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: М. В. Леухин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-20

Дата:
Оценка:
Подпись:

Москва, 2022

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: `телефонные номера, с кодами стран и городов в формате +<код страны> <код города> телефон.`

Вариант значения: `строки переменной длины (до 2048 символов).`

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки.

Основная идея поразрядной сортировки заключается в том, что сначала производится сортировка по младшему разряду, после по предпоследнему и т.д. до тех пор, пока входные данные не будут отсортированы по всем разрядам [1].

Как следует из описания, данная сортировка подходит для таких данных, в которых можно выделить поразрядную структуру — это не только числа, но и, например, строки. Чтобы сортировка получилась действительно линейной, в пределах одного разряда данные нужно сортировать так же с использованием сортировки линейной сложности — например, с помощью сортировки подсчётом. Основная её идея заключается в том, чтобы для каждого элемента x определить количество элементов, которые меньше x. С помощью этой информации можно разместить элемент x в нужной позиции выходной последовательности [1]

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение». Ключ и значение будем хранить в двух массивах с элементами типа *char*: ключ — в статическом массиве размера $SIZE = 32$, а значение — в динамическом. Объединённые ключ и значение будем хранить в *pair*.

После ввода значений непосредственно сортировка выполняется в цикле по всем разрядам (их коичество определяется максимальной длиной введённого ключа). В пределах разряда для сортировки элементов применяется сортировка подсчётом. Опишем этот процесс подробнее.

Сортировка по разряду d производится во время очередной итерации. В её пределах создаётся вектор tmp на 10 элементов и производится подсчёт сортируемых элементов — значение $tmp[i]$ показывает, сколько элементов имеют цифру i в разряде d. После того, как подсчёт произведён, для элементов $tmp[i], i = 1...9$ производится следующее преобразование: $tmp[i] = tmp[i-1] + tmp[i]$. Таким образом мы получаем вектор, в котором значение $tmp[i]$ показывает количество элементов, у которых в разряде d находится цифра, меньшая или равна i. Зная эту информацию, мы можем разместить очередной элемент на необходимой позиции в результирующем векторе, а именно на позиции с номером $tmp[i] - 1$, после чего нужно уменьшить значение $tmp[i]$ на 1.

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <cstring>
5  #include <algorithm>
6
7  using namespace std;
8
9  const int SIZE = 32;
10 using TPair = pair<char[SIZE], char*>;
11 using TVector = vector<TPair>;
12
13 int main() {
14     ios::sync_with_stdio(false);
15     cin.tie(nullptr);
16     cout.tie(nullptr);
17
18     TPair pair;
19     TVector elements;
20     string key;
21     string value;
22
23     size_t maxLength = 0;
24     while (cin >> key >> value) {
```

```
25          pair.first[0] = key.size();
26          for (size_t i = SIZE - key.size(); i < SIZE; i++) {
27              pair.first[i] = key[key.size() - (SIZE - i)];
28          }
29          pair.second = new char[value.length()];
30          strcpy(pair.second, value.c_str());
31          maxLength = max(key.size(), maxLength);
32          elements.push_back(pair);
33      }
34
35      TVector result(elements.size());
36
37      for (int pos = SIZE - 1; pos > SIZE - maxLength - 1; pos--) {
38
39          vector<int> tmp(10, 0);
40
41          for (TPair &element: elements) {
42              int num = (SIZE - pos <= element.first[0]) ? max(element.first[pos] - '0',
                    0) : 0;
43              tmp[num]++;
44          }
45
46          for (int i = 1; i < 10; i++) {
47              tmp[i] = tmp[i] + tmp[i - 1];
48          }
49
50          for (int i = elements.size() - 1; i >= 0; i--) {
51              int num = SIZE - pos <= elements[i].first[0] ? max(elements[i].first[pos] -
                     '0', 0) : 0;
52              tmp[num] -= 1;
53              result[tmp[num]].first[0] = elements[i].first[0];
54              for (int j = SIZE - elements[i].first[0]; j < SIZE; j++) {
55                  result[tmp[num]].first[j] = elements[i].first[j];
56              }
57              result[tmp[num]].second = elements[i].second;
58          }
59
60          swap(elements, result);
61
62      }
63
64      for (TPair & element : elements) {
65          for (int j = SIZE - element.first[0]; j < SIZE; j++) {
66              cout << element.first[j];
67          }
68          cout << '\t' << element.second << '\n';
69          delete element.second;
70      }
71  }
```

3 Консоль

```
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/1lab$ g++ -std=c++17 -pedantic
-Wall -Wextra -Wno-unused-variable lab1.cpp
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/1lab$ cat tests/03.t
+57-068-5593298 smOaL
+62801-679-3972201 vFCsOmaFzUQfuvLSVVKNRTurZpqlrqxiSzwUKTfNQArzKWmAYxUJRZjAgIzvxELvtgS
+9415-519-9719948 UXdghjYgfysjhihKHKrjIgFJcxIHYrzXcAyYrVuCabHOXljReSNAizYCVQOEAeoqxpSI
+7775-034-7619763 swATybQILMPlJtcsdhYLuMVPPWTgAdUmmwMwauqsUcrwiieugolAZWUwUXJBdLoaKkwC
+2947-095-8126677 fbecaAtWEWkVKhuwnirqRXPpkRUFXZXxiZyBWjCwEeTJM
+50020-988-3735763 MOQgYryHtloCAIVkVjXvnVBcvUdaSddnajBAuoxjMERPlPdqFKlYoLcnPXF
+13-542-5708693 PzJNesfThvRraGkwJUkMkECGbTWeecuogvztvFNCiSslfjeGrgOywZHWpZcvoJkQIMplWT
+867-420-0614850 TuLTytRTJEsttZHnIoFiSjVwRMMhNIIrFeJBetkqpNVyZJMAiCtibLJYkZlFucIhAG
+4-161-1137627 eDhZcKPJyBFjWnqpboKWzzbMZnlLMTXXLfJARUSquriZQRY
+0-893-5958471 JGDeytoPgwkTjoKug
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/1lab$ ./lab1 <tests/03.t
+0-893-5958471  JGDeytoPgwkTjoKug
+4-161-1137627  eDhZcKPJyBFjWnqpboKWzzbMZnlLMTXXLfJARUSquriZQRY
+13-542-5708693 PzJNesfThvRraGkwJUkMkECGbTWeecuogvztvFNCiSslfjeGrgOywZHWpZcvoJkQIMplWT
+57-068-5593298 smOaL
+867-420-0614850        TuLTytRTJEsttZHnIoFiSjVwRMMhNIIrFeJBetkqpNVyZJMAiCtibLJYkZlFuc
+2947-095-8126677       fbecaAtWEWkVKhuwnirqRXPpkRUFXZXxiZyBWjCwEeTJM
+7775-034-7619763       swATybQILMPlJtcsdhYLuMVPPWTgAdUmmwMwauqsUcrwiieugolAZWUwUXJBdL
+9415-519-9719948       UXdghjYgfysjhihKHKrjIgFJcxIHYrzXcAyYrVuCabHOXljReSNAizYCVQOEAe
+50020-988-3735763      MOQgYryHtloCAIVkVjXvnVBcvUdaSddnajBAuoxjMERPlPdqFKlYoLcnPXF
+62801-679-3972201      vFCsOmaFzUQfuvLSVVKNRTurZpqlrqxiSzwUKTfNQArzKWmAYxUJRZjAgIzvxL
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение эффективности реализованной поразрядной сортировки и сортировки из стандартной библиотеки C++.

```
# 100 lines
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/1lab$ ./lab1 <tests/01.t
Radix sort time: 2345us
STL stable sort time: 612us
# 100000 lines
matvey@matvey-Lenovo-IdeaPad-S340-15API:~/labs/3da/1lab$ ./lab1 <tests/02.t
Radix sort time: 1173374us
STL stable sort time: 1273545us
```

Тесты показали, что на входных данных достаточно большого объёма сортировка подсчётом по времени работает быстрее, чем устойчивая сортировка из стандартной библиотеки C++. Но если размер входных данных невилик, то поразрядная сортировка работает хуже — это связано с тем, что хоть поразрядная сортировка и имеет сложность $O(d(n+k))$ (в сравнении с $O(n \log n)$ устойчивой сортировки из STL), но она имеет досаточно большие значения констант.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я познакомился с сортировками линейной сложности (сортировкой подсчётом и поразрядной сортировкой) и научился их реализовывать. Довольно интересным для меня оказалось доказательство того, почему обменные сортировки не могут работать меньше, чем за $O(n \log n)$, и как сортировки линейной сложности позволяют избежать этого ограничения.

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.))