Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа по курсу "Дискретный анализ" №1

Студент: Клименко В. М.

Преподаватель: Макаров Н. К.

Группа: М8О-203Б-22

Дата: _____

Оценка: _____

Подпись: _____

Москва 2024 г.

Оглавление

Москва 2024 г.

Цель работы

Научиться писать алгоритмы сортировки, работающие за *O(n)*.

Постановка задачи

Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа с помощью **поразрядной сортировки** за линейное время и вывод отсортированной последовательности.

Тип ключа: даты в формате DD.MM.YYYY, например 1.1.1, 1.9.2009, 01.09.2009, 31.12.2009.

Тип значения: строки фиксированной длины 64 символа, во входных данных могут встретиться строки меньшей длины, при этом строка дополняется до 64-х нулевыми символами, которые не выводятся на экран.

Общие сведения о программе

Программа состоит из трех функций:

`main` - точка входа программы, происходит считывание в массив пар ключ-значение, вызывается функция `radix_sort`, после чего выводится отсортированный массив.

`radix_sort` - поразрядная сортировка массива по ключу

`counting_sort` - сортировка подсчетом, используется внутри поразрядной сортировки

Для удобства было создано две структуры данных - `key_value_pair`, `key_value_pairs`. Первая хранит в себе пару ключ-значение (и числовое представление ключа), вторая - динамический массив первой структуры данных.

Общий алгоритм решения

Поразрядная сортировка - сортировка подсчетом, происходящая по каждому “разряду” сортирующего элемента последовательно.

Сортировка подсчетом для сортировки использует массив длиной равной максимальному элементу сортируемого массива. При встрече элемента значения i, в массив на i индексе прибавляется единица. После полного прохода изначального массива, проходим массив, использованный для сортировки с

конца. Когда попадаем на i индекс, в выходной массив записываем n (число на i индексе массива сортировки) чисел i и уменьшаем n на .

Реализация

main.c:

```
#include <stdio.h>
#include <math.h>
#include <inttypes.h>

#include <stdlib.h>
#include <string.h>


#define KEY_LEN 4 + 1 + 2 + 1 + 2 + 1
#define VALUE_LEN 64 + 1


typedef struct {
    uint64_t key; // key = year * 10000 + month * 100 + day, max key is
99991231
    char input_key[KEY_LEN];
    char value[VALUE_LEN];
} key_value_pair;

typedef struct {
    key_value_pair *data;
    uint64_t count;
    uint64_t capacity;
} key_value_pairs;


const uint64_t initial_byte_mask = 0xFF;

void counting_sort(key_value_pairs *pairs, unsigned binary_shift) {
    uint64_t *sorting_array = calloc(initial_byte_mask + 1, sizeof(uint64_t));
    key_value_pair *output_array = calloc(pairs->capacity,
sizeof(key_value_pair));

    uint64_t byte_mask = initial_byte_mask << binary_shift;

    for (uint64_t pair_index = 0; pair_index < pairs->count; ++pair_index) {
        ++sorting_array[(pairs->data[pair_index].key & byte_mask) >>
binary_shift];
    }
```

```
    for (int byte = 1; byte <= initial_byte_mask; ++byte) sorting_array[byte]
+= sorting_array[byte - 1];

    for (uint64_t pair_index = pairs->count; pair_index > 0; --pair_index) {
        int byte = (pairs->data[pair_index - 1].key & byte_mask) >>
binary_shift;
        --sorting_array[byte];
        output_array[sorting_array[byte]] = pairs->data[pair_index - 1];
    }

    key_value_pair *previous_pairs = pairs->data;
    pairs->data = output_array;

    free(previous_pairs);
    free(sorting_array);
}

void radix_sort(key_value_pairs *pairs) {
    for (uint64_t byte_shift = 0; byte_shift < sizeof(uint64_t); ++byte_shift)
{
        unsigned binary_shift = byte_shift * 8;
        counting_sort(pairs, binary_shift);
    }
}

int main() {
    key_value_pairs pairs = (key_value_pairs) {
        .data = calloc(2, sizeof(key_value_pair)),
        .capacity = 2,
        .count = 0
    };

    uint64_t year = 0, month = 0, day = 0;

    char input_key[KEY_LEN];
    char value[VALUE_LEN];

    while (scanf("%s\t%s", input_key, value) > 0) {
        if (pairs.count + 1 > pairs.capacity) {
            pairs.capacity *= 2;
            void *new_pairs_ptr = realloc(pairs.data, pairs.capacity *
sizeof(key_value_pair));
            if (new_pairs_ptr != NULL) pairs.data = (key_value_pair*)
new_pairs_ptr;
            else exit(1);
        }

        sscanf(input_key, "%"PRIu64".%"PRIu64".%"PRIu64, &day, &month, &year);
```

```
        pairs.data[pairs.count].key = year * 10000 + month * 100 + day;
        strcpy(pairs.data[pairs.count].input_key, input_key);
        strcpy(pairs.data[pairs.count].value, value);

        ++pairs.count;
    }

    radix_sort(&pairs);

    for (uint64_t i = 0; i < pairs.count; ++i) {
        printf("%s\t%s\n", pairs.data[i].input_key, pairs.data[i].value);
    }

    free(pairs.data);

    return 0;
}
```

Пример работы

Input:

1.1.1 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt

01.02.2008 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat

1.1.1 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa

01.02.2008 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na

Output:

1.1.1 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt

1.1.1 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa

01.02.2008 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat

01.02.2008 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na

Сравнение с std::sort

Количество входных данных	Поразрядная сортировка, миллисекунд	Стандартная сортировка, миллисекунд
10000	2	2

100000	25	31
1000000	250	380
10000000	2600	4500

Поразрядная сортировка работает быстрее, так как ее сложность – $O(n \cdot m)$, а std::sort'а – $O(n \cdot \log(n))$.

Вывод

В ходе лабораторной работы я научился писать сортировки, работающие за линейное время – сортировку подсчетом и поразрядную сортировку – и использовать их на нечисленных даных.

Москва 2024 г.