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: А. К. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

# 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:** автомобильные номера в формате A 999 BC (используются буквы латинского алфавита).

**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

Интернет-ресурс [3] дает следующее описание алгоритма сортировки подсчетом:

Алгоритм сортировки подсчетом предназначен для сортировки целых чисел, записанных цифрами. Но так как в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения. Например, так сортировать можно не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и вообще произвольные значения в памяти, представленные в виде набора байт.

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

Так как выравнивать сравниваемые записи относительно друг друга можно в разную сторону, на практике существуют два варианта этой сортировки. Для чисел они называются в терминах значимости разрядов числа, и получается так: можно выровнять записи чисел в сторону менее значащих цифр (по правой стороне, в сторону единиц, least significant digit, LSD) или более значащих цифр (по левой стороне, со стороны более значащих разрядов, most significant digit, MSD).

При LSD сортировке (сортировке с выравниванием по младшему разряду, направо, к единицам) получается порядок, уместный для чисел. Например: 1, 2, 9, 10, 21, 100, 200, 201, 202, 210. То есть, здесь значения сначала сортируются по единицам, затем сортируются по десяткам, сохраняя отсортированность по единицам внутри десятков, затем по сотням, сохраняя отсортированность по десяткам и единицам внутри сотен, и т. п.

При MSD сортировке (с выравниванием в сторону старшего разряда, налево), получается алфавитный порядок, который уместен для сортировки строк текста. Например «b, c, d, e, f, g, h, i, j, ba» отсортируется как «b, ba, c, d, e, f, g, h, i, j». Если MSD применить к числам, приведённым в примере получим последовательность 1, 10, 100, 2, 200, 201, 202, 21, 210, 9.

Накапливать при каждом проходе сведения о группах можно разными способами — например в списках, в деревьях, в массивах, выписывая в них либо сами элементы, либо их индексы и т. п.

# 2 Исходный код

Здесь должно быть подробное описание программы и основные этапы написания кода.

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $KV$, в которой будем хранить ключ и значение. И так далее.

Метод решения Для совместного хранения ключей и значений, создана структура $KV$ (она состоит из массива $charkey[9]$ и $charvalue[64]$ для хранения ключа и значения соответственно). Считываю данные и добавляю их в вектор, который в последствии передаю в функцию поразрядной сортировки $RadixSort$, работающую по следующему алгоритму:

1. Итерация по каждому символу в ключе (8 символов): - Для каждого символа в ключе, начиная с наименее значимого символа:

2. Проверка символа: - Если символ является цифрой: - Увеличить соответствующий счетчик в digc. - Обновить счетчики для вышестоящих цифр. - Перенести элементы в temp в отсортированном порядке на основе счетчиков. - Если символ является заглавной буквой: - Увеличить соответствующий счетчик в charc. - Обновить счетчики для вышестоящих букв. - Перенести элементы в temp в отсортированном порядке на основе счетчиков.

3. Сброс счетчиков: - Обнулить счетчики charc и digc после каждого цикла.

4. Копирование отсортированного массива: - Скопировать элементы из temp обратно в исходный массив vect.

Алгоритм имеет сложность времени O(mn), где m - количество элементов в массиве, а n - длина ключа, в данном случае 8.

lab1.cpp

```
#include <iostream>

struct KV{
    char key[9];
    char value[65];
};

class TVect {
    int size;
    int cap;
    KV* vectData;

public:
     TVect() {
        size = 0;
        cap = 0;
```

```
17          vectData = new KV[2];
18      }
19
20      TVect(int i) {
21          size = i;
22          cap = i;
23          vectData = new KV[i];
24      }
25
26      KV& operator[] (int k) {
27          return vectData[k];
28      }
29
30      int VectSize() {
31          return size;
32      }
33
34      void PushBack(KV element) {
35          if (size == cap) {
36                  if (cap == 0) {
37                      cap = 1;
38                  }
39                  cap *=2;
40                  KV* temp = new KV[cap];
41                  for (int j = 0; j < size; j++) {
42                      temp[j] = vectData[j];
43                  }
44                  delete [] vectData;
45                  vectData = temp;
46          }
47          vectData[size] = element;
48          size++;
49      }
50
51      ~TVect() {
52          delete [] vectData;
53      }
54  };
55
56  void RadixSort(TVect &vect){
57      char charc[36] = {};
58      char digc[10] = {};
59      TVect temp(vect.VectSize());
60      for (int i = 8; i != 0; i--){
61          if (vect[0].key[i-1] == ' ') {
62              continue;
63          }
64          if (vect[0].key[i-1] >= '0' && vect[0].key[i-1] <= '9') {
65              int p = 0;
```

```
66              int m = 0;
67              for (p = 0; p < vect.VectSize(); p++) {
68                  digc[vect[p].key[i-1] - '0'] += 1;
69              }
70              for (m = 1; m < 10; m++) {
71                  digc[m] += digc[m - 1];
72              }
73              for (p = vect.VectSize(); p != 0; p--) {
74                  --digc[vect[p - 1].key[i-1] - '0'];
75                  temp[digc[vect[p - 1].key[i-1] - '0']] = vect[p - 1];
76              }
77          }
78
79          else {
80              int p = 0;
81              int m = 0;
82              for (p = 0; p < vect.VectSize(); p++) {
83                  charc[vect[p].key[i-1] - 'A'] += 1;
84              }
85              for (m = 1; m < 26; m++) {
86                  charc[m] += charc[m - 1];
87              }
88              for (p = vect.VectSize(); p != 0; p--) {
89                  --charc[vect[p - 1].key[i-1] - 'A'];
90                  temp[charc[vect[p - 1].key[i-1] - 'A']] = vect[p - 1];
91              }
92          }
93          for (int k = 0; k < 26; k++) {
94              charc[k] = 0;
95          }
96          for (int k = 0; k < 10; k++) {
97              digc[k] = 0;
98          }
99          for (int m = 0; m < vect.VectSize(); m++) {
100             vect[m] = temp[m];
101         }
102
103     }
104 }
105
106 int main()
107 {
108     KV el;
109     TVect data;
110     char str[74];
111     while (true)
112     {
113
114         str[0] = '\0';
```

```
115             std::cin.getline(str, 74);
116             int k = 0;
117             int j = 0;
118             if (std::cin.eof()) {
119                 break;
120             }
121             for (k = 0; k < 8; k++) {
122                 el.key[k] = str[k];
123             }
124             el.key[8] = '\0';
125             k += 1;
126             while (k < 74) {
127                 el.value[j++] = str[k++];
128             }
129             data.PushBack(el);
130             for (int i = 0; i < 74; i ++){
131                 str[i] = '\0';
132             }
133             for (int i = 0; i < 64; i ++){
134                 el.value[i] = '\0';
135             }
136         }
137 
138 
139         RadixSort(data);
140 
141         for (int j = 0; j < data.VectSize(); j++) {
142             std::cout << data[j].key << '\t' << data[j].value << std::endl;
143         }
144         return 0;
145     }
```

# 3 Консоль

```
alex@alex-HP-ENVY-x360-Convertible-13-ay1xxx:~/Desktop/DiskrAn/lab1/src$ g++
lab1.cpp
alex@alex-HP-ENVY-x360-Convertible-13-ay1xxx:~/Desktop/DiskrAn/lab1/src$ ./a.out
<input.txt
A 658 MJ        ryuabk1mbxzao2uetjs3h1v619lp3ruievbwwd8mph
B 406 CU        8lq1uloi0mry
B 417 QJ        hg
E 131 VR        kluidx0qf2oa053p48gbkb7xjiczqy6apbb6t8zv6335x2mscds7b
F 436 PF        t0wst8jcn0gmi0q3oc73jqgd86ee
G 046 IS        zhywl3nk6xdoqadirvle7bm8funn368ui4gqh08404
H 669 AB        d3rz838jra3b38vgjuov3d3ep216zntq
```

| | | |
|---|---|---|
| H | 705 | OV |
| I | 023 | VT |
| I | 349 | FE |
| I | 937 | QQ |
| J | 068 | HK |
| J | 469 | EL |
| K | 634 | BG |
| K | 779 | PK |
| M | 319 | YK |
| M | 421 | EJ |
| N | 042 | PP |
| N | 623 | CD |
| N | 959 | IQ |
| O | 312 | IV |
| P | 147 | KH |
| Q | 349 | RK |
| Q | 787 | MK |
| S | 852 | LF |
| V | 451 | GL |
| V | 516 | XT |
| X | 703 | XX |
| X | 771 | LM |
| Y | 886 | HW |

```
qyl2jskmmtt8lx9ej1wtk1rt2uprcmgco6uain1hfnoe
blv0ezn45myxx8sv4bt93ti3ysro4at32snhsmpiblxobzt2
d880k82o1an7
sowjvpqyp03r9j2f9qrrvwp8n78
k935ao4kvhs7fl0gw1dmr5jq36bvcpl7uozshb4ok8qoxr
bjqobk1i1zgmyb8d12wlqfegu2kd2s71pyra4dltpfaeb
xvo3sro2ef3bz37d8z0oqkvh3wg5hxlh9o6te4zq6woxxtxjfc4
ltsxcfvsjoc2c7encjklmza6ptc
iea9i447l8f3sgp3
qgizb11fjllpyd0ae9w5cjdwwxypmnpkeuixx6fdpqvdxwdn
4war2rbmkvn8p9l
iuuk3jzonv938t0reg4gwvrechefz7i5hf0rki4g4
xyku34iwzcvr7erw5i0
wkx8pkq1kz1ul4ibs2c845j29larp3hc7s9sl0i0tvq
25asyett06jh629qkbconoicy8ch0mqgglwy0daboaa0dw2cbx9yq
bbsnz7kr51u6etvhpjd16n2apyv4z0952jhhk7fp800kthuln7yctaw9o
uqilmc
aap66zzq643be0kpzc3ohvxid8carwvxnw8b2v
hsmgxa6n5zfis8qy68kgy1740swo477
5i63n
1
b0uivkrh3v5p9fwejhnukczn4
f
```

# 4 Тест производительности

Тест представлял из себя сравнение поразрядной сортировки, реализованной мной в этой лабораторной работе с STL сртировкой. Задачей была сортировка файла, состоящего из 10 000 пар «ключ-значение», и их упорядочивание по возрастанию ключа.

```
[info][Пт 23 авг 2024 18:01:22 MSK] Compiling OK
[info][Пт 23 авг 2024 18:01:22 MSK] Stage #2. Test generating...
[info][Пт 23 авг 2024 18:01:22 MSK] Test generating OK
[info][Пт 23 авг 2024 18:01:22 MSK] Stage #3. Chacking...
[info][Пт 23 авг 2024 18:01:22 MSK] tests/01.t,lines = 51 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/02.t,lines = 981 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/03.t,lines = 978 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/04.t,lines = 443 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/05.t,lines = 891 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/06.t,lines = 794 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/07.t,lines = 464 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/08.t,lines = 597 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/09.t,lines = 23 OK
[info][Пт 23 авг 2024 18:01:22 MSK] tests/10.t,lines = 765 OK
[info][Пт 23 авг 2024 18:01:22 MSK] Checking OK
[info][Пт 23 авг 2024 18:01:22 MSK] Stage #4 Benchmark test generating...
[info][Пт 23 авг 2024 18:01:41 MSK] Benchmark test generating OK
[info][Пт 23 авг 2024 18:01:41 MSK] Stage #5 Benchmarking...
make: 'benchmark'is up to date.
[info][Пт 23 авг 2024 18:01:41 MSK] Running benchmark.t
Count of lines is 10000
Counting sort time: 88016us
STL Sort time: 21800us
[info][Пт 23 авг 2024 18:01:53 MSK] Benchmarking OK
```

Из примера видно, что при больших объемах данных поразрядная сортировка проигрывает STL, однако при меньших объемах, она показывет себя более эффективной.

Также хочется продемонстрировать примерную линейность времени сортировки n – количество пар «ключ-значение» во входном файле

1. n = 10; time = 0,002s
2. n = 100; time = 0,002s
3. n = 1000; time = 0,014s

4. n = 10000; time = 0,192s

# 5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», мною был изучен и реализован алгоритм сортировки за линейное время – поразрядная сортировка.

Сортировка за линейное время наиболее эффективна при обработке небольшого количества данных. Сложность сортировок этого типа – O(m*n), где m – количество разрядов, по которым происходит сортировка, n – объём входных данных. Для сортировки большого количества данных этот тип сортировок будет неэффективен.

Также стоит отметить устойчивость линейных сортировок – элементы с одинаковыми ключами не меняют порядок в отсортированном наборе данных.

# 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] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008

[3] *Поразрядная сортировка — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 14.03.2024).