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: Т. Б. Даутов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

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.

Основная идея поразрядной сортировки заключается в поочерёдной сортировке ключей по каждому разряду (цифре), начиная с младшего (LSD). Причём метод сортровки по разряду должен быть устойчивым. Для этого на каждом этапе последовательность сортируется методом подсчёта, где ключ - текущий разряд сортировки.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $TKeyValue$, в которой будем хранить ключ и значение.

Ключ, описываемый суммой MD5, записывается в 16-байтный массив типа $TByte$ (байтовых беззнаковых чисел). Строковое значение хранится в виде указателя на соответсвующий буфер, который аллоцируется в ходе программы функцией *malloc*.

Затем была описана вспомогательная функция $HexCharToByte$, которая переводит шестнадцатиричный символ в его байтовое представление. Функция $GetRadix$ предназначена для выделения разряда MD5. Работает полностью за счёт битовых операций.

Далее была разработана структура $TCollection$ и связанные с ней функции. Является реализацией динамически расширяемой структурой хранения данных типа $TKeyValue$.

В главной функции *main* описана общая логика работы программы. Сначала производится ввод данных из входного потока (*stdin*). Для чтения ключа используется временный буфер $read\_key\_buf$, в который считывается строковое представление ключа с помощью $scanf$. Затем, в цикле, эта строка обрататывается, с помощью битовых операций формируется уже численное (байтовое) представление MD5. Наконец, создаётся буфер для значения пары - строки, в который оно и записывается. В конце избыточная выделенная часть буфера строки "отрезается". Далее следует сортировка полученных пар поразрядным методом. Внешний цикл проходит по всем индексам разрядов MD5 (т.е. от 0 до 31) от младших к старшим. Внутри него создаётся целочисленный массив подсчёта *counter* размера, равного количеству цифр в 16-чной СС (т.е. 16). Далее происходит сортировка подсчётом для каждой пары исходных данных при конкретном индексе разряда (параметра внешнего цикла). Причём создаётся дополнительный массив размера исходного массива для записи в него каждого последующего результата. В конце итерации указатели на массивы меняются местами. Поскольку количество итераций внешнего цикла чётное, то итог сортировки будет записан в исходный массив. В конце программа печатает отсортированное содержимое и завершается.

main.c	
uint8_t GetRadix(TByte* key, uint8_t idx)	Функция получения разряда MD5
TByte HexCharToByte(char c)	Функция преобразования HEX знака в число

```
1  typedef uint8_t TByte;
2
3  typedef struct _tkeyvalue {
4    TByte key[HEX_DIGITS_AMOUNT];
5    char* value;
6  } TKeyValue;
7
8  typedef struct _tcollection {
9      uint32_t capacity;
10     uint32_t size;
11     TKeyValue* buffer;
12 } TCollection;
```

3 Консоль

```
timur@workstation$ gcc -pedantic -Wall -std=c11 main.c -o lab1
timur@workstation$ cat input.txt
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
timur@workstation$ ./lab1 <input.txt
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнивается реализованный аогоритм поразрядной сортировки *radix* и стандартный алгоритм стабильной сортировки $std :: stable\_sort$. На вход обоим алгоритмам даётся массив из миллиона объектов.

```
timur@workstation$ g++ -std=c++2a benchmark.cpp -o bench
timur@workstation$ ./bench
[radix] =>78ms
[std::stable_sort] =>390ms
```

Таким образом несложно заметить видим, что поразрядная сортировка имеет преимущество в 5 раз! Размер входных данных составлял один миллион объектов, т.е. достаточно большой. На нём, очевидно, алгоритм с линейной сложностью будет весьма быстрее.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я изучил и реализовал алгоритм поразрядной сортировки сложных объектов. Этот алгоритм имеет линейную сложность, т.е. очень эффективен и может быть применён в различных задачах, в которых нужно упорядочить элементы, ключ которых принимает опередёлённое конечное (небольшое) множество значений. Хочется сказать, что данная работа научила меня более осторожно относиться к использованию памяти в программе, искать способы снижения её потребления и более эффективного использования.

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] *Поразрядная сортировка — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 09.03.2024).