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: И. П. Попов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

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: `MD5-суммы (32-разрядные шестнадцатеричные числа).`

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

1 Описание

Интернет-ресурс [4] 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 $TData$ (она состоит из массива $char[33]$ и $char*$ для хранения ключа и значения соответственно). Считываю данные и добавляю их в вектор, который в последствии передаю в функцию поразрядной сортировки $RadixSort$ (про считывание будет написано ниже). Данная функция использует в себе сортировку подсчётом, сортируя последовательно разряды моего 32-разрядного ключа от меньшего к большему. Сортировка подсчётом ($CountingSort$) реализована по стандартному алгоритму:

1. Создаем вспомогательный массив $count$ размером, равным максимально допустимому числу в сортировке, в данном случае – это число f (16), и инициализуем его нулями;

2. Инкрементируем в нем элементы, индексы которых равны рассматриваемым разрядам сортируемых чисел;

3. Запускаем префикс-функцию внутри массива $count$;

4. Идём справа налево по исходному вектору и сопоставляя элемент с числом вхождений этого элемента (указанным в массиве count) устанавливаем его на нужное место в результирующем векторе.

$CountingSort$ в свою очередь использует функцию $CharToNum$, которая переводит значения шестнадцатеричных чисел в их десятичное представление (для корректного заполнения массива count). Также перегружены операторы $>>$ и $=$ для удобства работы со структурой $TData$.

Возвращаясь к моменту считывания данных: считав ключ, программа выделяет место под значение и считывает его, далее с помощью realloc занятое на предыдущей итерации место освобождается и считанные и значение отправляются в вектор для сортировки.

```
1 #include <iostream>
2 #include <ctime>
3 #include <vector>
4 #include <string>
5 #include <cstring>
6
7
```

```cpp
const int NUMBER_OF_DIGIT = 32;

typedef unsigned long long ull;
typedef struct TData {
    char key[NUMBER_OF_DIGIT + 1];
    char* str;

    TData& operator= (const TData& tmp) {
        for (int i = 0; i < NUMBER_OF_DIGIT; i++)
            this->key[i] = tmp.key[i];
        str = tmp.str;
        return *this;
    }
} TData;


std::ostream& operator<< (std::ostream& out, const TData& td) {
    std::cout << td.key << "\t" << td.str << '\n';
    return out;
}


const int CONVERT_CONST_TO_HEX_SISTEM = 10;

int CharToNum(int inc_idx) {
    if ('a' <= inc_idx && inc_idx <= 'f') { //letter
        return inc_idx - 'a' + CONVERT_CONST_TO_HEX_SISTEM;
    }
    else if ('0' <= inc_idx && inc_idx <= '9') { //number
        return inc_idx - '0';
    }
    return 0;
}


const int SIZE_OF_COUNTING_MASSIVE = 16;

void CountingSort(std::vector<TData>& v_data, size_t digit_num) {
    std::vector<TData> res(v_data.size());
    ull count[SIZE_OF_COUNTING_MASSIVE] = { 0 };

    for (size_t i = 0; i < v_data.size(); i++) {
        size_t incremental_idx = v_data[i].key[digit_num];
        count[CharToNum(incremental_idx)]++;
    }

    for (int i = 1; i < SIZE_OF_COUNTING_MASSIVE; i++) {
        count[i] += count[i - 1];
    }
```

```
57
58      for (int i = v_data.size() - 1; i >= 0; i--) {
59          size_t idx = CharToNum(v_data[i].key[digit_num]);
60          res[count[idx] - 1] = v_data[i];
61          count[idx]--;
62      }
63
64      v_data = res;
65  }
66
67
68  void RadixSort(std::vector<TData>& v_data) {
69      for (int digit_num = NUMBER_OF_DIGIT - 1; digit_num >= 0; digit_num--) {
70          CountingSort(v_data, digit_num);
71      }
72  }
73
74
75  const int LENGTH_OF_STRING = 2048;
76
77  int main() {
78      std::ios::sync_with_stdio(false);
79      std::cin.tie(nullptr);
80      std:: cout.tie(nullptr);
81
82      TData tmp;
83      std::vector<TData> v_data;
84
85      while (scanf("%s", tmp.key) != EOF) {
86          tmp.str = new char[LENGTH_OF_STRING + 1];
87          scanf("%s", tmp.str);
88          tmp.str = (char *)realloc(tmp.str, sizeof(char) * (strlen(tmp.str) + 1));
89          v_data.push_back(std::move(tmp));
90      }
91
92
93      if (v_data.size()) {
94          RadixSort(v_data);
95      }
96
97      for (TData tmp : v_data) {
98          std::cout << tmp;
99      }
100
101     return 0;
102 }
```

3 Консоль

```
lunidep@lunidep-VirtualBox:~/Desktop/da_lab1$ g++ -pedantic -std=c++17 -Wall
-Werror da_lab1.cpp -o da_lab1
lunidep@lunidep-VirtualBox:~/Desktop/da_lab1$ cat test.t
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
lunidep@lunidep-VirtualBox:~/Desktop/da_lab1$ ./da_lab1 <test.t
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
```

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 10000 пар «ключ-значение», и их упорядочивание по возрастанию ключа.

```
lunidep@lunidep-VirtualBox:~/Desktop/da_lab1$ g++ -pedantic -std=c++17 -Wall
-Werror da_lab1.cpp -o da_lab1
lunidep@lunidep-VirtualBox:~/Desktop/da_lab1$ ./da_lab1 <tests/01.t >tmp
lunidep@lunidep-VirtualBox:~/Desktop/da_lab1$ cat tmp | grep "time"
Radix sort time: 81956us
STL Sort time: 15318us
```

Как видно, что поразрядная сортировка выиграла у STL, засчет сравнительо небольшого объема сортируемых данных, при увеличении этого объема со временем STL сортировка начнет показывать 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. n = 100000; time = 1,070s

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», мною были изучены и реализованы два алгоритма сортировки за линейное время – поразрядная сортировка и сортировка подсчётом.

Сортировка за линейное время наиболее эффективна при обработке небольшого количества данных. Сложность сортировок этого типа – O(d*n), где d – 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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

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

[4] *Поразрядная сортировка — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 14.03.2022).