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. С. Бычков
Преподаватель: Н. К. Макаров
Группа: М8О-201Б
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: даты в формате DD.MM.YYYY, например 1.1.1, 1.9.2009, 01.09.2009, 31.12.2009.

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

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки. Необходимо отсортировать даты вышеописанного вида. Для удобства каждую дату данного вида представим, как одно число следующей формулой: $day + 100 * month + 10.000 * year$, где day, $month$ и $year$ - день, месяц и год данной даты соответственно. Когда мы таким образом представили дату, у нас всегда последние две цифры отвечают за день, третья и четвертая цифра (считая справа) образуют месяц, а оставшиеся цифры образуют год. Теперь удобно будет применить порязрядную сортировку, для обычных чисел.

Основная идея поразрядной сортировки заключается в том, мы разбиваем ключ (по которому идёт сравнение) на несколько частей. И отдельно сортируем каждую из этих частей. Если мы будем идти от менее значимых признаков к более значимым, например от меньших разрядов чисел к большим, то такая сортировка будет называться LSD. После того, как отсортированы все части ключей, все исходные объекты будут так же отсортированы.

2 Исходный код

Первым делом необходимо реализовать наш ключ - дату. Для этого создан класс «Date», в котором поля - day, $month$, $year$ - день, месяц и год соответственно. А поле $date$ - дата в том формате, в котором она поступила на вход. Для даты реализовано 4 конструктора: $default$, $move-constructor$, $copy-constructor$, а так же конструктор от строки. Так же, реализован метод $parse\_string$, который по строке $date$ заполняет поля day, $month$, $year$. Так же, в этом классе реализованы операторы ввода/вывода, $copy-assigment\ operator$, $move-assigment\ operator$, а так же геттеры для получения приватных полей. Последний метод - $merge\_date$ осуществляет приведение даты из строки в число по вышеописанному алгоритму.

Теперь опишем структуру «Data», которая из себя представляет пару - «ключ-значение», где ключ - дата, значение - строка. Для этой структуры реализованы такие же конструкторы, что и класса «Date», реализованы операторы ввода/вывода, а так же, $copy-assigment\ operator$ и $move-assigment\ operator$.

В конце реализована сама «LSD» сортировка. В алгоритме мы будет сортировать дату, представив ее, как одно число. Сами эти числа-ключи, будем сортировать по цифрам в восьмеричной системе счисления, т.е. каждый раз будет брать по 8 бит двоичного представления числа. Дальнейший алгоритм таков:

1. Заводим массив - $counting\_sort$, в котором будет производить сортировку подсчётом, а так же подсчёт префиксных сумм этого массива.
2. Находим максимальный ключ, чтобы по нему определять, когда нам стоит прекратить сортировку.
3. Теперь, пока мы не перебрали все цифры каждого числа:
   (a) Выполняем сортировку подсчетом для i-го разряда, начиная с младшего.
   (b) Считаем префиксные суммы для подсчитанных цифр.
   (c) Проходимся по исходному массиву в обратом порядке. По текущей цифре каждого элемента и по префиксным суммам, определяем, куда нужно положить текущий пару «ключ-значение».
   (d) Зануляем массив $counting\_sort$.
   (e) Меняем исходным массив с буффером местами, обновляем номер разряда и обновляем сдвиг.
   (f) Повторяем пункты $a-e$, пока выполняется условие 3.
4. Массив отсортирован.

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <array>

void LSD(std::vector<Data>& vct) {
    std::array<size_t, 256> counting_sort = {};
    size_t n = vct.size();
    std::vector<Data> other(n);

    size_t max = vct[0].key.merge_date();
    size_t base = 255;
    size_t digit_number = 0;

    for (Data& el : vct) {
        max = el.key.merge_date() > max ? el.key.merge_date() : max;
    }

    /*
     1) Sort by counting by digit.
     2) Immediately calculate the prefix sum of this array.
     3) Write the result to another array.
    */

    /*
     (base <= max) || (base > max && max & base) ==
     A || ~A && B ==
     A || B =>
     base <= max || max & base
    */
    while ((base <= max) || (max & base)) {

        for (Data& el : vct) { // Sorting by counting
            size_t digit = (el.key.merge_date() & base) >> (8 * digit_number); //
                Specific number in the 256 system
            ++counting_sort[digit];
        }


        for (size_t i = 1; i < 256; ++i) { // Counting prefixes
            counting_sort[i] += counting_sort[i - 1];
        }

        for (ssize_t i = n - 1; i >= 0; --i) { // Sorting
            size_t digit = (vct[i].key.merge_date() & base) >> (8 * digit_number);
            size_t index = --counting_sort[digit];
            other[index] = std::move(vct[i]);
        }

```

```
49          for (size_t& el : counting_sort) { // We zero the array for counting sorting.
50              el = 0;
51          }
52
53          std::swap(vct, other);
54          base <<= 8;
55          ++digit_number;
56
57          if (digit_number == 8) { // We made 8 shifts.
58              break; // Thus, we exceeded the 8-byte number.
59          }
60      }
61  }
62
63
64  int main() {
65      std::ios::sync_with_stdio(false);
66      std::cin.tie(0);
67
68      std::vector<Data> data;
69
70      for (size_t i = 0;;++i) {
71          Data item;
72          if (std::cin >> item) {
73              data.push_back(std::move(item));
74          } else {
75              break;
76          }
77      }
78
79      if (data.size() == 0) {
80          return 0;
81      }
82
83      LSD(data);
84
85      for (size_t i = 0; i < data.size(); ++i) {
86          std::cout << data[i] << '\n';
87      }
88  }
```

Методы класса «Date» и структуры «Data».

main.c	
Date()	Конструктор по умолчанию
Date(Date&&)	Конструктор перемещения по умолчанию
Date(const Date& other)	Конструктор копирования

Date(const std::string& _date)	Конструктор от строки - даты. В этом конструкторе вызывается метод *parse_string*
Date& operator=(Date&&)	*move − assigment operator* по умолчанию
Date& operator=(Date& other)	*copy − assigment operator*
uint8_t get_day()	Геттер для получения значения поля *day*
uint8_t get_month()	Геттер для получения значения поля *month*
uint16_t get_year()	Геттер для получения значения поля *year*
size_t merge_date()	Метод чтобы представить дату в виде одного числа. Это происходит по формуле: $day + 100 * month + 10.000 * year$
void parse_string()	Приватный метод для парсинга даты в виде строки в три числа: день, месяц и год. В этом методе просто осуществляется проход по строке, и встретив точку анализируется то, что было до нее и арифметическими операциями получается одно из этих трёх чисел.
std::ostream& operator«(std::ostream& os, const Date& date)	Метод для вывода объекта
std::istream& operator»(std::istream& is, Date& date)	Метод для ввода объекта
Data()	Конструктор по умолчанию
Data(Data&&)	Конструктор перемещения по умолчанию
Data(const Data& other)	Конструктор копирования
Data(const Date& key, const std::string& value)	Конструктор от ключа - даты и значения - строки.
Data& operator=(Data&&)	*move − assigment operator* по умолчанию
Data& operator=(Data& other)	*copy − assigment operator*
std::ostream& operator«(std::ostream& os, const Data& data)	Метод для вывода объекта
std::istream& operator»(std::istream& is, Data& data)	Метод для ввода объекта

```cpp
class Date {

    friend std::ostream& operator<<(std::ostream& os, const Date& date);
    friend std::istream& operator>>(std::istream& is, Date& date);

private:
    uint8_t day = 1;
    uint8_t month = 1;
    uint16_t year = 1;
    std::string date;

    void parse_string();

public:
    Date() = default;
    Date(Date&&) = default;
    Date(const Date& other);
    Date(const std::string& _date);

    Date& operator=(Date&&) = default;
    Date& operator=(Date& other);

    uint8_t get_day();
    uint8_t get_month();
    uint16_t get_year();
    size_t merge_date();
};


struct Data {
    friend std::ostream& operator<<(std::ostream& os, const Data& data);
    friend std::istream& operator>>(std::istream& is, Data& data);

    Date key;
    std::string value;

    Data() = default;
    Data(Data&&) = default;
    Data(const Date& key, const std::string& value);
    Data(const Data& other);

    Data& operator=(Data& other);
    Data& operator=(Data&&) = default;
};
```

3 Консоль

```
mason@mint:~/Desktop/algorithms/DA/lab1$ g++ 4-2.cpp -Wall -Wpedantic -Wextra
-Werror -std=c++17
mason@mint:~/Desktop/algorithms/DA/lab1$ cat input
1.1.1   xGfxrxGGxrxMMMMfrrrG
01.02.2008      xGfxrxGGxrxMMMMfrrr
1.1.1   xGfxrxGGxrxMMMMfrr
01.02.2008      laklj
1.07.1005       xGfxrxGGxrxMMMMfrrrG
01.02.2007      aaaaa
12.3.4  xGfxrxGGxrxfffMMMMfrr
11.5.20 asdfasdfasdf
mason@mint:~/Desktop/algorithms/DA/lab1$ ./a.out <input
1.1.1   xGfxrxGGxrxMMMMfrrrG
1.1.1   xGfxrxGGxrxMMMMfrr
12.3.4  xGfxrxGGxrxfffMMMMfrr
11.5.20 asdfasdfasdf
1.07.1005       xGfxrxGGxrxMMMMfrrrG
01.02.2007      aaaaa
01.02.2008      xGfxrxGGxrxMMMMfrrr
01.02.2008      laklj
mason@mint:~/Desktop/algorithms/DA/lab1$
```

4 Тест производительности

Тест производительности представляет из себя следующее: сортировка массива из одного миллиона пар «ключ-значение». Все пары сгенерированы случайным образом, ключ - дата в формате, соответствующем заданию, а значение - случайная строка длины от 1 до 2048, состоящая из строчных и заглавных букв латинского алфавита.

```
$>g++ -Wall -Werror -Wextra -Wpedantic benchmark.cpp -std=c++17
$>./a.out <input
$>grep -e "Time:" LSD.txt
Time: 286ms
$>grep -e "Time:" stable_sort.txt
Time: 1034ms
$>diff LSD.txt stable_sort.txt
1000001c1000001
<Time: 286ms
---
>Time: 1034ms
```

Как видно, поразрядная сортировка выиграла у $std :: stable\_sort$. Это происходит из-за того, что $std :: stable\_sort$ реализуется на алгоритме $merge\ sort$, который работает за $\mathcal{O}(n \log n)$, в то время, как реализованная мной LSD сортировка работает за время $\mathcal{O}(nk)$, где k - максимальное число разрядов в числе. Т.к. в моей сортировке все числа рассматриваются в 256-ичной системе счисления, k очень мало. Например, для даты 26.02.2024 k будет чуть больше трёх.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я узнал, что есть сортировки, работающие за линейное время на произвольных данных. Раньше я знал только о «сортировке-подсчётом», которая, конечно, работает за линейное время, но применять ее имеет смысл только в том случае, когда размер входных данных близок к разнице между максимальным и минимальными элементами, т.е. когда количество уникальных значений не велико. Однако, для меня стало открытием, что существуют сортировки, позволяющие сортировать любое количество сколь угодно различных чисел (и не только), главное, чтобы каждый элемент можно было разделить на части.

Данные знания пригодится мне в спортивном программирование, где каждая миллисекунда важна. Я написал небольшой *benchmark* и выяснил, что реализованная мной LSD сортировка на массиве в 10^8 элементов работает аж в пять раз быстрее, чем встроенная в $C++$ «std::sort», являющаяся сортировкой Хоара с кучей оптимизацей!!!

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] *Порязрядная сортировка - ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 26.02.2024).