Selection and Sorting in the "Restore" Model

Saint Petersburg State University

6 марта 2019 г.

Сортировка со временем $O(n \lg n)$

Теорема 1. На вход даны n натуральных чисел принадлежащие $[U]$, а также k. Существует алгоритм находящий k-ый наименьший элемент за время $O(n \lg U)$ и используя $O(\lg n)$ дополнительных слов в $\mathrm{Restore}$ модели.
Доказательство Следующий алгоритм подходит.

1. $\mathrm{Шаг}\ 1$. Сводим задачу к $U/2$. Идем навстречу двумя указателями и если первый бит левого слова 1, а правого 0, то меняем местами у чисел все биты кроме главных.
2. $\mathrm{Шаг}\ 2$. Сравниваем место встречи указателей с k и рекурсивно запускаем алгоритм поиска и востановления, запоминая в буфер какой бит должен быть на первом месте.
3. $\mathrm{Шаг}\ 3$. Востанавливаем изначальную последовательность при помощи главных битов которые остались на месте.

Примечание Запуская Шаг 2 также на второй части мы получим алгоритм сортировки работающий за тоже время.

Лемма о перемещении

Лемма 1. На вход дана последовательность длины n чисел из $[2^w]$ и число l. Мы можем переместить l первых битов каждого числа в начало массива за время $O(n)$, используя $O(l \lg n)$ памяти. Алгоритм обратим за тоже время.

Доказательство. Разобьем все на блоки длины $a = [\lg n]$. И для каждого блока решим эту задачу наивно при этом сохранив порядок чисел в каждом блоке, используя $O(awl)$ дополнительных битов. Теперь массив имеет вид $A_1, B_1, A_2, B_2, \ldots$. Сделаем массив $A_1, A_2, \ldots, B_1, B_2, \ldots$ используя алгоритм Фитча за время $O(n/a) \lg(n/a) = O(n)$, причем он сделает лишь $O(n/a)$ перестановок, каждая из которых потребует $O(a)$ времени. Итого потребуется лишь $O(n)$ времени. Обратимость алгоритма и время его работы очевидны.

Лемма о кодировке

Лемма 2. Дана строка $s = s_1 s_2 s_3 \ldots$, где $s_i \in [2^l]$ и суффиксный код для алфавита $[2^l]$. С длиной не более $l < L < w$, тогда мы можем закодировать s за время $O(n + 2^L)$ используя $O(2^L)$ дополнительной памяти в предположении что код сжимает. Расшифровка также возможна за это же время.

Лемма о кодировке

Лемма 2. Дана строка $s = s_1 s_2 s_3 \ldots$, где $s_i \in [2^l]$ и суффиксный код для алфавита $[2^l]$. С длиной не более $l < L < w$, тогда мы можем закодировать s за время $O(n + 2^L)$ используя $O(2^L)$ дополнительной памяти в предположении что код сжимает. Расшифровка также возможна за это же время.

Доказательство. Используем следующий алгоритм.

1. Шаг 1. Пусть c_i равно l минус длина кодировки s_i, а $C_i = c_1 + \cdots + c_i$. Найдем такое i, что C_i минимально. Это делается за $O(n)$, используя $O(1)$ дополнительной памяти.
2. Шаг 2. Используя наивный алгоритм зашифруем строку $s_{i+1} \ldots s_n s_1 \ldots s_i$
3. Шаг 3. Переместим зашифрованную строку в начало массива. Легко делается за линию.

Лемма о кодировке

Лемма 2. Дана строка $s = s_1 s_2 s_3 \dots$, где $s_i \in [2^l]$ и суффиксный код для алфавита $[2^l]$. С длиной не более $l < L < w$, тогда мы можем закодировать s за время $O(n + 2^L)$ используя $O(2^L)$ дополнительной памяти в предположении что код сжимает. Расшифровка также возможна за это же время.

Доказательство. Используем следующий алгоритм.

1. Шаг 1. Пусть c_i равно l минус длина кодировки s_i, а $C_i = c_1 + \dots + c_i$. Найдем такое i, что C_i минимально. Это делается за $O(n)$, используя $O(1)$ дополнительной памяти.
2. Шаг 2. Используя наивный алгоритм зашифруем строку $s_{i+1} \dots s_n s_1 \dots s_i$
3. Шаг 3. Переместим зашифрованную строку в начало массива. Легко делается за линию.

Для окончания доказательства осталось понять почему второй шаг потребует не более чем $O(l)$ дополнительных битов. Заметим что код сжимает как подстроку $s_{i+1} \dots s_j$ при $j > i$, так и $s_{i+1} \dots s_n s_1 \dots s_j$, при $j \leq i$, так как количество сэкономленных битов равно $C_n - C_i + C_j \geq C_n \geq 0$

Сортировка за $O(n \lg n)$

Теорема 2. На вход даны n натуральных чисел принадлежащие $[U]$, а также k. Существует алгоритм находящий k-ый наименьший элемент за время $O(n \lg n)$ и используя $O(\lg n)$ дополнительных слов в $\mathrm{Restore}$ модели.

Сортировка за $O(n \lg n)$

Теорема 2. На вход даны n натуральных чисел принадлежащие $[U]$, а также k. Существует алгоритм находящий k-ый наименьший элемент за время $O(n \lg n)$ и используя $O(\lg n)$ дополнительных слов в $\mathrm{Restore}$ модели.

Доказательство Мы модернизируем второй шаг из алгоритма первой теоремы. Зафиксируем $\delta > 0$. Указатели делят массив на 2 части. Если доля одной из частей хотя бы $(1 - \delta)$ мы перейдем к шагу $3'$ описанному ниже.

Сортировка за $O(n \lg n)$

Теорема 2. На вход даны n натуральных чисел принадлежащие $[U]$, а также k. Существует алгоритм находящий k-ый наименьший элемент за время $O(n \lg n)$ и используя $O(\lg n)$ дополнительных слов в $\mathrm{Restore}$ модели.
Доказательство Мы модернизируем второй шаг из алгоритма первой теоремы. Зафиксируем $\delta > 0$. Указатели делят массив на 2 части. Если доля одной из частей хотя бы $(1 - \delta)$ мы перейдем к шагу $3'$ описанному ниже.
Шаг 3' НУО, пусть у нас $(1 - \delta)n$ нулей. Тогда используя лемму о перемещении перенесем первые биты в начало. Получим в начале строку где много нулей. Закодируем её используя лемму о кодировке. С $l = 2, L = 3$, где 00 отправляется в 0. Тогда мы освободим хотя бы $n/4$ битов. И запустим алгоритм выборal(сортировки) для read-only model, имеющий время $O(n)(O(n \lg n)$ для сортировки) и использующий αn битов, для некоторого фиксированного $\alpha > 0$.

Порязрядная сортировка

Теорема На вход даны n натуральных чисел принадлежащие $[U]$, а также $b < \min(n, U)$. Существуе алгоритм сортирующий за время $O(n \lg_b U)$ и дополнительную память $O(b^O(1) \lg n)$
Доказательство Положим b степенью двойки. Будем также считать, что $b < n^3$ иначе известен алгоритм решающий за лучешее время. Мы изменим шаг 1 алгоритма из алгоритма первой теоремы.
Шаг 1'. Мы отсортируем по первым $\lg b$ битам. Это можно сделать за 2 прохода по массиву. Первый считает количество чисел начинающихся с заданных битов. Раставляем указатели, И проходом слева направо раставляем сами числа.
(Замечание) Если на каком-то шаге чисел меньше чем b мы не можем применить шаг 1, но мы можем просто использовать обычную read-only сортировку за $O(b \lg b)$ и $O(b)$ памяти. Такие операции в сумме займут не более чем $O(b^2 \lg b) = o(n)$

Лемма о сортировке в read-only

Лемма 3. На вход даны n натуральных чисел принадлежащие $[U]$, Зафиксируем $n \leq T \leq n^2$ и $\epsilon > 0$, тогда существует алгоритм сортировки работающий за время $O(T + RAM - SORT(\mathrm{n}))$, использующий $O((n^2/T)\lg n + n^\epsilon \lg U)$ памяти.

Доказательство Ограничим память RAM-SORTалгоритма $O(n \lg n + n^\epsilon \lg U)$. Построим $O(b)$ квантилей которые разбивают массив на подмассивы длины не более чем n/b. Алгоритм Мунро и Патерсона позволяет нам это сделать используя $O(b^{O(1)} \lg^{O(1)} n \lg U)$ битов и $O(n/(b \lg_b n))$ операций слияния и сортировки, для подмножеств длины не более $O(b \lg_b n)$. В сумме мы потратим $O((n/(b \lg_b n))\text{RAM-SORT}(n/(b \lg_b n)) = O(\text{RAM-SORT}(n))$ времени. Отсортировав каждую группу вместе с квантилями, узнаем группу где лежит каждый элемент(оставим на него ссылку). Теперь рекурсивно отсортируем каждый интервал поотдельности. Всего шагов рекурсии будет $\lg_b n$. Подставляя $b = n^{\Theta(\epsilon)}$ получим требуемую сложность.

Продолжение доказательства леммы

Вернемся к лемме. Если $T > n \lg n$, то алгоритм уже известен. Иначе, найдем $O(T/n)$ квантилей. Которые делят на куски не более n^2/T. Это тоже известный алгоритм. Наивно найдем каждый кусок и запустим алгоритм выше.

Финальный результат

Теорема 5. На вход даны n натуральных чисел принадлежащие $[U]$, а также параметр b и константа ϵ. Существует алгоритм сортирующий за время $O(n \lg_b n + \textbf{RAM-SORT}(n))$ и используя $O(b^{O(1)} \lg n + \min(n^{O(\epsilon)}, lg^{O(\epsilon)} U)$ дополнительных слов в $\mathbf{Restore}$ модели.

Доказательство. Мы модернизируем алгоритм теоремы 1 в стиле теоремы 2. Мы не будем вдаваться в ненужные подробности.

Подставляя $b = n^{\Theta(\epsilon)}$ мы получаем алгоритм работающий за $O(\text{RAM-SORT}(n))$ время и $O(n^{\epsilon})$ памяти.

Подсчёт числа инволюций

Лемма 1 На вход даны n натуральных чисел принадлежащие $[b]$, где b довольно мало. Существует алгоритм подсчитывающий количество инверсий за время $O((n/b)\text{RAM-SORT}(b))$ и память $O(n)$ в read-only моделе.
Доказательство Разобьем все на блоки длины b. И в каждом из них поотдельности посчитаем инволюций. Теперь посчитаем количество инволюций между блоками одним указателем, каждый раз обновляя количество чисел которое встретилось. Для каждого блока это делается за линию.

Еще одна лемма

Лемма 2 Дана последовательность I непересекающихся интервалов $[i_1, j_1) \dots [i_b, j_b)$ покрывающие интервал $[l, u)$. Мы можем переставить элементы $A[l], \dots, A[u-1]$ в $A[i_1], \dots, A[j_1 - 1], A[i_2], \dots, A[i_b], \dots, A[j_b - 1]$ за время $O(n)$ используя дополнительно $b^{O(1)}$ памяти.

Доказательство. Будем поддерживать два связных списка I и $\hat{I}$, где они будут хранится в порядке возрастания координат. Если $b < n/\lg n$, то сортируем их по первой координате. Иначе $b^2 > n$ и можно применить наивный алгоритм.

Пусть первый интервал в I это $[l, j)$. Разберем 3 случая.

1. Если $len([i_1, j_1)) = len([l, j))$. Тогда все наивно, удаляем $[l, j)$ из I и $\hat{I}$ это делается за линию, если мы сохраним указатель на местоположение $[l, j)$ в I(храним и обратные указатели для пункта 3). И рекурсивно запускаемся полагая $l = j$.
2. $len([i_1, j_1)) < len([l, j))$. Тогда подразбиваем $[l, j)$ и запускаемся рекурсивно.
3. $len([i_1, j_1)) > len([l, j))$. Тогда подразбиваем $[i_1, j_1)$

И еще одна

Лемма 3. Дан read-only массив $K[1], \ldots, K[n] \in [b]$. Тогда мы можем переставить массив $A[1], \ldots, A[n]$ уважая стабильный порядок K. За время $O(n \lg_b n)$ и $O(b^{O(1)} \lg n)$ дополнительной памяти.

Доказательство. Разделим массив A на b равных частей. Рекурсивно решаем задачу для меньших блоков. Тогда массив имеет вид $B_{11}, B_{12}, \ldots, B_{bb}$, где B_{ij} это список элементов в i-ом блоке таких что $A[k] = j$. А мы хотим чтобы массив имел вид $B_{11}, B_{21}, \ldots, B_{bb}$. А эту задачу мы уже решили в лемме 2.

Замечание Обратимость алгоритма при наличии K очевидна

Последняя лемма

Лемма 4. На вход даны n натуральных чисел принадлежащие $[U]$, Зафиксируем $n \leq T \leq n \lg n$ и $\epsilon > 0$, тогда существует алгоритм сортировки работающий за время $O(T + \text{RAM-SORT}(n) + \text{RAM-INV}(n))$, использующий $O((n^2/T) \lg n + n^\epsilon \lg U)$ битов в read-only модели.

Доказательство Воспользуемся алгоритмом сортировки за $O(\text{RAM-SORT}(n))$, использующий $O(n \lg n + n^\epsilon \lg U)$ битов. Заменим сами числа на их индексы и запустим на них алгоритм подсчета инволюций.

Далее как и в лемме о сортировке разобьем массив при помощи $O(T/n)$ квантилей. И будем решать задачу рекурсивно для каждого подотрезка. Осталось только научится считать количество перестановок между группами. Это легко делается при помощи леммы 1 и замены каждого числа на номер отрезка в котором он лежит($b = O(T/n)$).

~~Лемма~~ Теорема

Теорема На вход даны n натуральных чисел принадлежащие $[U]$, а также параметр b и константа ϵ. Существует алгоритм подсчёта количества инверсий за время $O(n \lg_b^2 n + (n/b)\text{RAM-INV}(b) \lg_b n + \text{RAM-INV}(n) + \text{RAM-SORT}(n))$ и используя $O(b^{O(1)} \lg n + \min(n^{O(\epsilon)}, lg^{O(\epsilon)} U)$ дополнительных слов в $\mathrm{Restore}$ модели. **Доказательство** Шаги 1 и 3 не будут ничем отличаться от шагов в теореме 5.

$\mathrm{Шаг}\ 2'$ Давайте воспользуемся леммой 3, где K будет массивом составленным из первых $\lg b$ битов каждого числа Используя алгоритм 1, легко подсчитать количество инверсий между элементами с разными ведущими $\lg b$ битами. Далее применяем лемму 3 и сводим задачу к подотрезкам с одинаковыми ведущими битами если он довольно маленький. Либо сжимаем его и используем алгоритм из леммы 4.