3

Алгоритмы «разделяй и властвуй»

Эта глава посвящена практике парадигмы разработки алгоритмов «разделяй и властвуй» применительно к трем основным задачам. Первый пример — это алгоритм подсчета количества инверсий[1] массива (раздел 3.2). Данная задача предназначена для измерения подобия между двумя ранжированными списками. Это так называемая «совместная фильтрация», когда, опираясь на ваши знания о предпочтениях человека и предпочтениях других людей, ему предоставляются полезные рекомендации. Второй алгоритм типа «разделяй и властвуй» — это восхитительный рекурсивный алгоритм Штрассена для умножения матриц, который работает лучше по сравнению с очевидным итеративным методом (раздел 3.3). Третий алгоритм, который является продвинутым и дополнительным материалом, предназначен для решения фундаментальной задачи вычислительной геометрии: вычисление ближайшей пары точек на плоскости (раздел 3.4)[2].

3.1. Парадигма «разделяй и властвуй»

Мы уже рассматривали канонический пример алгоритма «разделяй и властвуй», `MergeSort` (раздел 1.4). В более общем плане парадигма разработки алгоритмов «разделяй и властвуй» состоит из трех концептуальных шагов.

ПАРАДИГМА «РАЗДЕЛЯЙ И ВЛАСТВУЙ»

1. *Разделить* входные данные на более мелкие подзадачи.
2. *Решить* подзадачи рекурсивным методом.
3. *Объединить* решения подзадач в решение исходной задачи.

[1] В информатике и дискретной математике последовательность имеет инверсию там, где два ее элемента находятся вне своего естественного порядка. Например, пусть дана числовая последовательность 1, 2, 5, 7, 4, 6, тогда пары (5, 4), (7, 4) и (7, 6) образуют инверсии в этой последовательности. — *Примеч. пер.*

[2] Демонстрация в разделах 3.2 и 3.4 черпает вдохновение из главы 5, «Проектирование алгоритмов», Джона Клейнберга и Эвы Тардос (Algorithm Design, Jon Kleinberg, Éva Tardos Pearson, 2005).

Например, в алгоритме `MergeSort` шаг «разделить» разбивает входной массив на левую и правую половины, шаг «решить» использует два рекурсивных вызова для сортировки левого и правого подмассивов, а шаг «объединить» реализуется подпрограммой слияния `Merge` (раздел 1.4.5). В алгоритме `MergeSort` и многих других алгоритмах этот последний шаг требует наибольшей изобретательности. Существуют также алгоритмы типа «разделяй и властвуй», в которых необходимо быть изобретательным уже на первом шаге (см. `QuickSort` в главе 5) или же в спецификации рекурсивных вызовов (раздел 3.2).

3.2. Подсчет инверсий за время O (*n* log *n*)

3.2.1. Задача

В этом разделе рассматривается задача вычисления количества инверсий массива. *Инверсия* массива — это пара элементов, которые расположены «вне своего естественного порядка». Это означает, что элемент, который в массиве встречается ранее, больше, чем тот, который встречается позже.

ЗАДАЧА: ПОДСЧЕТ ИНВЕРСИЙ

Вход: массив A разных целых чисел.

Выход: количество инверсий — число пар (i, j) индексов массива, где $i < j$ и $A[i] > A[j]$.

Например, если массив A отсортирован, он не имеет инверсий. Вы должны убедиться, что обратное также верно: каждый неотсортированный массив имеет по крайней мере одну инверсию.

3.2.2. Пример

Рассмотрим следующий ниже массив длиной 6:

1	3	5	2	4	6

Сколько инверсий имеет этот массив? В глаза бросаются значения 5 и 2 (соответствуя $i = 3$ и $j = 4$). Имеются две другие неупорядоченные пары: 3 и 2, а также 5 и 4.

ТЕСТОВОЕ ЗАДАНИЕ 3.1

Каково наибольшее количество инверсий для 6-элементного массива?

а) 15

б) 21

в) 36

г) 64

(Решение и пояснение см. в разделе 3.2.13.)

3.2.3. Совместная фильтрация

Для чего необходимо подсчитывать количество инверсий в массиве? Одна из причин заключается в том, чтобы вычислить числовую меру качественного подобия, которая количественно определяет, насколько близки два ранжированных списка друг к другу. Например, предположим, что я прошу вас и вашего друга ранжировать десять фильмов, которые вы оба смотрели, от любимого до наименее любимого. Являются ли ваши вкусы «похожими» или же они «различаются»? Один из способов решить эту задачу количественно — использовать 10-элементный массив A: пусть $A[1]$ содержит оценку вашим другом вашего любимого фильма в его списке, $A[2]$ — личную оценку вашим другом вашего второго любимого фильма... и $A[10]$ — личную оценку вашим другом вашего наименее любимого фильма (вы его поставили на 10-е место). Таким образом, если вашим любимым фильмом является «Звездные войны», но у вашего друга он только пятый в его списке, то $A[1] = 5$. Если ваши рейтинги идентичны, этот массив будет отсортирован и не будет иметь инверсий. Чем больше инверсий массив имеет, тем больше пар фильмов, по которым вы не сходитесь во мнении об их достоинствах, и тем больше различаются ваши предпочтения.

Одна из задач, для которой вам может понадобиться вычисление меры подобия рейтингов, — это *совместная фильтрация* (collaborative filtering), способ создания рекомендаций. Каким образом веб-сайты находят рекомендации для товаров, фильмов, песен, новостей и так далее? При использовании совместной фильтрации идея состоит в том, чтобы идентифицировать других людей, которые имеют аналогичные с вами предпочтения, чтобы затем рекомендовать вам товары, которые были популярны у них. Следовательно, алгоритмы совместной фильтрации осуществляют математическую формализацию качественного понятия «подобия» между предпочтениями людей. Вычисление инверсий частично помогает в этом вопросе.

3.2.4. Поиск полным перебором

Как быстро мы можем вычислить количество инверсий в массиве? Если мы лишены воображения, то в нашем распоряжении есть полный перебор, или метод «грубой силы» (brute force).

ПОЛНЫЙ ПЕРЕБОР ДЛЯ ПОДСЧЕТА ИНВЕРСИЙ

Вход: массив A из n разных целых чисел.

Выход: количество инверсий массива A.

```
numInv := 0
for i := 1 to n – 1 do
    for j := i + 1 to n do
        if A[i] > A[j] then
            numInv := numInv + 1
return numInv
```

Это, безусловно, правильный алгоритм. А как насчет времени его исполнения? Из решения тестового задания 3.1 мы знаем, что количество итераций цикла возрастает квадратично в зависимости от длины n входного массива. Поскольку алгоритм выполняет постоянное число операций в каждой итерации цикла, его асимптотическое время исполнения равняется $\Theta(n^2)$. Но помните ли вы мантру опытного проектировщика алгоритмов: *можно ли добиться лучшего*?

3.2.5. Подход «разделяй и властвуй»

Ответ: да, лучшего добиться можно, и решением будет алгоритм типа «разделяй и властвуй», который исполняется за время $O(n \log n)$, что гораздо лучше по сравнению с алгоритмом поиска методом «грубой силы». Шаг «разделить» будет точно таким же, как и в алгоритме `MergeSort`, с одним рекурсивным вызовом для левой половины массива и одним для правой половины. Чтобы понять остальные операции, которые должны быть выполнены вне двух рекурсивных вызовов, давайте отнесем инверсии (i, j) массива A длиной n к одному из трех типов.

1. *Левая инверсия*: инверсия, в которой оба индекса, i, j, находятся в первой половине массива (то есть $i, j \leq \frac{n}{2}$).
2. *Правая инверсия*: инверсия, в которой оба индекса, i, j, находятся во второй половине массива (то есть $i, j > \frac{n}{2}$).
3. *Разделенная инверсия*: инверсия, в которой i находится в левой половине, а j — в правой половине (то есть $i \leq \frac{n}{2} < j$).

Например, в шестиэлементном массиве в разделе 3.2.2 все три инверсии являются разделенными.

Первый рекурсивный вызов в первой половине входного массива рекурсивно подсчитывает все левые инверсии (и больше ничего). Точно так же второй рекурсивный вызов подсчитывает все правые инверсии. Дальнейшая задача состоит в подсчете инверсий, не выявленных ни одним рекурсивным вызовом, — разделенных инверсий. Это шаг «объединить» алгоритма, и нам нужно будет реализовать для него специальную линейную функцию, аналогичную подпрограмме слияния `Merge` в алгоритме `MergeSort`.

3.2.6. Высокоуровневый алгоритм

Наш подход «разделяй и властвуй» транслируется в приведенный ниже псевдокод; подпрограмма подсчета разделенных инверсий `CountSplitInv` на данный момент не реализована.