Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: Н. А. Рудаков
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата: 09.03.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №1

Задача: Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.

Вариант сортировки: Поразрядная сортировка.

Вариант ключа: `MD5-суммы (32-разрядные шестнадцатиричные числа).`

Вариант значения: `Строки переменной длины (до 2048 символов).`

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки.

Идея поразрядной сортировки заключается в том, чтобы применить сортировку подсчетом для каждого разряда. В свою очередь, идея сортировки подсчетом заключается в том, чтобы для каждого входного элемента x определить количество элементов, которые меньше x [1].

2 Описание работы программы и таблица с реализованными функциями

1. Входные данные представляют собой пары вида: ***«ключ-значение»***. Для того, чтобы их отсортировать, реализуем структуру хранения ключей ***memorizedKey***, для меньших затрат по памяти. Поделим ключи на две части, каждая из которых будет представлять собой 16-ти битное 16-ричное число. Затем переведем обе половины в 10-ные числа и положим их в unsign long long. Чтобы сделать перевод, реализуем функцию ***toTen***. В случае хранения строк в качестве ключа, мы бы тратили 32 байта, а так у нас расходуется 16 байт, так как каждый unsign long long хранит по 8 байт. Таким образом, мы оптимизировали хранение ключей ровно в 2 раза.
2. Теперь наша пара имеет следующий вид: ***«memorizedKey-значение»***. Мы изменим его следующим образом: создадим вектор со значениями, а ключу будет соответствовать индекс нужного значения в этом векторе. Это нужно для того, чтобы не пришлось копировать постоянно значения во время сортировки подсчетом. Очень большая экономия памяти. Теперь наши пары имеют следующий вид: ***«memorizedKey-index»***
3. Наконец, запускаем поразрядную сортировку. Она представляет собой сортировку подсчетом вызванную для каждых 2-х 16-ричных разрядов, то есть для 8-ми 2-чных. Это ускоряет работу программы. Сортировка подсчетом реализована через битовые сдвиги. Есть вектор размера 256 (16 байтов в половине ключа и 16 байтов в 16-ричной системе). Берутся два разряда, по ним мы находим индекс в векторе, в котором ведем подсчет. Делаем префикс данного вектора и, начиная с конца, составляем посорченый вектор по текущим двум разрядам. Затем приравниваем старый вектор к новому.
4. Осталось сделать вывод. Собирем две части нашего ключа при помощи функции ***fromTen***, выведем ключ, а затем по индексу в векторе выведем значение, соответствующее данному ключа.

Функции, которые были реализованы:

main.cpp	
void radixSort(std::vector<std::pair<memorizedKey, int>>, &arr)	Функция поразрядной сортировки.
void countSort(std::vector<std::pair<memorizedKey, int>> &arr, int &digit)	Функция сортировки подсчетом.

unsigned long long toTen(std::string &s)	Функция перевода строки в десятичную систему счисления из шестнадцатеричной.
std::string fromTen(unsigned long long &num)	Функция перевода числа из десятичной системы счисления в шестнадцатеричную.
memorizedKey toStructure(std::string &num)	Функция перевода ключа в структуру.
void print(std::vector<std::pair<memorizedKey, int>> &arr, std::vector<std::string> &values)	Функция вывода ответа.
int main()	Функция запуска программы.

Структура для хранения ключей.

```
1 struct memorizedKey {
2     unsigned long long left_half;
3     unsigned long long right_half;
4 }
```

3 Консоль

```
ruda@ruda-lp:~/CLionProjects/lab1/cmake-build-debug$ cat test
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
ruda@ruda-lp:~/CLionProjects/lab1/cmake-build-debug$ ./main <test
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
```

4 Тест производительности

Необходимо сравнить время работы встроенной сортировки std::sort с реализованной поразрядной сортировкой. Для этого напишем генератор тестов на python. Также, необходимо перегрузить оператор сравнения для структуры, чтобы мы могли использовать std::sort.

Код генератора:

```
import random
import string
import os

TEST_COUNT = 3
ALPHABET = '0123456789abcdef'


def get_random_item():
    key_size = 32
    value_size = random.randint(1, 2048)

    key = ""
    for _ in range(key_size):
        key += random.choice(ALPHABET)

    value = ""
    for _ in range(value_size):
        value += random.choice(string.ascii_letters + ALPHABET)

    return '\t'.join([key, value])


def make_test():
    arr = []
    for _ in range(random.randint(1, 8)):
        arr.append(get_random_item())

    return arr


def make_directory():
    try:
        os.mkdir('tests')
    except OSError:
        print('The directory already exists\n')


def make_all_tests():
    make_directory()
```

```
41      for number in range(TEST_COUNT):
42          test = open(f'tests/test{number + 1}', 'w+')
43          for item in make_test():
44              test.write(item + '\n')
45          test.close()
46
47
48  if __name__ == "__main__":
49      make_all_tests()
```

```
ruda@ruda-lp:~/CLionProjects/lab1 python3 test_generator.py
Tests were successfully created
ruda@ruda-lp:~/CLionProjects/lab1$ cd cmake-build-debug/
ruda@ruda-lp:~/CLionProjects/lab1/cmake-build-debug$ ./main <../tests/test1
Working time for radix sort is: 20115 nanoseconds
Working time for std::sort is: 17141 nanoseconds

ruda@ruda-lp:~/CLionProjects/lab1/cmake-build-debug$ ./main <../tests/test2
Working time for radix sort is: 20042 nanoseconds
Working time for std::sort is: 15262 nanoseconds

ruda@ruda-lp:~/CLionProjects/lab1/cmake-build-debug$ ./main <../tests/test3
Working time for radix sort is: 19252 nanoseconds
Working time for std::sort is: 13512 nanoseconds
```

Как видно, поразрядная сортировка работает медленнее, чем встроенная. Это связано с тем, что был сделан упор в оптимизацию памяти, для которой нам понадобилось реализовать перевод между системами счичления и перевод ключей в структуру.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать поразрядную сортировку и сортировку подсчетом; понял, как работают битовые сдвиги; улучшил свой скилл в работе с памятью; вспомнил, как пишутся структуры и перегрузил оператор.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008