

**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: Н. П. Ежов
Преподаватель: Н. С. Капралов
Группа: М8О-204Б
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 сортировки. 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 помощью двух 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 создаю новую структуру *TItem*, в которой буду хранить ключ (массив из восьми *integer*) и указатель на значение (массив из 2049 элементов 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 < TItem >*, который динамически выделяет память под новые пары и аналогичный *TVector<char*>* для хранения строк. Из ввода считываю ключ и значение как строки, затем перевожу строчный ключ в массив *integer* и сохраняю в *TVector < TItem >*, строку сохраняю без преобразований. Так как *TVector* перевыделяет память, если в уже выделенной не хватает места для нового элемента, то все ранее сохраненные указатели становятся невалидными. Поэтому указатели для *TItem* записываю после окончания ввода в отдельном цикле. Сортирую LSD-сортировкой. Фактически за один разряд беру четырехзначное шестнадцатиричное число (массив из восьми *integer*). Иными словами перевожу 32-разрядные шестнадцатиричные числа в 8-разрядные 65 536-ричные. Делаю это для ускорения сортировки, так как ограничение по памяти позволяет.

Выполняю сортировку в цикле, от младшего разряда к старшему. Для сортировки подсчётом выделяю массив из 65 536 элементов, обнуляю его. Подсчитываю сколько каждая пара встречалась во входных данных и записываю по соответствующим индексам. Считаю префиксные суммы, записываю в массив для подсчёта. Теперь по индексу указано, сколько есть элементов, которые меньше или равны элементу в индексе. Завожу указатель под результирующий массив *TItem*-ов и выделяю для него память. В обратном порядке прохожусь по исходному *TVector < TItem >* и записываю в результирующий массив элементы, глядя на массив для подсчёта. Уменьшаю на 1 значение по индексу для текущего *TItem*. Выполняю то же самое для остальных разрядов.

Вывожу ответ.

```

1 //
2 // vector.hpp
3 //
4
5 #ifndef TVector_HPP
6 #define TVector_HPP
7 #include <cstdio>
8 #include <cstring>
9 const long long KEY_SIZE = 8;
10
11 namespace NMyStd{
12     struct TItem{
13         int Key[KEY_SIZE];
14         char **Value;
15     };
16
17     template <class T>
18     class TVector{
19     private:
20         long long TVectorSize;
21         long long TVectorCapacity;
22         T* Data;
23     public:
24         void Assign(const T elem);
25         void Assign(const long long n, const T elem);
26         TVector();
27         TVector(const long long n);
28         TVector(const long long n, T elem);
29         long long Size();
30         void PushBack(const T &elem);
31         void Pop();
32         bool Empty();
33         T& operator[] (const long long iterator);
34         ~TVector();
35     };
36     template <class T>
37     void TVector<T>::Assign(const T elem){
38         for (long long i = 0; i < TVectorSize; ++i){
39             Data[i] = elem;
40         }
41     }
42
43     template <class T>
44     void TVector<T>::Assign(const long long n, const T elem){
45         for (long long i = 0; i < n; ++i){
46             Data[i] = elem;
47         }
48     }
49     template <class T>

```

```

50     TVector<T>::TVector(){
51         TVectorSize = 0;
52         TVectorCapacity = 0;
53         Data = nullptr;
54     }
55     template <class T>
56     TVector<T>::TVector(const long long n){
57         TVectorSize = n;
58         TVectorCapacity = n;
59         Data = new T[TVectorCapacity];
60         Assign(n, T());
61     }
62     template <class T>
63     TVector<T>::TVector(const long long n, T elem):TVector(n){
64         Assign(elem);
65     }
66     template <class T>
67     long long TVector<T>::Size(){
68         return TVectorSize;
69     }
70     template <class T>
71     void TVector<T>::PushBack(const T &elem){
72         if (TVectorCapacity == 0){
73             TVectorCapacity = 1;
74             TVectorSize = 0;
75             Data = new T[TVectorCapacity];
76         }
77         else if (TVectorCapacity == TVectorSize){
78             TVectorCapacity *= 2;
79             T* newData = new T[TVectorCapacity];
80             for (long long i = 0; i < TVectorSize; ++i){
81                 newData[i] = Data[i];
82             }
83             delete [] Data;
84             Data = newData;
85         }
86         TVectorSize += 1;
87         Data[TVectorSize - 1] = elem;
88     }
89     template <class T>
90     void TVector<T>::Pop(){
91         TVectorSize = 0;
92         TVectorCapacity = 0;
93         delete [] Data;
94     }
95     template <class T>
96     bool TVector<T>::Empty(){
97         return TVectorSize == 0;
98     }

```

```

99     template <class T>
100     T& TVector<T>::operator[] (const long long iterator){
101         return Data[iterator];
102     }
103     template <class T>
104     TVector<T>::~TVector(){
105         delete [] Data;
106     }
107 }
108 #endif

1  //
2  // main.cpp
3  //
4  #include "vector.hpp"
5  #include <iostream>
6  #include <iomanip>
7
8
9  const int KEY_BIT_SIZE = 4;
10 const int VALUE_SIZE = 2049;
11 const unsigned int MAX_KEY = 0xffff;
12 const int STR_KEY_SIZE = 33;
13 const int HEX_PRECISION = 4;
14 const char ZERO_CHAR = '0';
15 const int ZERO = 0;
16 const int ONE = 1;
17 const int HEX_MULTIPLY = 16;
18 const char A_CHAR = 'a';
19 const char F_CHAR = 'f';
20 const char NINE_CHAR = '9';
21
22 void CountingSort(NMyStd::TVector<NMyStd::TItem> &data, int bit) {
23     NMyStd::TVector<long long> countArray(MAX_KEY + ONE, ZERO);
24     long long size = data.Size();
25     for (long long i = 0; i < size; ++i) {
26         ++countArray[data[i].Key[bit]];
27     }
28     for (unsigned int i = 1; i <= MAX_KEY; ++i) {
29         countArray[i] += countArray[i - ONE];
30     }
31     NMyStd::TItem *result = new NMyStd::TItem[size];
32     for (long long i = size - ONE; i >= 0; --i) {
33         result[countArray[data[i].Key[bit]] - ONE] = data[i];
34         --countArray[data[i].Key[bit]];
35     }
36     for (long long i = 0; i < size; ++i) {
37         data[i] = result[i];
38     }
39     delete [] result;

```

```

40 }
41
42 void BitwiseSort(NMyStd::TVector<NMyStd::TItem> &data) {
43     for (int i = KEY_SIZE - ONE; i >= 0; --i) {
44         CountingSort(data, i);
45     }
46 }
47
48 int main() {
49     std::cin.tie(nullptr);
50     std::cout.tie(nullptr);
51     std::ios_base::sync_with_stdio(false);
52     NMyStd::TVector<NMyStd::TItem> data;
53     char strKey [STR_KEY_SIZE];
54     NMyStd::TItem cur;
55     cur.Value = nullptr;
56
57     char bufInput [VALUE_SIZE];
58     NMyStd::TVector<char*> valueData;
59
60     while (std::cin >> strKey >> bufInput) {
61         for (int & i : cur.Key) {
62             i = ZERO;
63         }
64         for (int i = KEY_SIZE - ONE; i >= 0; --i) {
65             int hexMultiply = ONE;
66             for (int j = KEY_BIT_SIZE - ONE; j >= 0; --j) {
67                 if (strKey[i * KEY_BIT_SIZE + j] >= ZERO_CHAR && strKey[i * KEY_BIT_SIZE
68                     + j] <= NINE_CHAR) {
69                     cur.Key[i] += (strKey[i * KEY_BIT_SIZE + j] - ZERO_CHAR) *
69                         hexMultiply;
70                 }
71                 else if (strKey[i * KEY_BIT_SIZE + j] >= A_CHAR && strKey[i *
72                     KEY_BIT_SIZE + j] <= F_CHAR) {
73                     cur.Key[i] += (strKey[i * KEY_BIT_SIZE + j] - A_CHAR + 10) *
74                         hexMultiply;
75                 }
76                 hexMultiply *= HEX_MULTIPLY;
77             }
78         }
79         char *curValue = new char[VALUE_SIZE];
80         std::memcpy(curValue, bufInput, sizeof(char)*VALUE_SIZE);
81         data.PushBack(cur);
82         valueData.PushBack(curValue);
83     }
84     for (int i = 0; i < valueData.Size(); ++i) {
85         data[i].Value = &valueData[i];
86     }
87     BitwiseSort(data);

```



```

85     for (int i = 0; i < data.Size(); ++i) {
86         for (int j = 0; j < KEY_SIZE; ++j) {
87             std::cout << std::hex << std::setw(HEX_PRECISION) << std::setfill(ZERO_CHAR
88                 ) << data[i].Key[j];
89         }
90         std::cout << " " << *data[i].Value << "\n";
91     }
92     for (int i = 0; i < valueData.Size(); ++i) {
93         delete[] valueData[i];
94     }
95     return 0;
96 }

```

vector.hpp	
TVector()	Конструктор TVector, обнуляет размер, емкость и указатель на данные.
TVector(const long long n, T elem);	Конструктор TVector, задаёт вектору размер и ёмкость n, и забивает данные эл-ом elem.
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 итератору.
Assign(const long long n, const T elem)	Функция, которая задаёт вектору размер и ёмкость n, и забивает данные эл-ом elem

3 Консоль

```
(base) nikita@nikita-desktop:~/Diskran/lab1$ make
g++ -O3 -std=c++14 -o solution main.cpp vector.cpp vector.h
(base) nikita@nikita-desktop:~/Diskran/lab1$ cat test_input.txt
a5db3d43b37a95cd00618a7ac3112f7e LQFZzcjKUIgaNHdxMhDRzSojQdKKdJRxqnt0
d40c0348390bdb0e0fee9348cc8d2e67 vVNAzruNkZUPUUqHcmfWgkwdOGTSJeaAINZ
364590e5c89b6b1c54231793d261a3b2 wMmpTNRfxIrkciKtSkSdLYabpVgehC
734bf391f564bd5aff79a1fefeb358b7 m
e31adffb1b4418881731733600387d82 vJCqcbdkEeVJLWwRbPTRWk
caf836a9f342e48deb43e8215b23b177 FbGyVbPqdyv1QoJoqmvdiaCboIkNOILfld0APtjxbzoVUxFKSOvpp
5a3378d5ca2706bffab672e07894212b QKutCZfccVqEZORVvVxPICHQYA0emh
0472d854d83d11c287ec7d9eff9b8146 CvqgXrzHLEWFRIORDYhvJH
c41d2af1b376b9fc1f95fce356012712 SYhTAeNdhtZlTAWOreQeMNPAGgFSYNRMpoldNrUDsEAIR
8a18a352c07e3af6411007385bffd0ed ibdyiGQrGFQbLZngBwpsNoMiUrIGvkQwHwYGwVSWZ
(base) nikita@nikita-desktop:~/Diskran/lab1$ ./solution <test_input.txt
0472d854d83d11c287ec7d9eff9b8146 CvqgXrzHLEWFRIORDYhvJH
364590e5c89b6b1c54231793d261a3b2 wMmpTNRfxIrkciKtSkSdLYabpVgehC
5a3378d5ca2706bffab672e07894212b QKutCZfccVqEZORVvVxPICHQYA0emh
734bf391f564bd5aff79a1fefeb358b7 m
8a18a352c07e3af6411007385bffd0ed ibdyiGQrGFQbLZngBwpsNoMiUrIGvkQwHwYGwVSWZ
a5db3d43b37a95cd00618a7ac3112f7e LQFZzcjKUIgaNHdxMhDRzSojQdKKdJRxqnt0
c41d2af1b376b9fc1f95fce356012712 SYhTAeNdhtZlTAWOreQeMNPAGgFSYNRMpoldNrUDsEAIR
caf836a9f342e48deb43e8215b23b177 FbGyVbPqdyv1QoJoqmvdiaCboIkNOILfld0APtjxbzoVUxFKSOvpp
d40c0348390bdb0e0fee9348cc8d2e67 vVNAzruNkZUPUUqHcmfWgkwdOGTSJeaAINZ
e31adffb1b4418881731733600387d82 vJCqcbdkEeVJLWwRbPTRWk
```

4 Тест производительности

Время на ввод и построение структур данных не учитывается, замеряется только время, затраченное на сортировку. Количество пар «ключ-значение» для каждого файла равно десять в степени номер теста минус один. Например, *02.t* содержит десять пар, а *07.t* миллион.

```
(base) nikita@nikita-desktop:~/Diskran/lab1$ ./benchmark <tests/04.t >04.a
custom bitwise sort 6 ms
stable sort from std 0 ms
(base) nikita@nikita-desktop:~/Diskran/lab1$ ./benchmark <tests/05.t >05.a
custom bitwise sort 9 ms
stable sort from std 6 ms
(base) nikita@nikita-desktop:~/Diskran/lab1$ ./benchmark <tests/06.t >06.a
custom bitwise sort 37 ms
stable sort from std 168 ms
(base) nikita@nikita-desktop:~/Diskran/lab1$ ./benchmark <tests/07.t >07.a
custom bitwise sort 478 ms
stable sort from std 2371 ms
```

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

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, соответственно выгодно сортировать по указателям. Хотя на больших тестах (10^5 и больше) сортировка по разрядам быстрее чем *std :: stable_sort*, но пространственная сложность $O(n + m)$ (где n - размер входного массива, а m - размер разряда) дает о себе знать. Также из-за сложности $O(n * m)$ поразрядная сортировка очень медленно работает для чисел, где много разрядов, из-за чего она может быть медленнее *std :: stable_sort*, поэтому конкретно в этом варианте задачи целесообразнее разделить 32-ричное число на 8 разрядов по 4 числа

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).