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: В. С. Епанешников
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 07.10.2020
Оценка:
Подпись:

Москва, 2021

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: Сортировка подсчётом.

Вариант ключа: Почтовые индексы.

Вариант 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.

1 Описание

Требуется написать реализацию алгоритма сортировки подсчётом. В качестве ключа выступают почтовые индексы.

Как сказано в [2]: «Идея алгоритма состоит в предварительном подсчете количества элементов с различными ключами в исходном массиве и разделении результирующего массива на части соответствующей длины (будем называть их блоками). Затем при повторном проходе исходного массива каждый его элемент копируется в специально отведенный его ключу блок, в первую свободную ячейку. Это осуществляется с помощью массива индексов P, в котором хранятся индексы начала блоков для различных ключей. $P[key]$ — индекс в результирующем массиве, соответствующий первому элементу блока для ключа key.».

Алгоритм состоит из двух проходов по массиву A размера n и одного прохода по массиву P размера k. Его трудоемкость, таким образом, равна $O(n + k)$. На практике сортировку подсчетом имеет смысл применять, если $k = O(n)$, поэтому можно считать время работы алгоритма равным $O(n)$. Как и в обычной сортировке подсчетом, требуется $O(n + k)$ дополнительной памяти — на хранение массива B размера n и массива P размера k.

2 Исходный код

Разобьем процесс написания программы на несколько этапов

1. Реализация необходимых новых типов (пара «ключ-значение» и вектор)
2. Реализация алгоритма сортировки подсчетом вектора пар по ключу по заданному разряду заданной длины
3. Реализация ввода-вывода
4. Реализация бенчмарка

Так как все функции и типы шаблонные, то пара, вектор и сортировки будут находиться в заголовочных файлах *pair.hpp*, *vector.hpp* и *sort.hpp* соответственно. Начнем с реализации пары и вектора:

Структура $TPair$ для хранения пар «ключ-значение»:

```
1  #pragma once
2
3  #include <iostream>
4
5  struct TPair {
6      uint32_t key;
7      char value[65];
8  };
9
10 std::ostream& operator<<(std::ostream& output, const TPair& p) {
11     output.fill('0');
12     output.width(6);
13     output << p.key << '\t' << p.value;
14     return output;
15 }
16
17 std::istream& operator>>(std::istream& input, TPair& p) {
18     input >> p.key >> p.value;
19     return input;
20 }
```

Шаблонный класс $TVector < T >$ для хранения пар «ключ-значение»:

vector.h	
TVector()	Конструктор по умолчанию
TVector(size_t size, T& value)	Конструктор от двух аргументов: размер вектора и значения по умолчанию

TVector(const TVector& other) : TVector();	Конструктор копирования
TVector& operator=(TVector other)	Оператор присваивания с копированием
TVector()	Деструктор
size_t Size() const	Размер вектора
void Swap(TVector& lhs, TVector& rhs)	Обмен векторов значениями
void PushBack(const T& value)	Добавление элемента в конец
const T& operator[](size_t index) const	Получение константной ссылки на элемент по индексу
T& operator[](size_t index)	Получение ссылки на элемент по индексу

```
    template <typename T>
    class TVector {
    public:
        TVector() {}
        TVector(size_t size) {}
        TVector(size_t size, uint64_t value) {}
        TVector(const TVector& other) : TVector() {}

        size_t Size() const {}

        void PushBack(const T& value) {}

        const T& operator[](size_t index) const {}

        T& operator[](size_t index) {}

        void Swap(TVector& lhs, TVector& rhs) {}

        TVector& operator=(TVector other) {}

        ~TVector() {}

    private:
        size_t size_;
        size_t capacity_;
        T* body_;

        void Swap(size_t& lhs, size_t& rhs) {}
    };
};
```

Теперь перейдём к реализации алгоритма сортировки подсчетом по ключу:

```
void CountingSort(TVector<TPair>& elems) {
    uint32_t max = 0;
    for (size_t i = 0; i < elems.Size(); ++i) {
        max = std::max(max, elems[i].key);
    }

    TVector<uint32_t> counters(max + 1, 0);
    for (size_t i = 0; i < elems.Size(); ++i) {
        ++counters[elems[i].key];
    }
    for (size_t i = 1; i < counters.Size(); ++i) {
        counters[i] += counters[i - 1];
    }

    TVector<TPair> result(elems.Size());
    for (int i = elems.Size() - 1; i >= 0; --i) {
        result[counters[elems[i].key] - 1].key = elems[i].key;
        memcpy(result[counters[elems[i].key] - 1].value, elems[i].value, sizeof(
            elems[i].value));
        --counters[elems[i].key];
    }

    elems = result;
}
```

Обработка ввода-вывода в соответствии с описанием задания:

```
#include "pair.hpp"
#include "vector.hpp"
#include "sort.hpp"

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);

    TVector<TPair> v;
    TPair pair;

    while (std::cin >> pair) {
        v.PushBack(pair);
    }

    CountingSort(v);
    for (size_t i = 0; i < v.Size(); ++i) {
        std::cout << v[i] << '\n';
    }

    return 0;
}
```

Обработка ввода и бенчмарк:

```
    #include "pair.hpp"
#include "vector.hpp"
#include "sort.hpp"

#include <iostream>
#include <cstdint>
#include <chrono>
#include <algorithm>

bool operator<(const TPair& lhs, const TPair& rhs) {
    return lhs.key < rhs.key;
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    TVector<TPair> v;
    TVector<TPair> a;
    TPair pair;

    auto start = std::chrono::steady_clock::now();
    while (std::cin >> pair) {
        v.PushBack(pair);
    }
    auto finish = std::chrono::steady_clock::now();
    auto dur = finish - start;
    std::cerr << "input" << ' ' << std::chrono::duration_cast<std::chrono::milliseconds
        >(dur).count() << " ms" << std::endl;

    start = std::chrono::steady_clock::now();
    std::stable_sort(v.begin(), v.end());
    //CountingSort(v);
    finish = std::chrono::steady_clock::now();
    dur = finish - start;
    std::cerr << "sort" << ' ' << std::chrono::duration_cast<std::chrono::milliseconds
        >(dur).count() << " ms" << std::endl;

    start = std::chrono::steady_clock::now();
    for (const auto& i : v) {
        std::cout << i << '\n';
    }
    finish = std::chrono::steady_clock::now();
    dur = finish - start;
    std::cerr << "output" << ' ' << std::chrono::duration_cast<std::chrono::
        milliseconds>(dur).count() << " ms" << std::endl;

    return 0;
}
```

3 Консоль

```
MacBook:solution vladislove$ make
g++ -std=c++17 -pedantic -g -Wall -Wextra -Wno-unused-variable -c lab1.cpp
-o lab1.o
g++ -std=c++17 -pedantic -g -Wall -Wextra -Wno-unused-variable lab1.o -o solution
MacBook:solution vladislove$ ./solution
543253 abc
000000 def
544253 abc
000000 zzz
101010 mmm
355555 zzz
702116 xyz
000000  def
000000  zzz
101010  mmm
355555  zzz
543253  abc
544253  abc
702116  xyz
```

4 Тест производительности

Тесты производительности представляет из себя следующее: сортировку 1 миллиона входных данных с помощью реализованной сортировки подсчётом и $std :: stable\_sort$.

Моя реализация:

```
MacBook:solution vladislove$ ./a.out <test.txt >result.txt
input 6817 ms
sort 250 ms
output 502 ms
```

$std :: stable\_sort$:

```
MacBook:solution vladislove$ ./a.out <test.txt >result.txt
input 6711 ms
sort 378 ms
output 496 ms
```

Как видно, сортировка подсчётом работает несколько быстрее. На этом тесте была продемонстрирована разница в ассимптотике между $O(n)$ и $O(\log_2 n)$.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился писать сортировку подсчётом за линейное время, закрепил навыки работы с памятью. Также я освоил новый Code-Style и вспомнил, как работать с make и TEX'ом Преимуществами сортировки подсчётом являются сложность алгоритма $O(n)$ и лёгкая реализация. Алгоритм особо эффективен, когда мы сортируем большое количество чисел, значения которых имеют небольшой разброс. Недостаток - требуется дополнительная память.

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 — Вики университета ITMO.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_подсчётом`.