САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кручинин Евгений Владимирович

Отчет по индивидуальному заданию

учебной дисциплины

Алгоритмы и анализ сложности

LSD сортировка

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа СВ.5003.2022 «Программирование и информационные технологии»

Преподаватель,
кандидат физ.-мат. наук,
доцент Никифоров К. А.

Санкт-Петербург

2024

Содержание

1. Введение и постановка задачи

1.1 Краткое описание

LSD (Least Significant Digit) сортировка, или поразрядная сортировка по младшим разрядам, представляет собой эффективный не сравнивающий алгоритм сортировки, применяемый для сортировки чисел или строк фиксированной длины. Алгоритм работает, сортируя элементы по разрядам, начиная с наименее значимого (младшего) разряда и заканчивая наиболее значимым (старшим).

1.2 Постановка задачи

Дан массив A из n целых неотрицательных чисел или строк фиксированной длины. Требуется отсортировать массив в порядке неубывания элементов.

1.3 Идея алгоритма

Идея алгоритма заключается в том, чтобы выполнить сортировку элементов по каждому разряду, начиная с младшего. Для этого на каждом шаге используется стабильный алгоритм сортировки, такой как сортировка подсчетом. Благодаря этому после завершения сортировки по всем разрядам элементы будут полностью отсортированы.

1.4 Алгоритм

1.4.1 Рекомендованные структуры данных

Массив A[1...n] — для хранения исходных данных, массив B[1...n] — для промежуточного хранения отсортированных элементов, массив C[0...k-1] — для подсчета частот (где k — количество возможных значений разряда).

1.4.2 Описание алгоритма LSD сортировки

Алгоритм можно описать следующим образом:

1. Определить число разрядов m и основание k. Каждый элемент массива рассматривается как число из m разрядов в системе с основанием k.
2. Для разряда от младшего к старшему (от 0-го до m-1-го):
   - Инициализировать счётчики C[0…k-1] нулями.
   - Пройти по массиву A и для каждого элемента определить цифру в текущем разряде; увеличить соответствующий C[d].
   - Преобразовать массив C в массив префиксных сумм. Это позволит определить, в какой сегмент массива B будут помещены элементы с каждой цифрой.
   - Пройти по массиву A с конца к началу: для каждого элемента найти позицию в B, используя префиксные суммы C, и поместить элемент в B.
   - Скопировать массив B обратно в A.
3. После обработки всех m разрядов массив A будет отсортирован.

1.4.3 Априорный анализ LSD сортировки

Обозначения

1. n — количество элементов в массиве A.
2. k — количество возможных значений цифры (основание системы счисления).
3. m — количество разрядов максимального числа (длина ключа).

На каждом разряде алгоритм выполняет следующие шаги:

1. Инициализация массива счетчиков C[0...k-1]:
   4. Количество операций: O(k).
2. Подсчет количества элементов для каждого значения:
   1. Проход по массиву A[1...n].

2. Количество операций: O(n).

3. Преобразование счетчиков в префиксные суммы:
    1. Количество операций: O(k).
4. Сборка отсортированного массива:
    1. Проход по массиву A[1...n] (с конца к началу).
    2. Количество операций: O(n).
5. Копирование массива B обратно в A:
    1. Количество операций: O(n).

Итого время работы на одном разряде

$$T_{digit}(n) = O(k) + O(n) + O(k) + O(n) + O(n) = O(n + k)$$

Общее время работы алгоритма

Алгоритм выполняет m таких итераций (по количеству разрядов), поэтому общее время работы составляет:

$$T_{total}(n) = m \times T_{digit}(n) = m \times O(n + k) = O(m(n + k))$$

1.5 Оценка затрачиваемой памяти

Для массива A и B требуются O(n) памяти, для C — O(k). Итого O(n + k). При фиксированных m и k памяти O(n), так как m не влияет напрямую на объём памяти, а k фиксированно.

1.6 Реализация алгоритма LSD-сортировки

Реализация алгоритма представлена на рисунке 1.

```python
def lsd_radix_sort(A, m, k):
    def get_digit(x, j):
        return (x // (k ** j)) % k

    n = len(A)
    B = [0] * n
    for digit_pos in range(m):
        C = [0] * k
        # Подсчёт частот
        for x in A:
            d = get_digit(x, digit_pos)
            C[d] += 1
        # Префиксные суммы
        for i in range(1, k):
            C[i] += C[i-1]
        # Раскладываем элементы
        for i in range(n-1, -1, -1):
            d = get_digit(A[i], digit_pos)
            C[d] -= 1
            B[C[d]] = A[i]
        # Копирование обратно
        A[:] = B[:]
    return A
```

Рисунок 1.

2. Описание входных данных и генератора

2.1 Входные данные

Массив из n неотрицательных целых чисел. Параметры m и k выбираются так, чтобы все числа умещались в m-разрядном числе в системе с основанием k.

2.2 Генератор данных

Генерируется случайный массив длины n, элементы от 0 до k^m - 1.

3. Эмпирический анализ

3.1 Вычислительный эксперимент и анализ результата

Проведём эксперименты для разных сценариев, замерив реальное время сортировки и сравнив его с теоретическими отношениями.

Параметры сценариев:

- k_fixed: фиксируем k и варьируем n и m. Ожидаем O(mn).
- both_fixed: фиксируем m и k, варьируем n. Ожидаем O(n).
- general: меняем все параметры, ожидая O(m(n+k)).

Ниже приведён пример полученных результатов.

3.1.1 Сценарий k_fixed

Результаты тестов при фиксирование k

(k, m, n)	Реальное	Теоретическое
(k=3, m=2, n=1000)	real=0.000759	theor=2000.000000
(k=3, m=2, n=2000)	real=0.001540	theor=4000.000000
(k=3, m=4, n=1000)	real=0.001404	theor=4000.000000
(k=3, m=4, n=2000)	real=0.002745	theor=8000.000000

Сравнение

Изменяющийся параметр	Реальное отношение	Теоретическое отношение	ratio_real/ ration_theor
(k=3, n=1000) m: 2 -> 4	1.8494	2.0000	0.9247

Изменяющийся параметр	Реальное отношение	Теоретическое отношение	ratio_real/ ration_theor
(k=3, n=2000) m: 2 -> 4	1.7821	2.0000	0.8911
(k=3, m=2) n: 1000 -> 2000	2.0284	2.0000	1.0142
(k=3, m=4) n: 1000 -> 2000	1.9546	2.0000	0.9773

Получается, зафиксировав k=3, мы получили, что при увеличение m в два раза реальное отношение будет около двух, как и наше теоретическое предположение. Если мы будем увеличивать n в два раза, то реальное отношение будет близко к двум, также как и теоретическое. Мы доказали O(mn).

3.1.2 Сценарий both_fixed

Результаты тестов при фиксирование m, k

(k, m, n)	Реальное	Теоретическое
(k=3, m=2, n=1000)	real=0.000625	theor=1000.000000
(k=3, m=2, n=2000)	real=0.001272	theor=2000.000000
(k=3, m=2, n=4000)	real=0.002428	theor=4000.000000
(k=3, m=4, n=1000)	real=0.001163	theor=1000.000000
(k=3, m=4, n=2000)	real=0.002668	theor=2000.000000
(k=3, m=4, n=4000)	real=0.005339	theor=4000.000000
(k=6, m=2, n=1000)	real=0.000602	theor=1000.000000
(k=6, m=2, n=2000)	real=0.001209	theor=2000.000000
(k=6, m=2, n=4000)	real=0.002425	theor=4000.000000
(k=6, m=4, n=1000)	real=0.001241	theor=1000.000000
(k=6, m=4, n=2000)	real=0.002625	theor=2000.000000
(k=6, m=4, n=4000)	real=0.005399	theor=4000.00000

Сравнение

Изменяющийся параметр	Реальное отношение	Теоретическое отношение	ratio_real/ ration_theor
(k,m)=(3, 2) n: 1000->2000	2.0549	2.0000	1.0274
(k,m)=(3, 2) n: 2000->4000	2.0299	2.0000	1.0149
(k,m)=(3, 4) n: 1000->2000	2.2936	2.0000	1.1468
(k,m)=(3, 4) n: 2000->4000	2.0011	2.0000	1.0006
(k,m)=(6, 2) n: 1000->2000	2.0073	2.0000	1.0037
(k,m)=(6, 2) n: 2000->4000	2.0051	2.0000	1.0025
(k,m)=(6, 4) n: 1000->2000	2.1159	2.0000	1.0580
(k,m)=(6, 4) n: 2000->4000	2.0566	2.0000	1.0283

Зафиксировав оба параметра m и k, мы получили реальное отношение на всех тестах 2 с отклонением максимум на +0.3. Это совпадает с нашим теоретическим предположением равным 2 и подтверждает O(n).

3.1.3 Сценарий general

Результаты тестов, когда мы не фиксируем ни одного параметра. В данном случае меняются сразу m, n, k

(k, m, n)	Реальное	Теоретическое
(k=2, m=1, n=1000)	real=0.000304	theor=1002.000000
(k=2, m=1, n=2000)	real=0.000603	theor=2002.000000
(k=2, m=1, n=4000)	real=0.001267	theor=4002.000000
(k=2, m=2, n=1000)	real=0.000609	theor=2004.000000

(k, m, n)	Реальное	Теоретическое
(k=2, m=2, n=2000)	real=0.001248	theor=4004.000000
(k=2, m=2, n=4000)	real=0.002756	theor=8004.000000
(k=2, m=3, n=1000)	real=0.000970	theor=3006.000000
(k=2, m=3, n=2000)	real=0.001952	theor=6006.000000
(k=2, m=3, n=4000)	real=0.003893	theor=12006.000000
(k=4, m=1, n=1000)	real=0.000293	theor=1004.000000
(k=4, m=1, n=2000)	real=0.000599	theor=2004.000000
(k=4, m=1, n=4000)	real=0.001224	theor=4004.000000
(k=4, m=2, n=1000)	real=0.000564	theor=2008.000000
(k=4, m=2, n=2000)	real=0.001176	theor=4008.000000
(k=4, m=2, n=4000)	real=0.002429	theor=8008.000000
(k=4, m=3, n=1000)	real=0.000850	theor=3012.000000
(k=4, m=3, n=2000)	real=0.001774	theor=6012.000000
(k=4, m=3, n=4000)	real=0.003748	theor=12012.000000
(k=8, m=1, n=1000)	real=0.000330	theor=1008.000000
(k=8, m=1, n=2000)	real=0.000647	theor=2008.000000
(k=8, m=1, n=4000)	real=0.001277	theor=4008.000000
(k=8, m=2, n=1000)	real=0.000611	theor=2016.000000
(k=8, m=2, n=2000)	real=0.001202	theor=4016.000000
(k=8, m=2, n=4000)	real=0.002447	theor=8016.000000
(k=8, m=3, n=1000)	real=0.000955	theor=3024.000000
(k=8, m=3, n=2000)	real=0.001959	theor=6024.000000
(k=8, m=3, n=4000)	real=0.004523	theor=12024.000000

Сравнение

Две точки	Реальное отношение	Теоретическое отношение	ratio_real/ ration_theor
(k=2, m=1, n=1000) (k=2, m=1, n=2000)	1.9819	1.9980	0.9919
(k=2, m=1, n=1000) (k=2, m=1, n=4000)	4.1641	3.9940	1.0426

Две точки	Реальное отношение	Теоретическое отношение	ratio_real/ ration_theor
(k=2, m=1, n=1000) (k=2, m=3, n=4000)	12.7979	11.9820	1.0681

Меняя сразу несколько параметров, полученное реальное отношение соотносится с теоретическим отношением из расчета O(m(n+k)).

4. Заключение

Анализ показывает нам, что алгоритм особо эффективно решает задачу при фиксированных m и k. Эксперименты подтвердили оценку сложности в O(m(n+k)) и лучшую оценку O(n).

5. Список литературы и источники

1. Левитин А. «Алгоритмы: введение в разработку и анализ»
2. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. «Introduction to Algorithms»
3. Sedgewick R., Wayne K. «Algorithms»
4. Дополнительные материалы по Radix Sort и Counting Sort из открытых источников

6. Характеристики вычислительной среды и оборудования

Вычислительная среда: интерпретатор Python

Оборудование:

Процессор: 2,6 GHz 6-ядерный процессор Intel Core i7

ОЗУ: 16,0 ГБ