МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

СОРТИРОВКА С ПОМОЩЬЮ РАЗДЕЛЕНИЯ

ЛАБОРАТОРНАЯ РАБОТА

студента 3 курса 331 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Никитина Арсения Владимировича

Проверил
доцент, к.т.о.к.б.и.к _____ А. Н. Гамова

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной работе будут рассмотрены принципы работы сортировки с помощью разделения, также известной под названием QuickSort, оценки работы сортировки в наилучшем и наихудшем случаях, а также ее программная реализация. Сортировку с помощью разделения также называют быстрой сортировкой, так как она является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы.

1 Принципы работы сортировки разделением

Для достижения наибольшей эффективности сначала лучше производить перестановки на большие расстояния. Предположим, что у нас есть n элементов, расположенных в обратно отсортированном порядке. Их можно отсортировать за $n/2$ операций, если заранее знать, что порядок именно таков.

Алгоритм функции $partition$

Итак, выберем наугад любой элемент входного массива данных и назовем его x. Затем будем просматривать слева направа весь массив до тех пор, пока не обнаружим первый элемент, больший выбранного. Далее будем просматривать весь массив справа, пока не встретим первый элемент, меньший выбранного. Затем поменяем местами найденные элементы и продолжим процесс просмотра и обмена, пока оба просмотра не встретятся в середине массива. В результате проделанных операций, массив окажется разбитым на две части: с ключами, меньшими или равными заданному числу и с ключами, большими этого числа. В программной реализации же, включение одной из частей является не строгим и выбранный элемент x является барьером между частями.

Алгоритм функции $sort$

Далее нужно применить описанный выше алгоритм к получившимся двум частям, затем к четырем и так далее. Поэтому требуется рекурсивный вызов некоторой функции $sort$.

Также у обычного метода сортировки выбором присутствует множество оптимизаций. Например, оптимизация выбора разделяющего. В данной реализации происходит выбор медианного элемента из массива, поэтому скорость работы алгоритма в среднем улучшается.

Также метод можно реализовать итеративно, но для этого требуются лишние затраты памяти.

2 Программная реализация алгоритма

```
def partition(array, start, end):
    pivot = array[start]
    low = start
    high = end

    while True:

        while low <= high and array[high] >= pivot:
            high = high - 1

        while low <= high and array[low] <= pivot:
            low = low + 1

        if low <= high:
            array[low], array[high] = array[high], array[low]
        else:
            break

    array[start], array[high] = array[high], array[start]

    return high


def quick_sort(array, start, end):

    if start >= end:
        return

    pivot = partition(array, start, end)

    quick_sort(array, start, pivot - 1)
    quick_sort(array, pivot + 1, end)


array = [29, 99, 27, 41, 66, 28, 44, 78, 87, 19, 31, 76, 58, 88, 83, 97, 12,
    ↪ 21, 44]

quick_sort(array, 0, len(array) - 1)
print(*array)
```

3 Оценка работы алгоритма

Для исследования производительности быстрой сортировки сначала необходимо разобраться, как идет процесс разделения. Выбрав некоторое граничное значение x, происходит обход по всему массиву, а, значит, при этом выполняется точно n сравнений. Число же обменов можно определить из следующих вероятностных соображений.

При заданной границе значений x ожидаемое число операций обмена равно числу элементов в левой части разделяемой последовательности, то есть $n-1$, умноженному на вероятность того, что при обмене каждый такой элемент перед этим находился в правой части. Вероятность этого равна $(n-(x-1))/n$. Поэтому ожидаемое число обменов есть среднее этих ожидаемых значений для возможных границ, то есть получаем: $((n-1)/n)/6$.

Пусть всегда удается выбрать в качестве границы медиану, в этом случае каждый процесс разделений ресщеплет массив на две половины и для сортировки требуется всего $\log n$ проходов. В результате общее число сравнений равно $n * \log n$, а общее число обменов - $n * \log(n)/6$.

Если же выбор разделяющего элемента не всегда идеален, то сложность алгоритма отличается на коэффициент $2 * \ln 2$.

Также стоит сказать, что алгоритм имеет преимущетсво перед другими усовершенствованными, что для обработки небольших частей массива в него можно включить какой-либо из методом прямой сортировки.

Наихудший же случай работы алгоритма достигается, когда для сравнения выбирается наибольший элемент из значений, сосоящих в части. Тогда на каждом этапе сегмент из n элементов будет расщепляться на левую часть, состоящую из $n-1$ элементов, и на правую часть, состоящую из одного элемента (выбранного). Тогда в результате требуется n разделений и наихудшая производительность метода будет порядка n^2.

Исследования Сама Хоара показывают, что наилучшая производительность алгоритма достигается при случайном выборе разделяющего элемента.

ЗАКЛЮЧЕНИЕ

Итак, в данной работе был рассмотрен алгоритм сортировки с помощью разделения. Произведена оценка его работы в худшем и в лучшем случае. В худшем случае ассимптотика работы алгоритма составляет $O(n^2)$ операций, в лучшем - $O(n * \log n)$.

Наиболее оптимальмальная работа алгоритма достигается при случайном выборе разделяющего элемента: в таком случае вероятность того, что за один проход алгоритма будет совершено операций обмена менее $\log n$ при $\lim\limits_{n \to \infty}$ стремится к нулю.