство лежит в основе всех дальнейших рассуждений.

Основные утверждения

Чтобы упростить дальнейшее изложение, введем несколько операторов. Под ними скрываются преобразования массивов, которые напоминают дифференцирование и интегрирование функций. Для случая одномерной последовательности подобные преобразования хорошо описаны в работе [10].

Префиксом будем называть область вида $P_{\mathbf{1},\mathbf{R}}, \mathbf{R} \in W$. Суффиксом будем называть область вида $P_{\mathbf{L},\mathbf{N}}, \mathbf{L} \in W$. Обозначим замену i-й компоненты вектора следующим образом: $\mathbf{x}_{x_i \leftarrow x_i'}$.

Определение 1. *Дифференциальный оператор по i-му индексу*

$D_i A(\mathbf{x}) = A(\mathbf{x}) - A(\mathbf{x}_{x_i \leftarrow x_i - 1})$.

Определение 2. *Интегральный оператор по i-му индексу*

$$S_i A(\mathbf{x}) = \sum_{j=1}^{x_i} A(\mathbf{x}_{x_i \leftarrow j}).$$

Приведем ключевые свойства этих операторов. Для экономии места доказательства тривиальных фактов опущены. Ознакомиться с ними можно в работе [10].

Утверждение 1. *Операторы D_a и D_b коммутируют.*

Утверждение 2. *Операторы S_a и S_b коммутируют.*

Утверждение 3. *Операторы S_i и D_i взаимообратны.* $S_i D_i = D_i S_i = E$, где E – тождественный оператор.

Утверждение 4. *Операторы S_a и D_b коммутируют.*

Определение 3. *Дифференциальный оператор* $D = D_1 D_2 \cdots D_d$.

Определение 4. *Интегральный оператор* $S = S_1 S_2 \cdots S_d$.

Введение этих операторов позволяет вплотную подобраться к основному результату.

Лемма 1. $SD = DS = E$.

Лемма 2. $SA(\mathbf{x}) = get\left(A, P_{\mathbf{1},\mathbf{x}}\right)$.

Доказательство.

$$SA(\mathbf{x}) = S_1 S_2 \cdots S_d A(\mathbf{x}) = \sum_{j_1=1}^{x_1} \sum_{j_2=1}^{x_2} \cdots \sum_{j_d=1}^{x_d} A(\mathbf{x}_{x_1 \leftarrow j_1 x_2 \leftarrow j_2 \ldots x_d \leftarrow j_d}) = \sum_{\mathbf{1} \le \mathbf{j} \le \mathbf{x}} A(\mathbf{j}) = get\left(A, P_{\mathbf{1},\mathbf{x}}\right).$$

Таким образом, запись $SA(\mathbf{x})$ можно воспринимать как обычный запрос на сумму на префиксе.

Обратимся теперь к оператору D. Это вполне определенное преобразование, которое можно совершить с любым массивом. В работе массив $DA = B$ будет храниться в явном виде. Сформулируем некоторые его свойства.

Лемма 3. *Сумма на префиксе в преобразованном оператором D массиве данных A есть элемент исходного массива.*

Доказательство. Используя лемму 1, получаем:

$B = DA \Rightarrow SB = SDA = EA = A \Rightarrow \forall \mathbf{x}\, SB(\mathbf{x}) = A(\mathbf{x})$.

Полученный результат показывает связь между преобразованным массивом (B) и тем, что был изначально (A). По сути, эта лемма доказывает эквивалентность этих двух массивов, что позволяет хранить лишь один из них. Осталось понять, чем же новый массив лучше старого.

Лемма 4. *Изменение одного элемента B влечет изменения целого суффикса массива* $A = SB$.

Доказательство. Действительно, выберем произвольные $\mathbf{x} \in W$ и $v \in G$ и предположим, что $B'(\mathbf{y}) = \begin{cases} B(\mathbf{y}) + v & \mathbf{y} = \mathbf{x} \\ B(\mathbf{y}) & \mathbf{y} \neq \mathbf{x} \end{cases}$.

Положим $A = SB$ и $A' = SB'$. Тогда $A'(\mathbf{y}) = \begin{cases} A(\mathbf{y}) + v & \mathbf{x} \le \mathbf{y} \\ A(\mathbf{y}) & \mathbf{x} \not\le \mathbf{y} \end{cases}$.

Преобразование данных и запросов

Построим массив $B = DA$. Один из самых простых способов сделать это – используя определение 3, по очереди применить дифференциальные операторы к исходному массиву.

Преобразование запроса на сумму. Действие оператора S тесно связано с запросом на сумму на префиксе. Например, по леммам 2 и 3 $get\left(A, P_{\mathbf{1},\mathbf{x}}\right) = SA(\mathbf{x}) = SSB(\mathbf{x})$.

Последнее можно записать и иначе:

$$get\left(A, P_{\mathbf{1},\mathbf{x}}\right) = SSB(\mathbf{x}) = \sum_{\mathbf{1} \le \mathbf{y} \le \mathbf{x}} \sum_{\mathbf{1} \le \mathbf{z} \le \mathbf{y}} B(\mathbf{z}).$$

Значение этой записи интуитивно понятно: сумма превратилась в «сумму сумм».

Преобразование запроса на прибавление. Как было показано в лемме 4, изменение одной ячейки в B эквивалентно массовому прибавлению к суффиксу массива A. Таким образом, массовое обновление сводится к изменению единственной ячейки массива B.

Эквивалентность. На данный момент будет не совсем корректно говорить о решении задачи. Действительно, было показано, чему соответствуют запросы на префиксах или суффиксах, однако исходные запросы формулируются в терминах произвольных областей. В дальнейшем запросы на префиксе для суммы и на суффиксе для обновления будем называть ослабленными. Покажем, что любой запрос можно выразить не более чем через 2^d ослабленных.

Чтобы решить эту проблему, нужно воспользоваться формулой включения-исключения. Предположим, необходимо ответить на запрос $get\left(A, P_{\mathbf{x},\mathbf{y}}\right)$. Предположим, что $x_1 \neq 1$. Тогда выразим исходный запрос следующим образом:

$$get\left(A, P_{\mathbf{x},\mathbf{y}}\right) = get\left(A, P_{\mathbf{x}_{x_1 \leftarrow -1},\mathbf{y}}\right) - get\left(A, P_{\mathbf{x}_{x_1 \leftarrow -1},\mathbf{y}_{y_1 \leftarrow x_1 - 1}}\right).$$

Повторяя подобное преобразование для остальных индексов, выразим исходный запрос только через запросы на префиксах. Для этого потребуется не более d шагов, потому итоговое число запросов не превзойдет 2^d. Аналогично можно выполнить массовые обновления. Здесь снова используется существование обратных элементов, что позволяет одним массовым обновлениям компенсировать другие.

Эффективность. Поскольку большинство алгоритмов работы с многомерными массивами данных содержат $O(\log^d N)$ в своей асимптотике, то умножение на 2^d не изменит асимптотическую оценку.

Решение задачи

Сосредоточимся на выполнении запроса на «сумму сумм». В дальнейшем будет удобно воспользоваться нотацией, предложенной в работе [11]:

$$[\text{утверждение}] = \begin{cases} 1, & \text{если утверждение истинно;} \\ 0, & \text{в противном случае.} \end{cases}$$

Лемма 5. *Для любого* $\mathbf{x} \le \mathbf{z}$ *значение* $B(\mathbf{x})$ *входит в* $SSB(\mathbf{z})$ *ровно* $\prod_{i=1}^{d}(z_i - x_i)$ *раз.*

Доказательство. Действительно $SSB(\mathbf{z}) = \sum_{\mathbf{1} \le \mathbf{y} \le \mathbf{z}} \sum_{\mathbf{1} \le \mathbf{y}' \le \mathbf{y}} B(\mathbf{y}')$. Количество вхождений значения $B(\mathbf{x})$ в эту сумму равно $\sum_{\mathbf{1} \le \mathbf{y} \le \mathbf{z}} \sum_{\mathbf{1} \le \mathbf{y}' \le \mathbf{y}} [\mathbf{y}' = \mathbf{x}]$.

$$\sum_{\mathbf{1} \le \mathbf{y}' \le \mathbf{y}} [\mathbf{y}' = \mathbf{x}] = \begin{cases} 1 & \mathbf{x} \le \mathbf{y} \\ 0 & \mathbf{x} \not\le \mathbf{y} \end{cases} = [\mathbf{x} \le \mathbf{y}] \Rightarrow \sum_{\mathbf{1} \le \mathbf{y} \le \mathbf{z}} \sum_{\mathbf{1} \le \mathbf{y}' \le \mathbf{y}} [\mathbf{y}' = \mathbf{x}] = \sum_{\mathbf{1} \le \mathbf{y} \le \mathbf{z}} [\mathbf{x} \le \mathbf{y}] = \sum_{\mathbf{x} \le \mathbf{y} \le \mathbf{z}} 1 = \prod_{i=1}^{d}(z_i - x_i).$$

Будем в каждой ячейке хранить не просто значение $B_{\mathbf{x}}$, а многочлен $B_{\mathbf{x}} \prod_{i=1}^{d}(z_i - x_i)$, где z_i заранее не известны. Отметим, что после раскрытия всех скобок получится 2^d слагаемых.

Необходимо пояснить, что подразумевается под умножением элемента абелевой группы $< G, + >$ на число. Можно было бы решить, что мы имеем дело с кольцом, но это не так. Подчеркнем, что все

коэффициенты целые, поэтому умножение $g \in G$ на число $a \geq 0$ есть всего лишь $\sum_{i=1}^{a} g$. Аналогично определяется эта операция для случая $a < 0$. В итоге приходим к тому, что в каждой ячейке хранится 2^d элементов G. Поскольку это коэффициенты многочлена, то их можно складывать как векторы. Массив, содержащий эти многочлены, обозначим как C.

Рассмотрим $\sum_{\mathbf{1} \leq \mathbf{x} \leq \mathbf{z}} C_{\mathbf{x}}$. Получим многочлен, представляющий собой число вхождений соответствующих элементов в запрос на сумму $sum(A, P_{1,z})$ (по леммам 2, 3, 5). Это означает, что

$$sum(A, P_{1,\mathbf{z}}) = \left[\sum_{\mathbf{1} \leq \mathbf{x} \leq \mathbf{z}} C_{\mathbf{x}} \right](\mathbf{z}).$$

Чтобы ответить на запрос на сумму на префиксе, нужно вычислить сумму на префиксе в массиве C, после чего в получившийся многочлен подставить параметры запроса. При выполнении ослабленного запроса на массовое обновление изменяется одна ячейка, т.е. прибавление на суффиксе было сведено к прибавлению нового многочлена к одной ячейке. Таким образом, для решения финальной задачи достаточно поддержки двух запросов: сумма на префиксе и изменение одного элемента. Для решения этой задачи существует множество структур данных, позволяющих выполнять указанные запросы за $O(\log^d N)$.

Оценка потребляемых ресурсов

Время работы. Характерное время обработки запросов популярными структурами данных составляет $O(\log^d N)$. Сведение исходных запросов к ослабленным приводит к не более чем 2^d запросам к структуре. Таким образом, асимптотическая оценка времени обработки одного запроса составляет $O(2^d \log^d N) = O(\log^d N)$ при фиксированной размерности пространства d.

Для завершения оценки времени работы осталось определить, какое время занимают элементарные операции над многочленами. Будем считать операцию $+$ в G элементарной, т.е. на ее выполнение уходит $O(1)$ времени. Тогда после преобразования каждый элемент заменяется на многочлен степени 2^d. Значит, сложение двух многочленов выполняется за $O(2^d)$. Таким образом, асимптотической оценкой времени выполнения запроса осталось $O(\log^d N)$.

Обратимся теперь к массовым обновлениям. По аналогии с предыдущими рассуждениями можно считать, что ослабление запросов не ведет к ухудшению асимптотики. Единственное, с чем осталось разобраться, – построение нового многочлена.

Ключевой вопрос – за какое время можно выполнять умножение на целое число. Если мы имеем дело с примитивными типами данных, то эту операцию можно выполнить за $O(1)$ встроенными командами процессора. В иных случаях можно представить умножение на число через $O(\log(L))$ операций $+$, где L – абсолютное значение множителя [12]. Коэффициенты многочлена не превышают N^d, поэтому общая оценка времени, затраченного на построение нового многочлена, есть $O(2^d \log(N^d)) = O(\log^d N)$.

Память. Потребление памяти зависит от используемой структуры данных, поэтому сложно привести какие-либо конкретные оценки. Тем не менее, можно уверенно утверждать, что после всех преобразований потребление памяти возрастет не менее чем в 2^d раз, что связано с хранением многочленов. Кроме того, в случае разреженных данных количество ненулевых ячеек может также возрасти, но не более чем в 2^d раз.

Для большинства структур данных применение разработанного метода не ухудшает асимптотическую оценку времени их работы. Тем не менее, при возрастании размерности массива данных константа, скрытая в асимптотической оценке, возрастает экспоненциально.

Заключение

В работе был выбран частный случай массовых операций: $+ \equiv \circ$, $< G, + >$ – абелева группа. Для него был разработан общий метод сведения задачи с массовыми обновлениями к хорошо изученной задаче без них. Разработанный метод можно использовать с любыми существующими структурами данных. Это свойство позволяет делать выбор оптимальной структуры данных для каждой задачи отдельно. Например, если используются все ячейки массива, то стоит сделать выбор в пользу дерева Фенвика. С другой стороны, в случае сильно разреженных данных можно использовать разреженное дерево отрезков. Подобная гибкость позволяет применять метод в различных областях.

Таким образом, метод получился универсальным настолько, насколько это возможно для данной задачи. К сожалению, вопрос об обобщении метода на более широкие классы задач остается открытым.

Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. – 2-е изд. – М.: МЦНМО, 2010. – 1296 с.
2. Шень А. Программирование: теоремы и задачи. – 2-е изд. – М.: МЦНМО, 2004. – 296 с.
3. Tarjan R.E. Data Structures and Network Algorithms. – PA.: Society for Industrial Mathematics, 1983. – 140 p.
4. Кнут Д.Э. Искусство программирования. Т. 3. Сортировка и поиск. – М.: Вильямс, 2007. – 824 с.
5. Романовский И.В. Дискретный анализ. – 4-е изд. – СПб: Невский Диалект; БХВ-Петербург, 2008. – 336 с.
6. Bentley J.L. Multidimensional binary search trees used for associative searching // Communications of the ACM. – 1975. – V. 18. – № 9. – P. 509–517.
7. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. – М.: Мир, 1989. – 478 с.
8. Fenwick P.M. A New Data Structure for Cumulative Frequency Tables // Software: Practice and experience. – 1994. – V. 24 – № 3. – P. 327–336.
9. Кнут Д., Грэхем Р., Паташник О. Конкретная математика. Основание информатики. – М.: Вильямс, 2005. – 784 с.
10. Банных А.Г. Применение деревьев для реализации массовых операций на многомерных массивах данных [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/papers/2011-bachelor-bannykh/, св. Яз. рус. (дата обращения 25.11.2011).
11. Кнут Д.Э. Искусство программирования. Т. 1. Основные алгоритмы. – М.: Вильямс, 2007. – 720 с.
12. Brauer A. On addition chains // Bulletin of the American Mathematical Society. – 1939. – V. 45. – № 10. – P. 736–739.

Банных Антон Геннадьевич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, студент, anton.bannykh@gmail.com