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. В. Постнов
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата:
Оценка:
Подпись:

Москва, 2023

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.

7.1:

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

Требуется написать реализацию алгоритма поразрядной сортировки.

Сам алгоритм состоит в последовательной сортировке объектов какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего разряда к старшему, после чего последовательности будут расположены в требуемом порядке. В качестве устойчивой сортировки буду использовать сортировку подсчётом.

2 Исходный код

Формально структура программы состоит из 3 файлов:

1. *radix_sort.h* В нем содержится структура сортируемых объектов, сигнатура функции поразрядной сортировки

2. *radix_sort.cpp* В нем содержится реализация поразрядной сортировки

3. *main_cpp* Ввод объектов, сортировка объектов, вывод объектов

```
#ifndef RADIX_SORT_H
#define RADIX_SORT_H

#include <iostream>
#include <string>
#include <vector>

struct TNode {
  static const size_t KEY_SIZE = 8;
  static const size_t DIGIT_SIZE = 128;
  std::string key;
  std::string *value;
  int GetValue(size_t n) {
    return static_cast<int>(key[n]);
  };
};


void RadixSort(std::vector <TNode>& data, size_t n);

#endif
```

```
#include "radix_sort.h"


void RadixSort(std::vector<TNode> &data, size_t n) {
  for (size_t i = TNode::KEY_SIZE; i > 0; --i) {
    std::vector <TNode> result(n);
    std::vector <int> count(TNode::DIGIT_SIZE);
    for (size_t j = 0; j < n; ++j) {
      count[data[j].GetValue(i - 1)]++;
    }
    for (size_t j = 1; j < TNode::DIGIT_SIZE; ++j) {
      count[j] += count[j - 1];
    }
    for (size_t j = n; j > 0; j--) {
      int index = count[data[j - 1].GetValue(i - 1)];
      result[index - 1] = data[j - 1];
      count[data[j - 1].GetValue(i - 1)]--;
    }
```

```
    for (size_t j = 0; j < n; ++j) {
      data[j] = result[j];
    }
  }
}
```

```
#include <iostream>

#include "radix_sort.h"


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

  std::vector <TNode> data;
  std::string first;
  std::string number;
  std::string second;
  std::string value;
  while (std::cin >> first >> number >> second >> value) {
    TNode elem;
    elem.value = new std::string;
    std::string key;
    key += first;
    key += " ";
    key += number;
    key += " ";
    key += second;
    elem.key = key;
    *elem.value = value;
    data.push_back(elem);
  }
  if (!data.empty()) {
    RadixSort(data, data.size());
  }
  for (const auto & elem : data) {
    std::cout << elem.key << "\t" << *elem.value << "\n";
    delete elem.value;
  }
}
```

3 Консоль

```
[alex@fedora mai-da-labs]$ cd build
[alex@fedora build]$ cmake ../
--Configuring done
--Generating done
--Build files have been written to: /home/alex/mai-da-labs/build
[alex@fedora build]$ cmake --build .
[ 21%] Built target lab1
[ 35%] Built target gtest
[ 50%] Built target gtest_main
[ 57%] Building CXX object tests/CMakeFiles/lab1_test.dir/lab1_test.cpp.o
[ 64%] Linking CXX executable lab1_test
[ 71%] Built target lab1_test
[ 85%] Built target gmock
[100%] Built target gmock_main
[alex@fedora build]$ cd lab1/
[alex@fedora lab1]$ ./lab1
A 999 ZZ ZZZZZZZZZZZ
A 888 ZZ F
B 999 EE DDDDDDDD
H 991 FF SSS
A 888 ZZ F
A 999 ZZ ZZZZZZZZZZZ
B 999 EE DDDDDDDD
H 991 FF SSS
```

4 Тесты

Тест представляет собой программу, в которой содержится:

1. генератор объектов
2. проверка на устойчивость сортировки
3. проверка на правильность с помощью $std :: stable\_sort$
4. сравнивается производительность $RadixSort$ и $std :: stable\_sort$ при $n = 10^6$

```
#include <algorithm>
#include <cstddef>
#include <gtest/gtest.h>
#include <string>
#include <cstdlib>
#include <ctime>
#include <chrono>

#include "radix_sort.h"

bool CompTNode(const TNode& lhs, const TNode& rhs) {
    return lhs.key < rhs.key;
}

namespace NDatagen {
    const size_t ALPHA_SIZE = 26;
    const size_t NUM_SIZE = 10;

    std::string GenerateValue(size_t n) {
        std::string str;
        for (size_t i = 0; i < n; ++i) {
            str += static_cast<char>(static_cast<int>('A') + rand() % ALPHA_SIZE);
        }
        return str;
    }

    std::string GenerateKey() {
        std::string key;
        std::string first;
        std::string num;
        std::string second;
        first += static_cast<char>(static_cast<int>('A') + rand() % ALPHA_SIZE);
        num += static_cast<char>(static_cast<int>('0') + rand() % NUM_SIZE);
        num += static_cast<char>(static_cast<int>('0') + rand() % NUM_SIZE);
        num += static_cast<char>(static_cast<int>('0') + rand() % NUM_SIZE);
        second += static_cast<char>(static_cast<int>('A') + rand() % ALPHA_SIZE);
        second += static_cast<char>(static_cast<int>('A') + rand() % ALPHA_SIZE);
        key += first;
        key += ' ';
        key += num;
```

```
41          key += ' ';
42          key += second;
43          return key;
44      }
45
46      std::vector <TNode> GenerateNode(size_t n) {
47          const size_t valueSize = 64;
48          srand(time(nullptr));
49          std::vector <TNode> data;
50          for (size_t i = 0; i < n; ++i) {
51              TNode elem;
52              elem.value = new std::string;
53              *elem.value = GenerateValue(valueSize);
54              elem.key = GenerateKey();
55              data.push_back(elem);
56          }
57          return data;
58      }
59  }
60
61  TEST(Lab1Test, StableTest) {
62      const size_t size = 5;
63      std::vector <std::pair<std::string, std::string>> data {
64          {"A 000 AA", "AAA1"},
65          {"A 000 AA", "AAA2"},
66          {"A 000 AA", "AAA3"},
67          {"A 000 AA", "AAA4"},
68          {"A 000 AA", "AAA5"}
69      };
70      std::vector <TNode> input(size);
71      std::vector <TNode> output(size);
72      for (size_t i = 0; i < size; ++i) {
73          input[i].value = new std::string;
74          *input[i].value = data[i].second;
75          input[i].key = data[i].first;
76
77          output[i].value = new std::string;
78          *output[i].value = data[i].second;
79          output[i].key = data[i].first;
80      }
81      RadixSort(input, size);
82      ASSERT_EQ(input.size(), output.size());
83      for (size_t i = 0; i < size; ++i) {
84          EXPECT_EQ(input[i].key, output[i].key);
85          EXPECT_EQ(*input[i].value, *output[i].value);
86      }
87  }
88
89  TEST(Lab1Test, CommonTest) {
```

```
90      const size_t size = 50000;
91      auto input = NDatagen::GenerateNode(size);
92      auto output = input;
93      std::stable_sort(output.begin(), output.end(), CompTNode);
94
95      RadixSort(input, size);
96      ASSERT_EQ(input.size(), output.size());
97
98      for (size_t i = 0; i < size; ++i) {
99          EXPECT_EQ(input[i].key, output[i].key);
100         EXPECT_EQ(*input[i].value, *output[i].value);
101     }
102
103 }
104
105 TEST(Lab1Test, Banchmark) {
106     const size_t size = 1000000;
107     auto input1 = NDatagen::GenerateNode(size);
108     auto input2 = input1;
109
110     auto begin1 = std::chrono::high_resolution_clock::now();
111     RadixSort(input1, size);
112     auto end1 = std::chrono::high_resolution_clock::now();
113     auto time1 = std::chrono::duration_cast<std::chrono::milliseconds>(end1 - begin1).
            count();
114
115
116     auto begin2 = std::chrono::high_resolution_clock::now();
117     std::stable_sort(input2.begin(), input2.end(), CompTNode);
118     auto end2 = std::chrono::high_resolution_clock::now();
119     auto time2= std::chrono::duration_cast<std::chrono::milliseconds>(end2 - begin2).
            count();
120     std::cout << "RadixSort time: " << time1 << " ms\n";
121     std::cout << "std::stable_sort time: " << time2 << " ms\n";
122
123     EXPECT_GE(time2, time1);
124
125 }
```

```
[alex@fedora build]$ ctest -V
UpdateCTestConfiguration  from :/home/alex/mai-da-labs/build/DartConfiguration.tcl
UpdateCTestConfiguration  from :/home/alex/mai-da-labs/build/DartConfiguration.tcl
Test project /home/alex/mai-da-labs/build
Constructing a list of tests
Done constructing a list of tests
Updating test list for fixtures
Added 0 tests to meet fixture requirements
```

```
Checking test dependency graph...
Checking test dependency graph end
test 1
Start 1: lab1_test

1: Test command: /home/alex/mai-da-labs/build/tests/lab1_test
1: Working Directory: /home/alex/mai-da-labs/build/tests
1: Test timeout computed to be: 10000000
1: Running main() from /home/alex/mai-da-labs/build/_deps/googletest-src/googletest/s
1: [==========] Running 3 tests from 1 test suite.
1: [----------] Global test environment set-up.
1: [----------] 3 tests from Lab1Test
1: [ RUN      ] Lab1Test.StableTest
1: [       OK ] Lab1Test.StableTest (0 ms)
1: [ RUN      ] Lab1Test.CommonTest
1: [       OK ] Lab1Test.CommonTest (219 ms)
1: [ RUN      ] Lab1Test.Benchmark
1: RadixSort time: 886 ms
1: std::stable_sort time: 1185ms
1: [       OK ] Lab1Test.Benchmark (4128 ms)
1: [----------] 3 tests from Lab1Test (4348 ms total)
1:
1: [----------] Global test environment tear-down
1: [==========] 3 tests from 1 test suite ran. (4348 ms total)
1: [  PASSED  ] 3 tests.
1/1 Test #1: lab1_test ........................   Passed    4.37 sec

100% tests passed,0 tests failed out of 1

Total Test time (real) =   4.37 sec
```

Как видно, $RadixSort$ выиграл у $std :: stable\_sort$, так как сложность по времени $RadixSort$ равна $O(k(n+d))$, где n - количество объектов, k - количество разрядов у ключа объекта, d - размер разряда, а сложность $std :: stable\_sort$ равна $O(n*log(n))$, где n - количество объектов.

5 Выводы

В ходе выполнения лабораторной работы по курсу «Дискретный анализ» я изучил сортировки за линейное время, реализовал поразрядную сортировку по своему варианту. Столкнулся с ML (memory limit) и TL (time limit). Было принято решение хранить в объекте не целиком строку, а только указатель на нее, так как указатель занимает меньше памяти и копировать указатели быстрее.

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] *Цифровая сортировка — Вики-конспекты ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 16.12.2013).