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: В. И. Пупкин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2020

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 шестнадцатиричные числа).`

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

1 Описание

Требуется написать реализацию алгоритма поразрядной сонртировки. Как сказано в [1]: «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 для разряда использую устойчивую версию сортировки подсчетом[2].

Устойчивая сортировка подсчетом осуществляется с помошью свух вспомогательных массивов: один для счетчика, другой для записи отсортированного результата. Для удобства обозначу: A - исходный массив, B - результирующий массив, C - массив для подсчета.

Сначала заполняю C нулями. Затем для каждого $A[i]$ увеличиваю $C[A[i]]$ на единицу. Суммирую каждый $C[i]C[i-1]$, кроме $C[0]$. Последним читаю входной массив с конца, записываю в $B[C[A[i]]]$ $A[i]$ и уменьшаю $C[A[i]]$ на единицу[2].

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создаду новую структуру *item*, в которой буду хранить ключ (массив из восьми *integer*) и указатель на значение (массив из 2050 элемнтов типа *char*). Сортирую по указателю, так как копирование строки занимает слишком много времени. Так как количество пар во входных данных не указано, считываю данные в цикле *while*. Для хранения пар использую $TVector < item >$, который динамически выделяет память под новые пары и аналогичный TVector<char*> для хранения строк. Из воода считываю ключ и значение как строки, затем перевожу строчный ключ в массив *integer* и сохраняю в $TVector < item >$, строку сохраняю без преобразований. Так как $TVector$ перевыдеяет память, если в уже выделенной не хватает места для нового элемента, то все ранее сохраненные указатели становтся невалидными. Поэтому указаетльи для *item* записыаю после окончания ввода в отдельном цикле.

Сортирую LSD-сортировкой. Фактически за один разряд беру четырехзначное шестнадцатиричное число (массив из восьми *integer*). Иными словани перевожу 32-разрядные шестнадцатиричные числа в 8-разрядные 65 536-ричные. Делаю это для ускорения сортировки, так как ограниечение по памяти позволяет.

Выполняю сортироку в цикле, от младщего разряда к старшему. Для сортировки подсчетом выдеяю массив из 65 536 элементов, обнуляю его. Подсчитываю сколько каждая пара встречалась во входных данных и записываю по соответствующим индексам. Считаю префексные суммы, записываю в массив для посдеча. Теперь по индексу указано, сколько есть элементов, которвые меньше или равны элементу в индексе. Завожу указатель под результирующий массив *item*-ов и выделяю для него память. В обратом порядке прохожусь по исходному $TVector < item >$ и записываю в результирующий массив элемнты, глядя на массив для подсчета. Уменьшаю на 1 значение по индексу для текущего *item*. Выполняю то же самое для остальных разрядов.

Вывожу ответ.

```
//
// vector.h
//

#ifndef LAB1_TVector_H
#define LAB1_TVector_H
template <class T>
class TVector{
    private:
        long long TVectorSize;
        long long TVectorCapacity;
        T* Data;
    public:
        void Assign(const T elem);
        TVector();
        TVector(const long long n);
        long long Size();
        void PushBack(const T &elem);
        T& operator[] (const long long iterator);
        ~TVector();
};
#endif
```

```
//
// main.cpp
//

#include <string>
#include <iostream>
#include <iomanip>
#include <cstring>
#include "vector.h"
#include "vector.cpp"

const unsigned int KEY_SIZE = 8;
const unsigned int KEY_RADIX_SIZE = 4;
const unsigned int VALUE_SIZE = 2050;
const unsigned int MAX_KEY_VALUE = 0xffff;

struct item{
    int Key[KEY_SIZE];
    char** Value;
};

int main(){
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    std::ios_base::sync_with_stdio(false);
    char strKey[33];
    TVector<item> sortArray;
```

```
28      TVector<char*> valueData;
29      char currentValue[VALUE_SIZE];
30      item currentPair;
31      while (std::cin >> strKey >> currentValue){
32          for (int & i : currentPair.Key){
33              i = 0;
34          }
35          for (int i = KEY_SIZE - 1; i >= 0; --i){
36              int radixMultiply = 1;
37              for (int j = KEY_RADIX_SIZE - 1; j >= 0; --j){
38                  if (strKey[i * KEY_RADIX_SIZE + j] >= '0' && strKey[i * KEY_RADIX_SIZE +
                        j] <= '9'){
39                      currentPair.Key[i] += (strKey[i * KEY_RADIX_SIZE + j] - '0') *
                            radixMultiply;
40                  }
41                  else if (strKey[i * KEY_RADIX_SIZE + j] >= 'a' && strKey[i *
                        KEY_RADIX_SIZE + j] <= 'f'){
42                      currentPair.Key[i] += (strKey[i * KEY_RADIX_SIZE + j] - 'a' + 10) *
                            radixMultiply;
43                  }
44                  radixMultiply *= 16;
45              }
46          }
47          char* newValue = new char[VALUE_SIZE];
48          std::memcpy(newValue, currentValue, sizeof(char)*VALUE_SIZE);
49          valueData.PushBack(newValue);
50          sortArray.PushBack(currentPair);
51      }
52      for (long long i = 0; i < sortArray.Size(); ++i){
53          sortArray[i].Value = &valueData[i];
54      }
55
56      for (int radixNumber= KEY_SIZE - 1; radixNumber >= 0; --radixNumber){
57          long long countingArray[MAX_KEY_VALUE + 1];
58          for (unsigned int i = 0; i <= MAX_KEY_VALUE; ++i){
59              countingArray[i] = 0;
60          }
61          for (long long i = 0; i < sortArray.Size(); ++i){
62              countingArray[sortArray[i].Key[radixNumber]] += 1;
63          }
64          for (unsigned int i = 1; i <= MAX_KEY_VALUE; ++i){
65              countingArray[i] += countingArray[i - 1];
66          }
67          item* result = new item[sortArray.Size()];
68          for (long long i = sortArray.Size() - 1; i >= 0; --i){
69              result[countingArray[sortArray[i].Key[radixNumber]] - 1] = sortArray[i];
70              countingArray[sortArray[i].Key[radixNumber]] = countingArray[sortArray[i].
                    Key[radixNumber]] - 1;
71          }
```

```
72          for (long long i = 0; i < sortArray.Size(); ++i){
73              sortArray[i] = result[i];
74          }
75          delete [] result;
76      }
77
78      for (long long i = 0; i < sortArray.Size(); ++i){
79          for (int j : sortArray[i].Key){
80              std::cout << std::hex << std::setw(4) << std::setfill('0') << j;
81          }
82          std::cout << " " << *sortArray[i].Value << "\n";
83      }
84      for (long long i = 0; i < valueData.Size(); ++i){
85          delete [] valueData[i];
86      }
87      return 0;
88  }
```

vector.cpp	
TVector()	Конструктор TVector, обнуляет размер, емкость и указатель на данные.
Size()	Метод TVector, возвращает размер.
PushBack(const T &elem)	Метод TVector, добавляет новый элемент шаблонного класса T в конец TVector.
operator[](const long long iterator)	Перегруженный оператор []. Возвращает данные по итераторру.

3 Консоль

```
protaxy@protaxY:~/DA/da_lb1$ make
g++ -O3 -std=c++14 -o solution main.cpp vector.cpp vector.h
protaxy@protaxY:~/DA/da_lb1$ cat test_input.txt
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
1dc575e626f7c8b417f2020770273d MIFWzvYrPBZAXVY
9af96543c4481d9f9baeb19ad282f0b4 UJ
94e8780e66d797a4df042f131450dbd3 bceuHWDhH
621521cddd0e8ec18bd0f1fbead89d68 PMFknR
9d9c4419436223e1e1817d7796067a61 JCUrPPTsniODesytPNfUacELQZi
8ce57ef3cb0db9ea7bee536a82e31be2 TKSYZpVbVTyMx
protaxy@protaxY:~/DA/da_lb1$ ./solution <test_input.txt
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrrrG
00000000000000000000000000000000 xGfxrxGGxrxMMMMfrr
1dc575e626f7c8b417f2020770273d MIFWzvYrPBZAXVY
621521cddd0e8ec18bd0f1fbead89d68 PMFknR
8ce57ef3cb0db9ea7bee536a82e31be2 TKSYZpVbVTyMx
94e8780e66d797a4df042f131450dbd3 bceuHWDhH
9af96543c4481d9f9baeb19ad282f0b4 UJ
9d9c4419436223e1e1817d7796067a61 JCUrPPTsniODesytPNfUacELQZi
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfrrr
ffffffffffffffffffffffffffffffff xGfxrxGGxrxMMMMfr
```

4 Тест производительности

Время на ввод и посторение структур данных не учитывается, замеряется только время, затраченное на сортировку. Количество пар «ключ-значение» для каждого файла равно десять в степени номер теста. Например, $02.txt$ содержит сто пар, а $06.txt$ миллион.

```
protaxy@protaxY:~/DA/da_lb1$g++ -std=c++14 benchmark.cpp -o benchmark
protaxy@protaxY:~/DA/da_lb1$ ./bechmarck <tests/01.txt
Radix sort:1550 microseconds
std::stable_sort:6 microseconds
protaxy@protaxY:~/DA/da_lb1$ ./benchmark <tests/02.txt
Radix sort:1566 microseconds
std::stable_sort:46 microseconds
protaxy@protaxY:~/DA/da_lb1$ ./benchmark <tests/03.txt
Radix sort:1930 microseconds
std::stable_sort:704 microseconds
protaxy@protaxY:~/DA/da_lb1$ ./benchmark <tests/04.txt
Radix sort:5187 microseconds
std::stable_sort:5689 microseconds
protaxy@protaxY:~/DA/da_lb1$ ./benchmark <tests/05.txt
Radix sort:26987 microseconds
std::stable_sort:79901 microseconds
protaxy@protaxY:~/DA/da_lb1$ ./benchmark <tests/06.txt
Radix sort:370515 microseconds
std::stable_sort:1185799 microseconds
```

Глядя на результаты, видно, что $std :: stable\_sort$ выигрывает больше всего на самых мальеньких тестах, а $Radixsort$ на самых больших. Сложность $std :: stable\_sort$ $O(n * log\ n)$, а сложность $Radixsort$ $O(d * n)$, где d - количество разрядов в числе. Так как логарифм возрастающая функция, переломный момент наступает, когда $log\ n$ становится больше, чем постоянная d.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать шаблонный класс с динамическим выделением памяти на примере реализации класса $TVector$. Реализовал поразрядную сортировку и стабильную сортировку подсчетом. Узнал, что оператор копирования для массива $char$-ов работает очень медленно, соответсвенно выгодно сортировать по указателям. Хоть на больших тестах(10^5 и больше) соритровка по разрядам быстрее чем $std :: stable\_sort$, но пространсвенная сложность $O(n + m)$ (где n - рамер входного массива, а m - размер разряда) дает о себе знать.

Список литературы

[1] *Поразрядная сортировка — Википедия.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 03.10.2020).

[2] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 03.10.2020).

[3] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 3-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.)) стр. 220 - 229 (дата обращения: 03.10.2020).