

**Московский авиационный институт
(национальный исследовательский университет)**

**Институт информационных технологий и прикладной
математики**

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: Л. В. Ивенкова
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19
Дата:
Оценка:
Подпись:

Москва, 2020

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

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 поразрядной сортировки.

Как сказано в [1]: «в алгоритме поразрядной сортировки мы предполагаем, что каждый ключ сортировки можно рассматривать как d -значное число, каждая цифра которого находится в диапазоне от 0 до $m - 1$. Мы поочередно используем устойчивую сортировку для каждой цифры справа налево. Если в качестве устойчивой применяется сортировка подсчетом, то время сортировки по одной цифре составляет $O(m + n)$, а время сортировки по всем d цифрам - $O(d(m + n))$. Если d является константой ..., то время работы поразрядной сортировки становится равным $O(d * n)$. Если d также представляет собой константу ..., то время работы поразрядной сортировки превращается в просто $O(n)$.»

Я буду следовать этому алгоритму. В качестве устойчивой сортировки возьму поразрядную сортировку. А в качестве разрядов - дни, месяцы и годы в датах.

Единственное отличие моей сортировки от описанной выше - это то, что я буду идти по разрядам ключа (даты) не справа налево, а наоборот. Т.к. в случае чисел правый разряд имеет «наименьшее» значение, и в случае дат, именно день также имеет «наименьшее» значение.

2 Исходный код

На каждой непустой строке файла располагается пара «ключ - значение», поэтому создаем структуру TPair, где хранятся ключ и значение.

Чтобы сортировка работала быстро, нам нужно иметь доступ к произвольному элементу за $O(1)$. Для этого реализуем шаблон класса TVector, который по функционалу похож на std::vector. В нём мы будем хранить наши пары «ключ - значение».

Рассмотрим алгоритм сортировки:

1. На вход поступает набор элементов (по ссылке) и максимальное значение среди этого набора (по константной ссылке).
2. Разбиваем ключ на три разряда - день, месяц, год.
3. Для каждого из трёх разрядов осуществляем сортировку подсчётом. Начинаем с наименее значимого разряда - с дней.
 - Создаём временный вектор чисел tmp (его размер на 1 больше максимального элемента). Инициализируем его нулями.
 - Далее считаем сколько раз каждое число встретилось в начальном векторе.
 - Затем к каждому элементу вектора tmp прибавляем его предыдущий элемент.
 - Создаём результирующий вектор. Заполняем его.
 - Меняем местами значения начального вектора и результирующего.
4. В результате работы сортировки мы получаем изменённый вектор.

Сам код:

```
1 #include <iostream>
2 #include <cstdio>
3 #include <string.h>
4 template <typename T>
5 class TVector{
6 public:
7     TVector() = default;
8     ~TVector() {
9         delete[] arr;
10    }
11    TVector(const size_t len){
12        arr = new T[len];
13        size = len;
```

```

14     capecity = len;
15 }
16 TVector(const size_t len, const T value){
17     arr = new T[len];
18     size = len;
19     capecity = len;
20     for (size_t i = 0; i < size; i++) {
21         arr[i] = value;
22     }
23 }
24 void PushBack(const T& elem){
25     if (size == capecity) {
26         capecity++;
27         capecity = capecity*2;
28         size_t new_size = capecity;
29         T* tmp = new T[new_size];
30         std::copy(begin(), end(), tmp);
31         delete[] arr;
32         arr = tmp;
33     }
34     arr[size] = elem;
35     size++;
36 }
37 void swap(TVector& b){
38     T* c = arr;
39     arr = b.arr;
40     b.arr = c;
41 }
42 T& operator[] (const int i){
43     return arr[i];
44 }
45 TVector& operator=(TVector& other) {
46     if (&other == this) {
47         return *this;
48     }
49     if (other.size <= capecity) {
50         std::copy(other.begin(), other.end(), begin());
51         size = other.size;
52     }
53     else {
54         TVector<T> tmp(other);
55         std::swap(tmp.arr, arr);
56         std::swap(tmp.size, size);
57         std::swap(tmp.capecity, capecity);
58     }
59     return *this;
60 }
61 size_t Size() const {
62     return size;

```

```

63 }
64 T* begin(){
65     return arr;
66 }
67 T* end(){
68     return arr + size;
69 }
70
71 private:
72     size_t size = 0;
73     size_t capacity = 0;
74     T* arr = nullptr;
75 };
76
77 struct TPair{
78     int znacheniye[3] = {0};
79     char stroka[11] = {'\0'};
80     char val[65];
81 };
82
83 int RadixSort(TVector<TPair> &elems, const int (&max)[3]) {
84     size_t N = elems.Size();
85     if (N == 0) {
86         return 0;
87     }
88     for (int i = 0; i < 3; ++i) {
89         TVector<int> tmp((max[i] + 1), 0);
90         for (size_t j = 0; j < N; ++j) {
91             tmp[elems[j].znacheniye[i]]++;
92         }
93
94         for (size_t j = 1; j < tmp.Size(); ++j) {
95             tmp[j] += tmp[j - 1];
96         }
97
98         TVector<TPair> res(N);
99         for (int j = N - 1; j > (-1); j--) {
100             res[tmp[elems[j].znacheniye[i]] - 1] = elems[j];
101             --tmp[elems[j].znacheniye[i]];
102         }
103
104         elems.swap(res);
105     }
106     return 0;
107 }
108
109
110 int main(){
111     std::ios::sync_with_stdio(false);

```

```

112     std::cin.tie(nullptr);
113     std::cout.tie(nullptr);
114     int max[3] = {0};
115
116     TVector<TPair> elems;
117     TPair x;
118
119     while (scanf("%s %s", x.stroka, x.val)>0) {
120         sscanf(x.stroka, "%d.%d.%d", &x.znacheniye[0], &x.znacheniye[1], &x.znacheniye[2]);
121         elems.PushBack(x);
122         if(x.znacheniye[0] > max[0]) max[0] = x.znacheniye[0];
123         if(x.znacheniye[1] > max[1]) max[1] = x.znacheniye[1];
124         if(x.znacheniye[2] > max[2]) max[2] = x.znacheniye[2];
125     }
126
127     RadixSort(elems, max);
128
129     for (size_t i = 0; i < elems.Size(); ++i) {
130         std::cout << elems[i].stroka << " " << elems[i].val << "\n";
131     }
132     return 0;
133 }
```

3 Консоль

```
parsifal@DESKTOP-3G70RV4:~/DA/lab1/make/solution$ g++ main.cpp
parsifal@DESKTOP-3G70RV4:~/DA/lab1/make/solution$ cat test.t
1.1.1 n399tann9annt3tn93aat3naatt
01.02.2008 n399n9nann93na9t3an9annt3tn93aat3naat
1.1.1 n399tann9nnt3ttnaaan9nann93nt3tn93aat3naa
01.02.2008 n399tann9nann93na9t3a3t9999na3a3na
23.12.9999 ubintsthrstjry
01.1.1 hhhhhhhhhhhhhh
parsifal@DESKTOP-3G70RV4:~/DA/lab1/make/solution$ ./a.out <test.t
1.1.1 n399tann9annt3tn93aat3naatt
1.1.1 n399tann9nnt3ttnaaan9nann93nt3tn93aat3naa
01.1.1 hhhhhhhhhhhhhh
01.02.2008 n399n9nann93na9t3an9annt3tn93aat3naat
01.02.2008 n399tann9nann93na9t3a3t9999na3a3na
23.12.9999 ubintsthrstjry
```

4 Тест производительности

Тут Вы описываете собственно тест производительности, сравнение Вашей реализации с уже существующими и т.д.

Тест производительности представляет из себя следующее: мы сортируем начальный вектор пар «ключ - значение» двумя сортировками: нашей поразрядной сортировкой и встроеноой сортировкой std::stable_sort.

Для этого создаём программу для генерации тестов.

```
1 import sys
2 import random
3 import string
4
5 def get_random_string():
6     length = random.randint(1, 64)
7     random_list = [random.choice(string.ascii_letters) for _ in range(length)]
8     return ''.join(random_list)
9
10 def main():
11     n = random.randint(100, 100)
12     for _ in range(n):
13         a = random.randint(1, 31)
14         b = random.randint(1, 12)
15         c = random.randint(1, 9999)
16         value = get_random_string()
17         print(f'{a}.{b}.{c} {value}')
18 main()
```

И проверяем время работы сортировок (с помощью библиотеки chrono - она позволяет замерить время от запуска сортировки до её завершения). Причём проверям на одинаковых данных, но разных рамеров.

```
parsifal@DESKTOP-3G70RV4:~/DA/lab1/make/solution$ python3 generator.py >1.t
parsifal@DESKTOP-3G70RV4:~/DA/lab1/make/solution$ g++ benchmark1.cpp
parsifal@DESKTOP-3G70RV4:~/DA/lab1/make/solution$ ./a.out <1.t
Count of lines is 10000
Counting sort time: 3999us
STL stable sort time: 4170us
```

Количество строк в тесте	Время работы сортировок (мкс)
100	Counting sort time: 150 STL stable sort time: 25
1000	Counting sort time: 283 STL stable sort time: 332
5000	Counting sort time: 1774 STL stable sort time: 1866
10.000	Counting sort time: 3999 STL stable sort time: 4170
100.000	Counting sort time: 39795 STL stable sort time: 49599
1.000.000	Counting sort time: 411158 STL stable sort time: 705164

Так как в данном случае у нас m - константа, то сложность поразрядной сортировки становится $O(d * n)$, где n - число строк в тестовом файле, а $d = 3$ - число разрядов. Можно увидеть, что при количестве текстов (n) большем или равном 1000 поразрядная сортировка опережает встроенную. При значениях же меньших 1000 встроенная сортировка опережает поразрядную, т.к. $O(3 * n) < O(\log(n) * n) = O(2 * n)$

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научилась с помощью шаблонных классов реализовывать класс `vector` и с его помощью писать сортировки за линейное время (поразрядную и сортировку подсчётом). Научилась генерировать тесты для этих сортировок, а также оценивать их сложность и эффективность.

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).