Санкт-Петербургский Государственный Университет

Факультет Прикладной Математики и Процессов Управления

Задание по эмпирическому анализу алгоритма по курсу

«Алгоритмы и анализ сложности»

Алгоритм LSD-поразрядной сортировки

Преподаватель: Никифоров К. А.

Выполнил: студент группы 16.Б13-пу Кириченко В. В.

Санкт-Петербург, 2018

Содержание

1 Описание и области применения алгоритма 3

2 Математический анализ алгоритма 3

3 Характеристики входных данных и способ измерения трудоёмкости 3

4 Реализация алгоритма 3

5 Генерация входных данных 4

6 Вычислительный эксперимент в исследуемом диапазоне входных данных 5

7 Анализ полученных результатов 5

8 Список источников 8

9 Характеристики использованной вычислительной среды и оборудования 9

1 Описание и области применения алгоритма

Алгоритм LSD-поразрядной сортировки является одним из вариантов алгоритма поразрядной сортировки. В нём числа сортируются по разрядам, причём сначала сортируются младшие разряды, а затем старшие. Данный алгоритм применялся для сортировки перфокарт. Алгоритм применялся в работе Германа Холлерита над табуляторами в 1887 году. Компьютерная реализация алгоритма была разработана в 1954 году Гарольдом Сьюардом.

2 Математический анализ алгоритма

В ходе работы алгоритму необходимо сортировать массив элементов для каждого разряда. Для этого числа распределяются по корзинам в соответствии с рассматриваемым разрядом, а затем корзины склеиваются. В таком случае основной операцией в ходе работы алгоритма является помещение элемента в корзину (список или массив). Таким образом, сложность алгоритма составляет $O(wn)$, где n — число элементов в списке, а w — количество разрядов.

3 Характеристики входных данных и способ измерения трудоёмкости

Входными данными для алгоритма является массив чисел. Трудоёмкость алгоритма измерялась в миллисекундах, затраченных на сортировку массива алгоритмом. Измерение времени производилось с помощью встроенного в стандартную библиотеку языка модуля time. Массивы генерировались с различным количеством как чисел, так и разрядов в каждом числе. Для упрощения интерпретации результатов измерения трудоёмкости в массиве присутствовали числа только с одинаковым количеством разрядов. Было выбрано следующее количество чисел в массиве: 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200; количество разрядов в каждом числе: 1, 2, 4, 8.

4 Реализация алгоритма

Алгоритм LSD-поразрядной сортировки был реализован на языке Python версии 3.7.1. Массив чисел представлен в виде списка. Для сортировки чисел по разрядам используются дополнительные списки. Их количество равно основанию системы счисления рассматриваемых чисел.

```
1 from math import log
2
3
4 def get_digit(num, base, digit_num):
5     # pulls the selected digit
6     return (num // base ** digit_num) % base
7
8
9 def make_blanks(size):
```

```
    # create a list of empty lists to hold the split by digit
    return [[] for i in range(size)]


def split(a_list, base, digit_num):
    buckets = make_blanks(base)
    for num in a_list:
        # append the number to the list selected by the digit
        buckets[get_digit(num, base, digit_num)].append(num)
    return buckets


# concatenate the lists back in order for the next step
def merge(a_list):
    new_list = []
    for sublist in a_list:
        new_list.extend(sublist)
    return new_list


def max_abs(a_list):
    # largest abs value element of a list
    return max(abs(num) for num in a_list)


def split_by_sign(a_list):
    # splits values by sign - negative values go to the first bucket,
    # non-negative ones into the second
    buckets = [[], []]
    for num in a_list:
        if num < 0:
            buckets[0].append(num)
        else:
            buckets[1].append(num)
    return buckets


def radix_sort(a_list, base):
    # there are as many passes as there are digits in the longest number
    passes = int(round(log(max_abs(a_list), base)) + 1)
    new_list = list(a_list)
    for digit_num in range(passes):
        new_list = merge(split(new_list, base, digit_num))
    return merge(split_by_sign(new_list))
```

5 Генерация входных данных

Генерация массивов чисел осуществляется на основе двух параметров — количества элементов в результирующем массиве и количества разрядов чисел (задаётся в виде минимального числа). Для генерации псевдослучайных равномерно распределённых чисел используется встроенный в стандартную библиотеку модуль random, реализованный на основе Вихря Мерсенна. Благодаря этому ГПСЧ лишён многих недостатков, присущих другим ГПСЧ, таких как малый период, предсказуемость и легко выявляемые статистические закономерности.

```
def generate(length, amount):
    return [random.randint(length, 10 * length - 1) for _ in range(amount)]
```

6 Вычислительный эксперимент в исследуемом диапазоне входных данных

С помощью описанного выше генератора данных было произведено тестирование алгоритма со всеми возможными парами параметров, указанных в разделе 3. На каждой такой паре алгоритм запускался 2000 раз с различными сгенерированными данными, чтобы минимизировать влияние как фоновых процессов, так и особенностей случайных данных. Результаты таких тестовых прогонов усреднялись и записывались в качестве результата.

```
def timer(length, amount, number=2000):
    times = []
    for _ in range(number):
        to_sort = generate(length, amount)
        start = time.time()
        algorithm.radix_sort(to_sort, 10)
        stop = time.time()
        times.append(stop - start)
    return sum(times) / number


lengths = [1, 10, 1000, 10000000]
amounts = [100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200]

def test():
    tested = list()
    for length in lengths:
        for amount in amounts:
            ex_time = timer(length, amount)
            tested.append({'from': length,
                           'to': 10 * length,
                           'amount': amount,
                           'time': ex_time * 1000})
    return tested
```

7 Анализ полученных результатов

В таблицах 1–4 представлены результаты, полученные в ходе вычислительного эксперимента.

Количество элементов	Время работы, мс
100	0.0668172836303711
200	0.14360392093658447
400	0.28176605701446533
800	0.555321455001831
1600	1.2162401676177979
3200	2.363770008087158
6400	4.941829562187195
12800	9.466262340545654
25600	18.687508702278137
51200	37.44811272621155

Таблица 1. Массивы чисел с 1 разрядом.

Количество элементов	Время работы, мс
100	0.09697568416595459
200	0.22541213035583496
400	0.4338417053222656
800	0.9395959377288818
1600	1.827741265296936
3200	3.6263442039489746
6400	6.917779326438904
12800	14.10499358177185
25600	28.16263520717621
51200	56.175602436065674

Таблица 2. Массивы чисел с 2 разрядами.

Количество элементов	Время работы, мс
100	0.19944369792938232
200	0.3894462585449219
400	0.7644665241241455
800	1.5598496198654175
1600	3.0463205575942993
3200	6.123116374015808
6400	12.208956241607666
12800	24.4019957780838
25600	48.945778012275696
51200	98.92793309688568

Таблица 3. Массивы чисел с 4 разрядами.

Количество элементов	Время работы, мс
100	0.3834867477416992
200	0.7435144186019897
400	1.435721755027771
800	2.9485440254211426
1600	5.750182867050171
3200	11.492654204368591
6400	23.274991035461426
12800	46.117165088653564
25600	92.64567613601685
51200	187.23099100589752

Таблица 4. Массивы чисел с 8 разрядами.

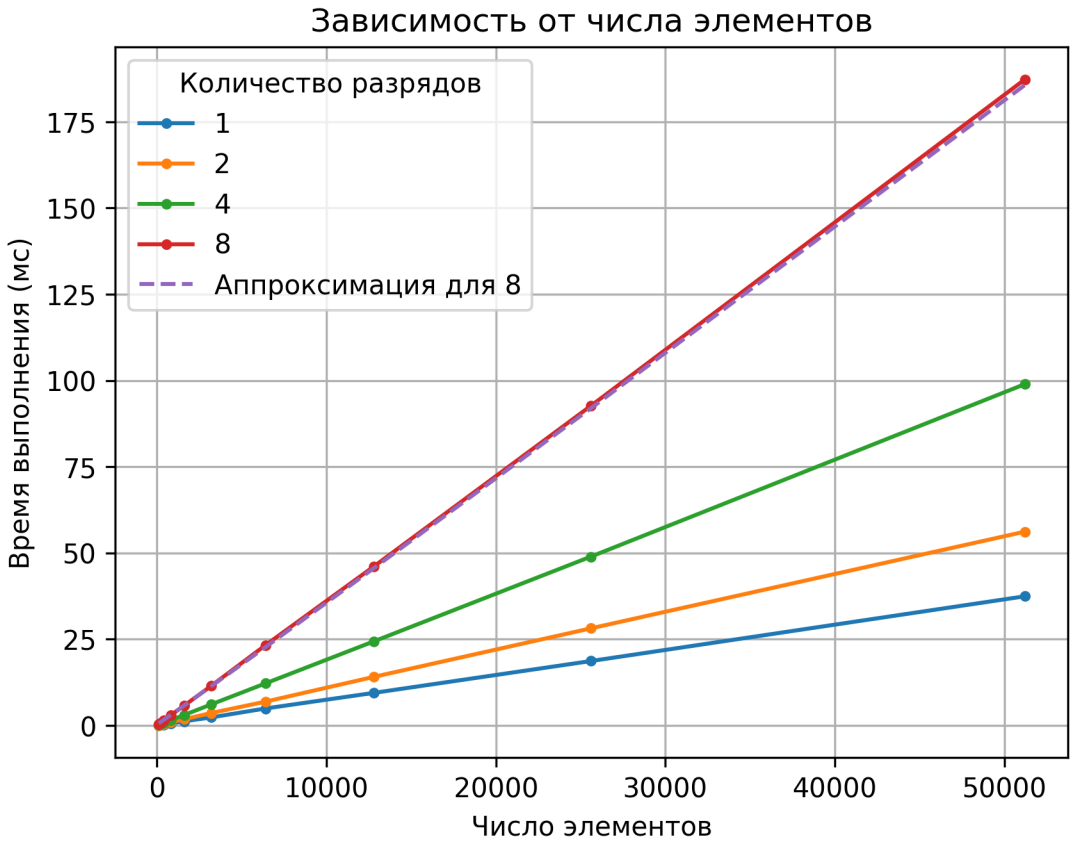

Рис. 1: зависимость времени выполнения от числа элементов

Найдём аппроксимацию функции трудоёмкости алгоритма $aw^b n^c + d$ при помощи функции FindFit из пакета Wolfram Mathematica: $a = 0.000519248, b = 0.851639, c = 1.01574, d = 0.30329$. На рисунках 1 и 2 можно видеть, что найденная аппроксимация близка к полученным данным. Можно сделать вывод, что количество разрядов в числах влияют не столь существенно по сравнению с количеством элементов в массиве. Тем не менее, полученная оценка трудоёмкости близка к оценке, полученной в ходе математического анализа алгоритма.

Проанализируем соотношение $\frac{T(2n)}{T(n)}$ для массива чисел с 8 разрядами:

Количество элементов n	$\frac{T(2n)}{T(n)}$
100	1.93882689
200	1.93099383
400	2.053701572
800	1.950177042
1600	1.998658907
3200	2.025205894
6400	1.981404204
12800	2.008919585
25600	2.020936096

Во всех случаях соотношение $\frac{T(2n)}{T(n)}$ соответствует примерно 2. Это означает, что увеличение количества чисел в массиве линейно влияет на трудоёмкость алгорит-

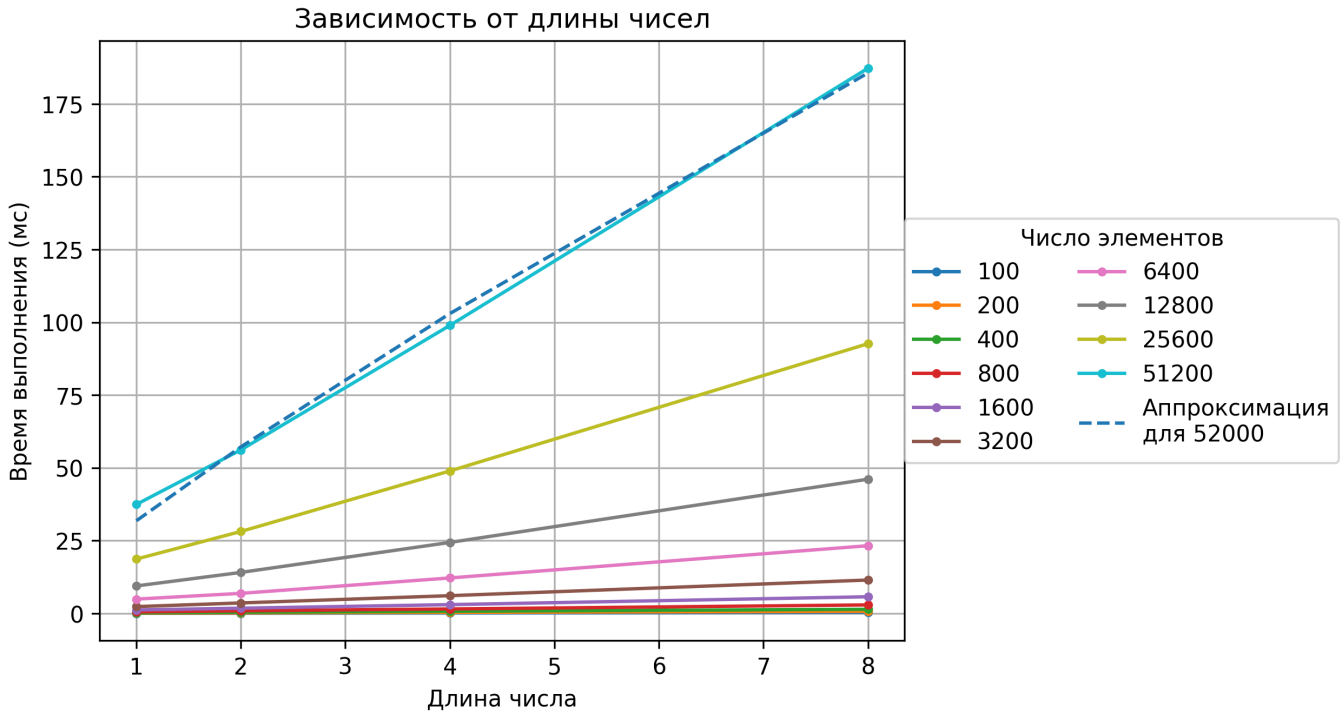

Рис. 2: зависимость времени выполнения от количества разрядов

ма. Аналогично проанализируем влияние количества разрядов на примере массивов с 25600 элементов:

Количество разрядов w	$\frac{T(2w)}{T(w)}$
1	1.507029945
2	1.737968683
4	1.892822627

Соотношение $\frac{T(2w)}{T(w)}$ стремится к 2, поэтому можно сделать вывод, что количество разрядов чисел так же оказывает линейное влияние на трудоёмкость алгоритма.

8 Список источников

1. [Radix Sort — Википедия];
2. [Поразрядная сортировка — Manual.ru];
3. Д. Э. Кнут. Искусство программирования, том 3. Сортировка и поиск — 2-е изд. — М.: И.Д.Вильямс, 2007. — С. 192–196;
4. [Модуль random в Python 3]
5. [Модуль time в Python 3]
6. [Реализация алгоритма на языке Python]
7. [Функция FindFit в Wolfram Mathematica]

9 Характеристики использованной вычислительной среды и оборудования

В качестве вычислительной среды использовался интерпретатор CPython 3.7.1, запущенный на персональном компьютере со следующими характеристиками:

- Процессор: Intel Core i7-8700K (6 ядер, 12 потоков, работал на тактовой частоте 4.35 ГГц, L1-кэш: 384 КиБ, L2-кэш: 1.5 МиБ, L3-кэш: 12 МиБ)
- Оперативная память: 16 ГиБ DDR3 3200 МГц в двухканальном режиме
- Графический ускоритель: NVIDIA GeForce GTX 1070 (8 ГиБ видеопамяти GDDR5)
- Операционная система: Microsoft Windows 10.