Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Дискретный анализ" №1

Студент: Кострюков Е.С.

Преподаватель: Макаров Н.К.

Группа: М8О-207Б-22

Дата:

Оценка:

Подпись:

Москва 2024 г.

Оглавление

Цель работы

Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.
Вариант задания определяется типом ключа (и соответствующим ему методом сортировки) и типом значения.

Постановка задачи

Вариант 5.2

Поразрядная сортировка.

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

Тип значения: строки переменной длины (до 2048 символов).

Общий алгоритм решения

Структура Pairs представляет пары ключ-значение. Элемент структуры состоит из массива key для хранения ключа и указателя value на значение.

Для удобства работы с ключами, в коде представлена функция get_int_from_char для преобразования символов ASCII в числовые значения.

В функции radix_sort реализован сам алгоритм поразрядной сортировки. Он разбивает ключи на отдельные символы, сортирует по каждому символу начиная с последнего, сохраняя при этом порядок совпадающих ключей.

В функции main, программа считывает пары ключ-значение с консоли, выделяет память под них, вызывает функцию radix_sort для их сортировки, и после сортировки выводит отсортированные пары на экран.

В конце программы выделенная память освобождается.

Алгоритм поразрядной сортировки оказывается быстрым в данном случае, поскольку ключи имеют фиксированную длину.

Реализация

```
#include <iostream>
#include <cstring>

const short int MAX_STRING_LENGTH = 2048;
const int MAX_AMOUNT_PAIRS = 1e6;
const short int RADIX = 32;

struct Pairs {
   char key[RADIX + 1];
   char* value;

   Pairs() : value(nullptr) {}


   // Конструктор копирования
   Pairs(const Pairs& other) {
      std::strcpy(key, other.key);
      value = new char[std::strlen(other.value) + 1];
      std::strcpy(value, other.value);
   }

   // Конструктор перемещения
   Pairs(Pairs&& other) noexcept : value(other.value) {
      for (int i = 0; i < RADIX + 1; ++i)
         key[i] = std::move(other.key[i]);

      other.value = nullptr;
   }

   // Оператор присваивания
   Pairs& operator=(const Pairs& other) {
      if (this != &other) {
         std::strcpy(key, other.key);
         delete[] value;
         value = new char[std::strlen(other.value) + 1];
         std::strcpy(value, other.value);
      }
      return *this;
   }

   // Оператор перемещения
   Pairs& operator=(Pairs&& other) noexcept {
      if (this != &other) {
         for (int i = 0; i < RADIX + 1; ++i)
            key[i] = std::move(other.key[i]);

         delete[] value;
         value = other.value;
         other.value = nullptr;
      }
      return *this;
   }
```

```
    ~Pairs() {
        delete[] value;
    };
};

int get_int_from_char(char c) {
    if (c >= '0' && c <= '9')
        return c - '0';

    else if (c >= 'a' && c <= 'f')
        return 10 + c - 'a';

    return 10 + c - 'A';
}

void radix_sort(Pairs *pairs, int &size) {
    Pairs *sorted = new Pairs[size];
    for (int number_of_radix = 0; number_of_radix < RADIX; ++number_of_radix) {
        int temporary[16] = {0};
        for (int count = 0; count < size; ++count)
            ++temporary[get_int_from_char(pairs[count].key[RADIX - 1 - number_of_radix])];

        for (int count = 1; count < 16; ++count)
            temporary[count] += temporary[count - 1];

        short int symbol = 0;
        for (int count = size - 1; count >= 0; --count) {
            symbol = get_int_from_char(pairs[count].key[RADIX - 1 - number_of_radix]);
            sorted[temporary[symbol] - 1] = std::move(pairs[count]);;
            --temporary[symbol];
        }

        for (int count = 0; count < size; ++count)
            pairs[count] = std::move(sorted[count]);
    }
    delete[] sorted;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);

    Pairs *pairs = new Pairs[MAX_AMOUNT_PAIRS];

    int size = 0;
    char buffer_string[MAX_STRING_LENGTH + 1];
    while (std::cin >> pairs[size].key >> buffer_string){
        pairs[size].value = new char[std::strlen(buffer_string) + 1];
        std::strcpy(pairs[size].value, buffer_string);
        ++size;
    }

    radix_sort(pairs, size);
```

```
    for (int count = 0; count < size; ++count)
        std::cout << pairs[count].key << "\t" << pairs[count].value << "\n";

    delete[] pairs;
}
```

Пример работы

Input	Output
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr 00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr	00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG 00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr

Сравнение сортировок

	radix sort	**bubble sort**	**std::sort**
seconds	0.0374648	22.9251	0.0805886

Сравнение проводилось на массиве из 100 000 элементов. Из таблицы мы можем видеть, что встроенная сортировка работает немного хуже поразрядной. Вероятно, это связано с оптимизацией std::sort. Пузырьковая сортировка оказалась самой медленной и выполнялась гораздо дольше остальных сортировок.

Вывод

В ходе выполнения лабораторной работы я узнал про сортировки за линейное время и научился работать с ними. Для этого мне пришлось вспомнить особенности работы с памятью в C++, так как это может сильно помочь в оптимизации программы. Также было проведено ислдеование, которое показало, насколько линейные сортировки быстрее тех, что выполняются O(n^2).