МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АДЫГЕЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Инженерно-физический факультет
Кафедра управления и информатики в технических системах

Отчет по практике

Вариант 4. Сортировки Быстрая и Слиянием.

2 курс, группа УТС

Выполнил:
_____ К. В. Швыдко
«___» _____ 2023 г.

Руководитель:
_____ С. В. Теплоухов
«___» _____ 2023 г.

Майкоп, 2023 г.

1. Введение

Quick Sort

Алгоритм относится к алгоритмам «разделяй и властвуй». Его используют чаще других алгоритмов, описанных в этой статье. При правильной конфигурации он чрезвычайно эффективен и не требует дополнительной памяти, в отличие от сортировки слиянием. Массив разделяется на две части по разные стороны от опорного элемента. В процессе сортировки элементы меньше опорного помещаются перед ним, а равные или большие — позади.

Merge Sort

Алгоритм также относится к алгоритмам «разделяй и властвуй». Он разбивает список на две части, каждую из них он разбивает ещё на две и т. д. Список разбивается пополам, пока не останутся единичные элементы. Соседние элементы становятся отсортированными парами. Затем эти пары объединяются и сортируются с другими парами. Этот процесс продолжается до тех пор, пока не отсортируются все элементы.

1.1. Текстовая формулировка задачи Quick Sort

Quick Sort

На входе массив a[0]...a[N] и опорный элемент p, по которому будет производиться разделение.

1. Введем два указателя: i и j. В начале алгоритма они указывают, соответственно, на левый и правый конец последовательности.

2. Будем двигать указатель i с шагом в 1 элемент по направлению к концу массива, пока не будет найден элемент a[i] >= p. Затем аналогичным образом начнем двигать указатель j от конца массива к началу, пока не будет найден a[j] <= p.

3. Далее, если i <= j, меняем a[i] и a[j] местами и продолжаем двигать i,j по тем же правилам.

4. Повторяем шаг 3, пока i <= j.

Реализация кода, решающего данную задачу

```python
def partition(nums, low, high):
    pivot = nums[(low + high) // 2]
    i = low - 1
    j = high + 1
    while True:
        i += 1
        while nums[i] < pivot:
            i += 1
        j -= 1
        while nums[j] > pivot:
            j -= 1
        if i >= j:
            return j
        nums[i], nums[j] = nums[j], nums[i]

def quick_sort(nums):

    def _quick_sort(items, low, high):
        if low < high:
            split_index = partition(items, low, high)
            _quick_sort(items, low, split_index)
            _quick_sort(items, split_index + 1, high)

    _quick_sort(nums, 0, len(nums) - 1)

random_list_of_nums = [22, 5, 1, 18, 99]
quick_sort(random_list_of_nums)
print(random_list_of_nums)
```

Принцип работы:

1.

Выбирается опорный элемент (рандомно, либо последний/первый элемент в массиве, либо среднее арифметическое, если массив состоит из чисел)

2.

Массив разбивается на два подмассива, в одном элементы не больше опорного, в другом не меньше опорного

3.

Полученные подмассивы сортируются рекурсивно вызовом процедуры быстрой сортировки

ОЦЕНКА ВРЕМЕНИ РАБОТЫ

Лучший случай:

Предположим, что опорный элемент выбирается так, что при каждом разбиении, массив разделяется на две примерно одинаковых части тогда имеем:

$$T = O(n) + O(\frac{n}{2} + \frac{n}{2}) + O(\frac{n}{4} + ... + \frac{n}{4}) + ... + O(\frac{n}{2^k} + ... + \frac{n}{2^k}) =$$
$$= \left[2^k = n;\ k = log_2 n\right] = O(n) + O(n) + O(n) + ... + O(n) =$$
$$= O(nk) = O(n\ log_2 n) = O(n\ log\ n)$$

Худший случай:

Предположим, что опорный элемент выбирается так, что при каждом разбиении, массив разделяется на две части: в одной - 1 элемент, в другой - n-1, тогда имеем:

$$T = O(n) + O(n - 1) + O(n - 2) + ... + O(1) = O(n^2)$$

Средний случай:

В среднем время сортировки слиянием составляет O(n log n).

При любом фиксированном соотношении между левой и правой частями разделения изменится глубина рекурсии, а следовательно и основание логарифма, которое можно не учитывать (при помощи свойств логарифма, его можно привести к любому основанию), а следовательно сложность алгоритма будет той же, только с разными константами.

Оценка используемой памяти

M=O(1) - в теории, но рекурсивная реализация M=O(n), так как в худшем случае понадобится сделать O(n) вложенных рекурсивных вызовов.

Сортировка нестабильна: все зависит от конкретных данных. Она может работать как быстро, так и долго.

Преимущества быстрой сортировки:

Алгоритм эффективен на больших наборах данных.

Алгоритм имеет низкие накладные расходы, поскольку для ее функционирования требуется лишь небольшой объем памяти.

Недостатки быстрой сортировки:

Это не лучший выбор для небольших наборов данных.

Это нестабильная сортировка, означающая, что если два элемента имеют одинаковый ключ, их относительный порядок не будет сохранен в отсортированных выходных данных в случае быстрой сортировки, потому что здесь мы меняем местами элементы в соответствии с положением оси (без учета их исходных положений).

1.2. Текстовая формулировка задачи Merge Sort

Функция merge на месте двух упорядоченных массивов a[lb]...a[split] и a[split+1]...a[ub] создает единый упорядоченный массив a[lb]...a[ub].

Рекурсивный алгоритм обходит получившееся дерево слияния в прямом порядке. Каждый уровень представляет собой проход сортировки слияния - операцию, полностью переписывающую массив.

Обратим внимание, что деление происходит до массива из единственного элемента. Такой массив можно считать упорядоченным, а значит, задача сводится к написанию функции слияния merge.

Один из способов состоит в слиянии двух упорядоченных последовательностей при помощи вспомогательного буфера, равного по размеру общему количеству имеющихся в них элементов. Элементы последовательностей будут перемещаться в этот буфер по одному за шаг.

Реализация кода, решающего данную задачу

```python
def merge_sort(sequence):

    if len(sequence) < 2:
        return sequence
    mid = len(sequence) // 2

    left_sequence = merge_sort(sequence[:mid])
    right_sequence = merge_sort(sequence[mid:])

    return merge(left_sequence, right_sequence)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]

    return result

# Print the sorted list.
print(merge_sort([5, 2, 6, 8, 5, 8, 1]))
```

Принцип работы:

Делим массив пополам, каждый из них сортируем слиянием и потом соединяем оба массива. Каждый разделённый массив тоже нарезаем на два подмассива до тех пор, пока в каждом не окажется по одному элементу.

Здесь тоже используется рекурсия — то есть повторение алгоритма внутри самого алгоритма. Но это только один из элементов алгоритма.

Второй элемент — соединение отсортированных элементов между собой, причём тоже хитрым способом: раз оба массива уже отсортированы, то нам достаточно сравнивать элементы друг с другом по очереди и заносить в итоговый массив данные по порядку.

Оценка используемой памяти

$$\mathbf{T = O(n) + O(n/2 + n/2) + O(n/4 + \ldots + n/4) + \ldots + O(n/2\text{^}k + .. + n/2\text{^}k) =}$$

$$\mathbf{= O(nk) = O(n \log n)}$$

В среднем время сортировки слиянием составляет O(n log n).

Области применимости:

Применяется при использовании большого упорядоченного файла данных, в который регулярно необходимо добавлять новые элементы.

Преимущества сортировки слиянием:

Сортировка слиянием - это стабильный алгоритм сортировки, который означает, что он поддерживает относительный порядок равных элементов во входном массиве

Временная сложность сортировки слиянием в наихудшем случае равна O (N logN), что означает, что она хорошо работает даже на больших наборах данных.

Недостатки сортировки слиянием:

Сортировка слиянием требует дополнительной памяти для хранения объединенных подмассивов в процессе сортировки.

Для хранения отсортированных данных требуется дополнительная память. Это может быть недостатком в приложениях, где беспокоит использование памяти.

2. Список использованных источников

Список литературы

[1] Вирт Н. Алгоритмы и структуры данных; 2010г.

[2] Технология программирования. Методы сортировки данных 2017г.

[3] сортировка и системы сортировки. Лорин Г.

[4] Дональд Э. Кнут Искусство программирования. Том 3. Сортировка и поиск;

[5] URL: https://thecode.media/merge-sort/

[6] URL:https://www.geeksforgeeks.org/merge-sort/