Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра компьютерных технологий

А.Г. Банных

Применение деревьев для реализации массовых операций на многомерных массивах данных

Дипломная работа

Научный руководитель: А.С. Станкевич

Санкт-Петербург
2011

Оглавление

Введение

Человечество производит огромное количество информации. Эту информацию необходимо не только хранить, но и обрабатывать, изменять, собирать статистику «на лету». Подтверждением этих слов может служить любая база данных, поисковая система, социальная сеть.

Для эффективной работы с информацией были разработаны разнообразные структуры данных. Поскольку значение понятия «эффективность» зависит от специфики решаемой задачи, то и предназначение, и возможности этих структур значительно разнятся. Некоторые из них предназначены для эффективного хранения данных, другие же позволяют собирать статистическую информацию или изменять данные с минимальными затратами ресурсов. Нас будут интересовать последние.

Самая распространенная структура данных — дерево. Существует большое число различных деревьев. Эти структуры позволяют собирать статистику на отрезках и изменять отдельные элементы. Представление об этих структурах данных можно получить в классических трудах [1, 2, 3, 4].

Практически все эти деревья предназначены для данных, на которых можно ввести линейный порядок. Дело в том, что не так уж часто встречаются задачи, которые нельзя свести к одномерному случаю. Там же, где размерности действительно велики, данные обычно сильно разрежены. Многомерные деревья нашли свое применение в различных областях: файловых системах, геометрических алгоритмах, поисковых системах.

Обратимся к многомерным массивам данных. В первую очередь, это изображения и видео, а также базы 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.

Специфика задач обработки многомерных массивов данных может

сильно разниться. Размерность может быть как небольшой (изображения), так и значительной (базы данных). Кроме того, в последнем случае данные скорее всего будут разреженными. Все это необходимо учитывать разработке или выборе существующей структуры данных для решения конкретной задачи.

Широко известны квадродеревья, кд-деревья, обобщения дерева отрезков и дерева Фенвика. Существующий структуры данных для работы с многомерными структурами данных позволяют эффективно получать статистическую информацию и изменять отдельные элементы.

Цель данной работы 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. Такая возможность востребована в базах данных и задачах автоматической обработки медиаинформации.

Вопрос массовых обновлений хорошо изучен для одномерного случая. При этом отметим, что при попытке обобщения используемых идей на случай многомерных массивов данных, возникают проблемы, обсуждения которых проводится в главе 1.

Работа с данными произвольной размерности трудно поддается описанию. Простейшие преобразования и доказательства становятся трудновоспринимаемыми. Для того, чтобы избежать этих проблем, в главе 2 вводятся специальные операторы, в терминах которых доказываются основные утверждения.

В главе 3 обсуждается использование полученных результатов на практике. В результате исходная задача приводится к виду, в котором отсутствуют массовые обновления. Обратной стороной этого преобразования является усложнение массового запроса.

Глава 4 посвящена решению новой задачи. В ходе обсуждения проблемы производится замена элементов массива на многочлены и переформулирование запросов таким образом, чтобы задача решалась стандартными методами. Итогом главы становится метод преобразования задачи с массовыми запросами и обновлениями к задаче с массовыми запросами и единичными обновлениями. Эта задача хорошо изучена и может быть решена с использованием различных структур данных.

В главе 5 обсуждаются вопросы производительности и использование ресурсов. Кроме того, производится обсуждение выбора предпочтительной структуры данных в зависимости от специфики задачи. В этой главе вскрывается основное преимущество разработанного метода — сохранение положительных сторон структур данных. Это дает возможность эффективно использовать метод в различных областях.

В приложении А приведен пример кода на языке программирования Java. Поскольку результат работы — общий метод, то код приведен лишь для демонстрации эффективности использования этого метода.

Глава 1. Массовые операции

1.1. Постановка задачи

Прежде чем решать задачу, необходимо четко сформулировать, что именно мы решаем. Действительно, под туманными терминами «многомерные массивы данных» и «массовая операция» может скрываться что угодно. Определение этих понятий задает модель задачи.

В выборе модели скрыта определенная проблема. Если взять слишком общую модель, то задача станет слишком сложной и, возможно, не будет иметь решения. Если же модель будет слишком узкой, то решение задачи не будет иметь практической пользы, так не будет применимо нигде, кроме это модели.

Ниже мы рассмотрим довольно общую постановку задачи. Эта формулировка с одной стороны позволяет покрыть довольно много естественные случаев. С другой стороны, накладываемые на нее требования позволяют надеяться на существование эффективного решения.

1.1.1. Основные определения

Для начала, нужно определить объект, с которым мы работаем. В этой работе будет удобнее использовать математические понятия, поэтому многомерный массив будет определен, как функция со специфической областью определения.

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 $\mathbf{x} \leq \mathbf{y} \Leftrightarrow \forall k \in [1..d]\ x_k \leq y_k$. Кроме того, будем считать, что $\mathbf{1}_d =$

$(1, 1, \ldots, 1) \in U_d$.

Настало время определить многомерную область. По сути ее можно рассматривать, как множество индексов d-мерного массива. Для любых $x, y \in U_d\, x \leq y$ *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 $\mathbf{N} \in U_d, \mathbf{1} \leq \mathbf{N}$. Эта область — не что иное, как множество индексов некоторого d-мерного массива. Осталось сопоставить индексам значения.

Рассмотрим множество G и два бинарных оператора: $+$ и $\circ$ ($+, \circ : G \times G \to G$). Оператор $+$ используется для выполнения массовых запросов, например суммирование на прямоугольнике. Оператор $\circ$ используется при выполнении массовых обновлений. В качестве примера можно привести умножение всех элементов подпрямоугольника на число.

Рассмотрим функцию $A : W \to G$. Получается, что A — многомерный массив, а G — множество допустимых значений ячеек массива.

Осталось определить массовые операции.

Массовый запрос определим следующим образом: $\text{get}\,(A, P) = \sum\limits_{x \in P} A(x)$

Массовое обновление: $\text{update}(A, P, v) = A' : W \to G$, где

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

1.1.2. Требования к операторам

Как уже упоминалось ранее, на модель накладываются некоторые ограничения. Для того, чтобы понять, какого рода эти ограничения, обратимся к определениям массовых операций.

Рассмотрим массовый запрос. Во-первых, потребуем от оператора $+$ ассоциативность. Это свойство требуется для того, чтобы мы имели право разбивать всю область на группы и суммировать в них отдельно. Именно это свойство лежит в основе дерева отрезков.

Кроме того, потребуем от $+$ коммутативности. В одномерном случае подобное ограничение не накладываются, тут же они необходимо. Действи-

тельно, если отказаться от коммутативности, то уже в двумерном случае не ясно, какому порядку суммирования отдать приоритет. Выбирая разный порядок суммирования, мы можем получать разный результат. Таким образом, сам запрос становится абсолютно бессмысленным.

Подводя итог, получаем, что $< G, + >$ — коммутативная полугруппа.

Теперь обратимся к массовому обновлению. В основе многих методов применения массовых операций на одномерных массивах данных, лежит тот факт, что обновления можно комбинировать. То есть для какой-то области можно сначала накопить несколько обновлений, а потом их применить. Для того, чтобы мы могли использовать подобные идеи, потребуем от $\circ$ ассоциативность. Таким образом, $< G, \circ >$ — полугруппа.

В довершении всего, будем предполагать, что выполняется дистрибутивный закон. Это свойство нужно для того, чтобы можно было легко модифицировать вычисленную ранее сумму элементов в некоторой области после того, как все элементы этой области были умножены на некоторое число.

Подобная модель описывает не все возможные массовые запросы. Тем не менее, есть причины для того, чтобы использовать именно ее. Во-первых, многие естественные запросы, легко описываются в терминах этой модели. Во-вторых, она проста. Области имеют незамысловатый вид, введено всего две операции. В-третьих, на операции наложены требования, которые учитывают особенности реализации массовых операций в одномерном случае. Это позволяет надеяться на то, что задача имеет эффективное решение.

1.2. Классический подход

Существует широко известный в кругах ACM метод выполнения массовых операций для одномерного случай. Этот метод применим к структурам данных, в которых хранение информации организовано на основе дерева. Наиболее известный вариант его использования — так называемое «несогласованное дерево отрезков».

К сожалению, по загадочным причинам, упоминания даже об обыкновенном дереве отрезков встречаются крайне редко. Ознакомится с ним можно в книге [5].

1.2.1. Одномерный случай

Основная идея состоит в хранении в каждом внутреннем узле дополнительной информации — «несогласованности». Значение ее интерпретируется, как некоторое указание о том, какую операцию требуется применить ко всем элементам поддерева.

При попытке обращения к вершине применяется методика «раздать детям». Суть ее состоит в том, чтобы изменить значение в вершине так, как будто указанная операция была применена ко всему поддереву. После этого «дети» оповещаются о необходимости выполнения этой операции.

Важно отметить, что несогласованности можно «складывать» — операция $\circ$ ассоциативна. Если бы это было не так, то пришлось бы хранить в каждой вершине список всех команд. Поэтому время работы в худшем случае осталось бы таким же, как и без использования какой-либо структуры данных.

Для того, чтобы изменить значение в вершине так, как будто операция уже была применена ко всему поддереву, необходимо выполнение дистрибутивного закона.

При использовании этого подхода минимальными изменениями достигается правильность получаемого значения в любой посещенной вершине. Ключевой момент — необходимость начинать все обходы дерева из корня.

Существует и другой подход. В нем в каждой вершине также хранится несогласованность. Отличие состоит в том, что ее никогда не «раздают детям», а аккумулируют во время выполнения запроса. Этот подход является менее общим, так как для его применения требуется, чтобы оператор $\circ$ был коммутативным.

1.2.2. Многомерный случай

Начиная с двумерного случая, общеизвестного метода решения подобных задач не существует. Приведем некоторые соображения, которые могут дать начальные представления о природе этого факта.

Для обработки многомерных массивов данных часто используют квадродерево и кд-дерево. Получить представление о них можно в работах [6, 7].

Эти структуры хорошо себя зарекомендовали и нашли свое применение в самых разных областях — от компьютерной графики до баз данных.

Основное преимущество этих структур данных состоит в том, что они являются деревьями — на них вполне естественным образом распространяются идеи, используемые в одномерном случае. К сожалению, применение такого подхода не дает желаемого быстродействия. Дело в том, что уже в двумерном случае существуют прямоугольники специального вида, для получения статистики на которых требуется задействовать порядка N узлов дерева (N — сторона прямоугольника).

Возможно, для каких-то задач этого достаточно. Тем не менее, в большинстве случаев такая асимптотика неприемлема. Особенно остро проблема проявляется для разряженных данных.

Обратимся к многомерному дереву отрезков. Эта структура данных позволяет выполнять многие запросы за $O(\log^d N)$, где d — размерность пространства, а N — характерный размер области.

Несмотря на многообещающее начало, существует целый ряд проблем, которые возникают при попытке применить классическую методику к этой структуре данных.

Многомерное дерево отрезков не является деревом в строгом его понимании. К одной и той же ячейке путь может пролегать через разные вершины в зависимости от запроса. Концепция «раздачи детям» хорошо работает только если каждой вершине соответствует какая-то область, а области, соответствующим «детям» — разбиение на более мелкие дизъюнктные области.

1.3. Сужение задачи

К сожалению, решение задачи в общей формулировке найдено не было. Тем не менее, были получены интересные результаты для некоторого специфического подкласса исходной задачи. Несмотря на то, что надежда на обобщение полученного метода для решения исходной задачи достаточно мала, сам метод несомненно достоин рассмотрения.

Данная работа исследованию указанного подкласса: $+ \equiv \circ$, $< G, + >$ — абелева группа. Отметим, что требуется обратимость оператора $+$. Это

свойство лежит в основе всех дальнейших рассуждений.

Оказывается, в этом случае можно свести исходную задачу к хорошо изученной задаче [5, 8] *без массовых обновлений*.

Глава 2. Основные теоремы

Для того, чтобы упростить дальнейшее изложение, введем несколько операторов. Им соответствуют преобразования массивов, которые чем-то напоминают дифференцирование и интегрирование функций.

Для случая одномерной последовательности подобные преобразования описаны в книге [9]. Тем не менее, для данной работы свойства, связанные с многомерностью данных, являются принципиальными, поэтому приведение всех основных определений оказалось неизбежным. Кроме того, некоторые определения были незначительно изменены для лучшего соответствия исследуемой задаче.

Введение операторов и доказательство теорем, связанных с ними, позволяет оправдать преобразования, применение которых было бы трудно объяснить иначе. Несмотря на кажущуюся сложность, все дальнейшие рассуждения тривиальны и по большей части базируются на коммутативности операции $+$.

Связь операторов с программным кодом обсуждается в главе 3.

2.1. Обозначения

Префиксом будем называть область вида $P_{\mathbf{1},\mathbf{R}}$, $\mathbf{R} \in W$. *Суффиксом* — область вида $P_{\mathbf{L},\mathbf{N}}$, $\mathbf{L} \in W$.

Обозначим замену i-й компоненты вектора следующим образом: $\mathbf{x}_{x_i \leftarrow x'_i}$.

2.2. Основные утверждения

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

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

Лемма 1. *Операторы D_a и D_b перестановочны.*

Доказательство. Рассмотрим произвольный $A : W \to G$. Тогда

$$\begin{aligned}
D_a D_b A(\mathbf{x}) &= D_b A(\mathbf{x}) - D_b A(\mathbf{x}_{x_a \leftarrow x_a - 1}) = \\
&= (A(\mathbf{x}) - A(\mathbf{x}_{x_b \leftarrow x_b - 1}))) - (A(\mathbf{x}_{x_a \leftarrow x_a - 1} - A(\mathbf{x}_{x_a \leftarrow x_a - 1 x_b \leftarrow x_b - 1})) = \\
&= (A(\mathbf{x}) - A(\mathbf{x}_{x_a \leftarrow x_a - 1})) - (A(\mathbf{x}_{x_b \leftarrow x_b - 1}) - A(\mathbf{x}_{x_a \leftarrow x_a - 1 x_b \leftarrow x_b - 1})) = \\
&= D_a A(\mathbf{x}) - D_a A(\mathbf{x}_{x_b \leftarrow x_b - 1}) = \\
&= D_b D_a A(\mathbf{x}).
\end{aligned}$$

□

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

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

Лемма 2. *Лемма. Операторы S_a и S_b перестановочны.*

Доказательство. Для произвольного $A : W \to G$ и $\mathbf{x} \in W$:

$$\begin{aligned}
S_a S_b A(\mathbf{x}) &= \sum_{j=1}^{x_a} S_b A(\mathbf{x}_{x_a \leftarrow j}) = \\
&= \sum_{j=1}^{x_a} \sum_{k=1}^{x_b} A(\mathbf{x}_{x_a \leftarrow j, x_b \leftarrow k}) = \\
&= \sum_{k=1}^{x_b} \sum_{j=1}^{x_a} A(\mathbf{x}_{x_a \leftarrow j, x_b \leftarrow k}) = \\
&= \sum_{k=1}^{x_b} S_a A(\mathbf{x}_{x_b \leftarrow k}) = \\
&= S_b S_a(\mathbf{x}).
\end{aligned}$$

□

Лемма 3. *Лемма. Операторы S_i и D_i взаимообратны. $S_i D_i = D_i S_i = E$.*

Доказательство. Действительно,

$$\begin{aligned}
\forall A\ S_i D_i A(\mathbf{x}) &= \sum_{j=1}^{x_i} D_i A(\mathbf{x}_{x_i \leftarrow j}) = \\
&= \sum_{j=1}^{x_i} (A(\mathbf{x}_{x_i \leftarrow j}) - A(\mathbf{x}_{x_i \leftarrow j-1})) = \\
&= \sum_{j=1}^{x_i} A(\mathbf{x}_{x_i \leftarrow j}) - \sum_{j=1}^{x_i} A(\mathbf{x}_{x_i \leftarrow j-1}) = \\
&= \sum_{j=1}^{x_i} A(\mathbf{x}_{x_i \leftarrow j}) - \sum_{j=0}^{x_i-1} A(\mathbf{x}_{x_i \leftarrow j}) = \\
&= \sum_{j=1}^{x_i} A(\mathbf{x}_{x_i \leftarrow j}) - \sum_{j=1}^{x_i-1} A(\mathbf{x}_{x_i \leftarrow j}) = \\
&= S_i A(\mathbf{x}) - S_i A(\mathbf{x}_{x_i \leftarrow j-1}) = \\
&= D_i S_i A(\mathbf{x}).
\end{aligned}$$

С другой стороны $\sum\limits_{j=1}^{x_i} A(\mathbf{x}_{x_i \leftarrow j}) - \sum\limits_{j=1}^{x_i-1} A(\mathbf{x}_{x_i \leftarrow j}) = A(\mathbf{x})$. □

Лемма 4. *Лемма. Операторы S_a и D_b перестановочны.*

Доказательство. Действительно,

$$\begin{aligned}
S_a D_b A &= \sum_{j=1}^{x_a} D_b A(\mathbf{x}_{x_a \leftarrow j}) = \\
&= \sum_{j=1}^{x_a} (A(\mathbf{x}_{x_a \leftarrow j}) - A(\mathbf{x}_{x_a \leftarrow j, x_b \leftarrow x_b - 1})) = \\
&= \sum_{j=1}^{x_a} A(\mathbf{x}_{x_a \leftarrow j}) - \sum_{j=1}^{x_a} A(\mathbf{x}_{x_a \leftarrow j, x_b \leftarrow x_b - 1}) = \\
&= S_a A(\mathbf{x}) - S_a A(\mathbf{x}_{x_b \leftarrow x_b - 1}) = \\
&= D_b S_a A(\mathbf{x}).
\end{aligned}$$

□

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 этой главы.

Теорема 1. *Лемма.* $SD = DS = E$.

Доказательство.

$$\begin{aligned} SD &= S_1 S_2 \cdots S_d D_1 D_2 \cdots D_d = \\ &= S_1 D_1 S_2 D_2 \cdots S_d D_d = \\ &= E. \end{aligned}$$

Аналогично получаем тождество $DS = E$. □

Доказательство этой теоремы и было основной целью введения математических терминов в данную работу. Несмотря на кажущуюся простоту, правильная интерпретация операторов позволит ниже значительно упростить запрос на массовое обновление.

Глава 3. Практическое применение операторов

3.1. Связь задачи и операторов

Как уже упоминалось ранее, построенная математическая теория тесно связана с реальными преобразованиями, которые будут применены для решения задачи. Рассмотрим использование полученных результатов на практике.

Определение оператора S практически в точности соответствует определению запроса на сумму из главы 1.

Лемма 5. $SA(\mathbf{x}) = get\,(A, P_{\mathbf{1},\mathbf{x}})\,.$

Доказательство.

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

□

Таким образом, запись $SA(\mathbf{x})$ можно воспринимать, как обычный запрос на сумму на префиксе, не создавая при этом новых массивов. Ключевым моментом здесь является восприятие всей записи, как единого целого. Действительно, можно было бы сначала в явном виде построить массив SA, а после этого узнать значение в ячейке $\mathbf{x}$. Тем не менее, на практике подобный

подход слишком расточителен, поэтому приходится поступать хитрее и вычислять только те значения, которые действительно потребуются. Грамотная интерпретация математической нотации позволяет пользоваться результатами главы 2 без дополнительных затрат как по времени, так и по памяти.

Обратимся теперь к оператору D. Запись $DA(\mathbf{x})$ можно рассматривать двояко.

С одной стороны, ее можно интерпретировать как некоторый запрос, так же, как и в случае с оператором S. При этом не потребуется строить никаких дополнительных массивов, необходимо просто раскрыть определение оператора D. В полученной сумме будет 2^d слагаемых.

Однако существует и другой взгляд, который и будет использоваться в этой работе. Можно в явном виде построить массив DA и работать с ним. Именно такая интерпретация и приведет нас к желаемому результату. Первым шагом нему будет следующая лемма.

Лемма 6. *Сумма на префиксе в преобразованном массиве данных есть элемент исходного массива.*

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

Полученный результат показывает связь между преобразованным массивом (B) и исходным массивом (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 7. *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 $A'(\mathbf{y}) = \begin{cases} A(\mathbf{y}) + v, & \mathbf{x} \leq \mathbf{y}; \\ A(\mathbf{y}), & \mathbf{x} \not\leq \mathbf{y}. \end{cases}$ □

Подведем промежуточный итог. Используя результаты главы 2, было предложено преобразование исходных данных, которое значительно упро-

щает некоторые запросы на массовое обновление и при этом не приводит к потере информации. Перейдем к рассмотрению деталей использования предложенного преобразования.

3.2. Преобразование данных и запросов

Покажем, как изменилась исходная задача.

Построим массив $B = DA$. Поскольку оператору D соответствует конкретное преобразование, то это не должно составить никакого труда. Один из самых простых способов сделать это — использовать определение $D = D_1 D_2 \cdots D_d$ и по очереди применить дифференциальные операторы к исходному массиву.

Отметим, что без разбиения оператора D на составляющие, его определение включает в себя 2^d слагаемых. Это утверждение несложно проверить, подставив определения D_i и раскрыв все скобки. Опираясь на него, выполнение преобразования DA занимало бы по 2^d операции на каждую ячейку.

В то же время, введение изложенной выше теории позволило получить более простой и быстрый способ достижения того же результата. Действительно, описанный выше подход не только более прост для понимания, но и позволяет затрачивать всего $O(d)$ операций на каждую ячейку.

3.2.1. Преобразование запроса на сумму

Как уже обсуждалось ранее, действие оператора S тесно связано с запросом на сумму на префиксе. Например, $\text{get}\,(A, P_{\mathbf{1},\mathbf{x}}) = SA(\mathbf{x}) = SSB(\mathbf{x})$.

Последнее можно записать и по-другому:

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

Подобная запись показывает, что запрос на сумму на префиксе несколько усложнился. Тем не менее, его значение понятно: сумма превратилась в «сумму сумм». Как будет показано в главе 4, выполнение этого запроса вполне решаемая задача.

3.2.2. Преобразование запроса на прибавление

Как было показано ранее, изменение одной ячейки в массиве B эквивалентно массовому прибавлению к суффиксу массива A. Таким образом, массовое обновление сводится к изменению единственной ячейки массива B.

Это именно тот результат, который лежит в основе возможности использования классических структур данных. Путем усложнения запроса на сумму, было получено существенно упрощение запроса на массовое обновление. Получение статистической информации обычно гораздо проще, чем изменение данных, и это наблюдение верно и для рассматриваемой задачи.

3.3. Эквивалентность

Выше было показано, чему соответствуют запросы на префиксах или суффиксах. 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.

Для того, чтобы решить эту проблему, требуется воспользоваться формулой включения-исключения. Эта формула широко известна и неоднократно используется в курсе высшей математики. Тем не менее, приведем основные идеи.

Предположим, необходимо ответить на запрос $\mathrm{get}\,(A, P_{\mathbf{x},\mathbf{y}})$, при этом $x_1 \neq 1$. Тогда выразим исходный запрос следующим образом:

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

Повторяя подобное преобразование для остальных индексов, выразим исходный запрос только через ослабленные запросы. Для этого потребуется не более d шагов, поэтому итоговое число запросов не превзойдет 2^d.

Аналогичным образом можно выполнить массовые обновления. Здесь снова используется тот факт, что операция $+$ обратима, что позволяет одним массовым обновлениям компенсировать другие.

3.4. Эффективность

Поскольку большинство структур данных для работы с многомерными массивами данных содержат $O(\log^d n)$ в своей асимптотике, то умножение на 2^d не изменят асимптотическую оценку.

С практической точки зрения вопрос несколько более спорный. С одной стороны, явное выражение исходных запросов в этой форме мотивирует использовать такие структуры данных, как дерево Фенвика [8].

С другой стороны, умножение на такую большую константу кажется необоснованным замедлением алгоритма. Действительно, при выполнении запроса на очень маленьком прямоугольнике, в случае использования многомерного дерева отрезков можно добиться того, чтобы для запроса на сумму в асимптотике фигурировало $O(\log^d w)$, где w — характерный размер области.

3.5. Примеры

Для того, чтобы лучше разобраться с тем, что же происходит, рассмотрим несколько примеров.

3.5.1. Одномерный случай

Рассмотрим одномерный массив A. Применим преобразование $DA = B$. Получаем, что $B_i = A_i - A_{i-1}$. Кроме того, по доказанным леммам, $A_i = \sum\limits_{j=1}^{i} B_j$.

Изменение одного элемента B_k влечет за собой увеличение $A_k, A_{k+1}, \ldots$. Для того, чтобы увеличить только ячейки $A_k, A_{k+1}, \ldots, A_t$ на величину v, достаточно прибавить v к B_k и вычесть из B_{t+1}.

Для того, чтобы вычислить сумму на префиксе, требуется найти «сумму сумм»: $\sum\limits_{j=1}^{k} A_j = \sum\limits_{j=1}^{k} \sum\limits_{l=1}^{j} B_l$.

3.5.2. Двумерный случай

Рассмотрим, как и ранее, массив A, на этот раз двумерный. Построим массив $B = DA$. Полный вид применяемого преобразования выглядит следующим образом: $B_{i,j} = A_{i,j} - A_{i-1,j} - A_{i,j-1} + A_{i-1,j-1}$.

Сосредоточимся на выражении исходных запросов через ослабленные. Запрос на сумму достаточно прост:

$$\begin{aligned}\mathrm{get}(A, P_{\mathbf{x},\mathbf{y}}) = \mathrm{get}(A, P_{\mathbf{1},\mathbf{y}}) - \\ - \mathrm{get}(A, P_{\mathbf{1},(x_1-1,y_2)}) - \\ - \mathrm{get}(A, P_{\mathbf{1},(y_1,x_2-1)}) + \\ + \mathrm{get}(A, P_{\mathbf{1},(x_1-1,x_2-1)}).\end{aligned}$$

Аналогичным образом выражается массовое обновление. Стоит отметить, что приведенные выше формулы достаточно часто используются для эффективного получения статистики на прямоугольниках, в то время как использование самого преобразование массива является нововведением.

Глава 4. Сведение задачи

Вся основная работа была уже проделана. Исходная задача была сведена к другой, которую и надо решить.

Сосредоточимся на выполнении запроса на сумму сумм. На первый взгляд, это место еще более непонятное, чем то, что было изначально, и мы только удалились от решения. Оказывается, это не так.

Для выполнения дальнейших доказательств будет удобно воспользоваться нотацией, предложенной в работе [10]:

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

4.1. Применение многочленов

Лемма 8. *Для любого* $\mathbf{x} \le \mathbf{z}$ *значение* $B(\mathbf{x})$ *входит в* $S^2B(\mathbf{z})$ *ровно* $\prod_{i=1}^{d}(z_i - x_i)$ *раз.*

Доказательство. Действительно $S^2B(\mathbf{z}) = \sum\limits_{\mathbf{1}\le\mathbf{y}\le\mathbf{z}} \sum\limits_{\mathbf{1}\le\mathbf{y}'\le\mathbf{y}} B(\mathbf{y}')$. Число вхождений значения $B(\mathbf{x})$ в эту сумму равно $\sum\limits_{\mathbf{1}\le\mathbf{y}\le\mathbf{z}} \sum\limits_{\mathbf{1}\le\mathbf{y}'\le\mathbf{y}} [\mathbf{y}' = \mathbf{x}]$.

Заметим, что $\sum\limits_{\mathbf{1}\le\mathbf{y}'\le\mathbf{y}} [\mathbf{y}' = \mathbf{x}] = \begin{cases} 1, & \mathbf{x} \le \mathbf{y}; \\ 0, & \mathbf{x} \not\le \mathbf{y}. \end{cases}$

Получаем: $\sum\limits_{\mathbf{1}\le\mathbf{y}\le\mathbf{z}} [\mathbf{x} \le \mathbf{y}] = \sum\limits_{\mathbf{x}\le\mathbf{y}\le\mathbf{z}} 1 = \prod_{i=1}^{d}(z_i - x_i)$.

□

Основная идея — 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.

Прежде чем двигаться дальше, необходимо пояснить, что имеется в виду под умножением элемента абелевой группы $< G, + >$ на число. Можно было бы решить, что неявно используется некоторое кольцо $< G, +, \times >$, но это не так. Отметим, 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 друг с другом. Массив, содержащий эти многочлены, обозначим как C.

Рассмотрим $\sum_{\mathbf{1} \leq \mathbf{x} \leq \mathbf{z}} C_{\mathbf{x}}$. Получим некий многочлен, смысл которого следующий: сколько раз все соответствующие элементы войдут в запрос на сумму $\mathrm{get}(A, P_{\mathbf{1},\mathbf{z}})$. Это означает, что

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

Таким образом, для того, чтобы ответить на запрос на сумму на префиксе, необходимо вычислить сумму на префиксе в массиве C, а после этого в получившийся многочлен подставить параметры запроса.

4.2. Подведение итогов

Для решения исходной задачи требуется:

1. Преобразовать исходные данные, тем самым упростив массовые обновления.

2. Заменить элементы нового массива на многочлены, что позволяет эффективно отвечать на запросы на сумму.

Обсудим, что происходит при массовом обновлении. Поскольку при выполнении ослабленного запроса изменяется одна ячейка, то поменяется также только один многочлен. Таким образом, прибавление на суффиксе было сведено к прибавлению нового многочлена к одной ячейке. Тонким местом здесь является построение этого многочлена в том случае, если умножение на

целое число не является тривиальной операцией. Этот вопрос обсуждается в главе 5.

Для решения финальной задачи достаточно поддержки двух запросов:

1. Сумма на префиксе.
2. Изменения одного элемента.

Для решения этой задачи существует множество структур данных, позволяющих выполнять указанные запросы за $O(\log^d N)$. Для многомерного случая подобная асимптотика является во многих случаях вполне приемлемой. Подробное обсуждение выбора структуры данных происходит в главе 5.

Глава 5. Замечания по реализации

5.1. Структура данных

Как уже упоминалось ранее, предложенный метод не зависит от структуры данных. Поэтому для демонстрации эффективности использования метода предлагается обзор из нескольких популярных структур данных, широко используемых для работы с многомерными данными.

5.1.1. Многомерное дерево отрезков

Многомерное дерево отрезков [5] является обобщением дерева отрезков для работы с многомерными массивами данных. Вопреки названию, эта структура данных не является деревом. Этот вопрос освещен в главе 1.

Дерево отрезков использует в $O(2^d)$ больше памяти, чем исходные данные. Это недостаток компенсируется возможностью эффективной работы с разреженными данными.

К сожалению, при увеличении размерности пространства, реализация данной структуры данных значительно усложняется, особенно для случая разреженных данных. Для решения этой проблемы можно применять шаблоны, но при этом имеется риск потери производительности.

5.1.2. Многомерное дерево Фенвика

«Дерево Фенвика» — неофициальное название структуры данных, представленной Питером Фенвиком в работе [8], которое получило широкое распространение в олимпиадных кругах России. У дерева Фенвика есть несколько важных преимуществ в сравнении с деревом отрезков:

- Для построения этой структуры данных не требуется дополнительной памяти. Это свойство особенно полезно при работе с данными большой размерности, так как накладные расходы при использовании дерева отрезков растут экспоненциально с увеличением размерности данных.
- Программный код, реализующий эту структуру данных, необычайно прост. Это позволяет получать значительный выигрыш во времени по сравнению с деревом отрезков.

Существуют характеристики, по которым дерево Фенвика проигрывает дереву отрезков. Перечислим основные:

- Принципиальная невозможность использования концепции работы с «несогласованностью». Этот вопрос обсуждался в главе 1.
- Невозможность эффективной работы с разреженными данными.

5.1.3. KД-дерево

Эта структура данных, впервые представленная в статье [7], изначально предназначена для работы с разреженными многомерными данными. КД-дерево сравнительно просто в реализации и хорошо себя зарекомендовало на практике. Разнообразные модификации этой структуры данных применяются в базах данных и вычислительной геометрии.

Тем не менее, в случае плотного набора данных, асимптотические оценки на время работы в худшем случае значительно хуже, чем у вышеперечисленных структур данных. Этот вопрос обсуждался в главе 1.

5.2. Оценка времени работы и потребляемой памяти

5.2.1. 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 запросов 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 + над множеством G элементарной — на ее выполнение уходит $O(1)$ времени. Тогда после преобразования каждому элементу соответствует многочлен степени 2^d. Сложение двух многочленов выполняется за $O(2^d)$. Таким образом, асимптотическая оценка на выполнение запроса осталась $O(\log^d N)$.

Обратимся теперь к массовым обновлениям. По аналогии с предыдущими рассуждениями, 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. При работе с примитивными типами данных 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. Коэффициенты многочлена не превышают N^d, поэтому общая оценка времени, затраченного на построение нового многочлена, есть $O(2^d \log(N^d)) = O(\log^d N)$ [11].

5.2.2. 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, число ненулевых ячеек может также возрасти, но не более чем в 2^d раз.

Подводя итог, для большинства структур данных применение разработанного метода не ухудшает асимптотическую оценку на время их работы. Тем не менее, при возрастании размерности массива данных константа, скрытая асимптотической оценке, возрастает экспоненциально. Поэтому при больших размерностях метод следует использовать с осторожностью.

Заключение

В работе был выбран узкий случай $+ \equiv \circ$, $< G, + >$ — абелева группа. Для него был разработан общий метод сведения задачи с массовыми обновления 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 все ячейки массива используются, то стоит сделать выбор в пользу дерева Фенвика. С другой стороны, в случае сильно разреженных данных, можно использовать разреженное дерево отрезков. Подобная гибкость может позволить применять метод в таких областях, как базы данных.

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.

Источники

[1] *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. МЦНМО, 2004.

[2] *Шень А.* Программирование: теоремы и задачи. МЦНМО, 2004.

[3] *Tarjan R.E.* Data Structures and Network Algorithms. SIAM, 1983.

[4] *Кнут Д.Э.* Искусство программирования, т. 3. М.: Вильямс, 2007.

[5] *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение. Мир, 1989.

[6] *Романовский И.В.* Дискретный анализ. СПб.: Диалог, 2004.

[7] *Bentley J.L.* Multidimensional binary search trees used for associative searching // *Communications of the ACM.* 1975. no. 9. Pp. 509 – 517.

[8] *Fenwick P.M.* A New Data Structure for Cumulative Frequency Tables // *Software — Practice and experience.* 1994. no. 3. P. 327 – 336.

[9] *Кнут Д., Грэхем Р., Паташник О.* Конкретная математика. Основание информатики. М.: Вильямс, 2005.

[10] *Кнут Д.Э.* Искусство программирования, т. 1. М.: Вильямс, 2007.

[11] *Brauer A.* On addition chains // *Bulletin of the American Mathematical Society.* 1939. Pp. 736 – 739.

Приложение. Реализация

Для демонстрации эффективности метода ниже приведен код на языке программирования *Java*. Основная цель примера — показать, что для использование метода не требуется вносить существенных изменений в реализацию структур данных. Для достижения требуемого эффекта было использовано дерево Фенвика, которое отличается простотой реализации.

Приведенный класс позволяет эффективно прибавлять целое значение к прямоугольнику (метод `add`) и получать значение суммы элементов в прямоугольнике (метод `get`).

```
1  public class Fenwick2D {
2
3    private int[][] a, b, c, d;
4    private int n, m;
5
6    public Fenwick2D(int n, int m) {
7      this.n = n;
8      this.m = m;
9      a = new int[n][m];
10     b = new int[n][m];
11     c = new int[n][m];
12     d = new int[n][m];
13   }
14
15   public int get(int x1, int y1, int x2, int y2) {
16     if (x1 > x2 || y1 > y2) return 0;
17     --x1;
18     --y1;
19     return get(x2, y2) - get(x1, y2) - get(x2, y1) + get(x1, y1);
20   }
21
```

```
22    public void add(int x1, int y1, int x2, int y2, int v) {
23      if (x1 > x2 || y1 > y2) return;
24      ++x2;
25      ++y2;
26      add(x1, y1, v);
27      add(x1, y2, -v);
28      add(x2, y1, -v);
29      add(x2, y2, v);
30    }
31
32    private void add(int x1, int x2, int v) {
33      int ta = v * (x1 - 1) * (x2 - 1);
34      int tb = -v * (x2 - 1);
35      int tc = -v * (x1 - 1);
36      int td = v;
37      for (int ti = x1; ti < n; ti |= ti + 1) {
38        for (int tj = x2; tj < m; tj |= tj + 1) {
39          a[ti][tj] += ta;
40          b[ti][tj] += tb;
41          c[ti][tj] += tc;
42          d[ti][tj] += td;
43        }
44      }
45    }
46
47    private int get(int x1, int x2) {
48      int sa = 0, sb = 0, sc = 0, sd = 0;
49      for (int ti = x1; ti >= 0; ti = (ti & (ti + 1)) - 1) {
50        for (int tj = x2; tj >= 0; tj = (tj & (tj + 1)) - 1) {
51          sa += a[ti][tj];
52          sb += b[ti][tj];
53          sc += c[ti][tj];
54          sd += d[ti][tj];
55        }
56      }
57      return sa + sb * x1 + sc * x2 + sd * x1 * x2;
58    }
59  }
```

Fenwick2D.java