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: Э. Л. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-307Б
Дата:
Оценка:
Подпись:

Москва, 2021

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: 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. Поразрядная сортировка по сути своей представляет из себя поэтапную сортировку подсчетом, которая по очереди применяется к каждому из разрядов, начиная с самого младшего и заканчивая самым старшим.

Для сортировки ключей, являющихся автомобильными номерами типа «A 999 AA», мне показалось наиболее естественным разбиение на следующие разряды:

- Первая буква номера;
- Значение числа в номере автомобиля;
- Две последние буквы номера.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру TKVPair, в которой будем хранить ключ и значение. Так как я не знаю заранее количество пар, я реализую класс TExpandingArray, то есть расширяющийся массив. Функция RadixSort вызывает для каждого элемента функцию DigitCountingSort.

```
#include <iostream>
#include <iomanip>
#include <time.h>
using namespace std;

const int VALUE_LENGTH = 65;
const int NUMBER_OF_LETTER = 26;
const int NUMBER_OF_INT_KEY = 1000;


struct TKVPair{
    char Key1;
    short Key2;
    char Key3[3];
    char Value [VALUE_LENGTH];

 TKVPair(){
        Key1 = '\0';
        Key2 = 0;
        Key3[0] = '\0';
        Value[0] = '\0';
    }

    friend ostream& operator<< (ostream &out, const TKVPair &kv);
    friend istream& operator>> (istream &in, TKVPair &kv);
};

ostream& operator<< (ostream &out, const TKVPair &kv)
{
    out << kv.Key1 << ' ' << setfill('0') << setw(3) << kv.Key2 << ' ' << kv.Key3 << ' ' << kv.Value;
    return out;
}

istream& operator>> (istream &in, TKVPair &kv)
{
    in >> kv.Key1;
    in >> kv.Key2;
    in >> kv.Key3;
    in >> kv.Value;
    return in;
}

struct TExpandingArray{
    struct TKVPair * Exar;
    unsigned long long Length;
    int Capacity;

    TExpandingArray(){
        Length = 0;
        Capacity = 1;
        Exar = new struct TKVPair[1];
    }

    ~TExpandingArray(){
        delete [] Exar;
```

```
    }

    void InsertElement (struct TKVPair new_elem) {
        if(Capacity == Length){
            struct TKVPair * tmp;
            tmp = new struct TKVPair[Capacity * 2];
            for (int i = 0; i < Length; ++i){
                tmp[i] = Exar[i];
            }
            delete [] Exar;
            Exar = tmp;
            Capacity *= 2;
        }
        Exar[Length] = new_elem;
        Length++;
    }

    struct TKVPair & operator [](unsigned long long index){
        if (index >= Length){
            throw out_of_range("Vector outside of range");
        } else {
            return Exar[index];
        }
    }
};

void DigitCountingSort(struct TExpandingArray *dataKeeper, int digitNumber){
    int * counters;
    struct TKVPair * data = new struct TKVPair[dataKeeper->Length];
    int key_to_int;
    if(digitNumber == 1){
        counters = new int[NUMBER_OF_LETTER];
        for(int i = 0; i<NUMBER_OF_LETTER; ++i){
            counters[i] = 0;
        }
        for(int i = 0; i < dataKeeper->Length; ++i){
            counters[(*dataKeeper)[i].Key1-VALUE_LENGTH]++;
        }
        for(int i = 1; i < NUMBER_OF_LETTER; ++i){
            counters[i] += counters[i-1];
        }
        for(int i = dataKeeper->Length-1; i >= 0; --i){
            data[counters[(*dataKeeper)[i].Key1 - VALUE_LENGTH] - 1] = (*dataKeeper)[i];
            counters[(*dataKeeper)[i].Key1 - VALUE_LENGTH]--;
        }
        delete [] counters;
    }
    else if(digitNumber == 2){
        counters = new int [NUMBER_OF_INT_KEY];
        for(int i = 0; i < NUMBER_OF_INT_KEY; ++i){
            counters[i] = 0;
        }
        for(int i = 0; i < dataKeeper->Length; ++i){
            counters[(*dataKeeper)[i].Key2]++;
        }
        for(int i=1; i<NUMBER_OF_INT_KEY; ++i){
            counters[i] += counters[i-1];
        }
        for(int i = dataKeeper->Length-1; i >= 0; --i){
            data[counters[(*dataKeeper)[i].Key2] - 1] = (*dataKeeper)[i];
            counters[(*dataKeeper)[i].Key2]--;
        }
```

```
118         delete [] counters;
119     }
120     else if(digitNumber == 3){
121         counters = new int [NUMBER_OF_LETTER*NUMBER_OF_LETTER];
122         for(int i = 0; i < NUMBER_OF_LETTER*NUMBER_OF_LETTER; ++i){
123             counters[i] = 0;
124         }
125         for(int i=0; i<dataKeeper->Length; ++i){
126             key_to_int = ((*dataKeeper)[i].Key3[0] - VALUE_LENGTH) * NUMBER_OF_LETTER + ((*dataKeeper)[i].
                    Key3[1] - VALUE_LENGTH);
127             counters[key_to_int]++;
128         }
129         for(int i = 1; i < NUMBER_OF_LETTER*NUMBER_OF_LETTER; ++i){
130             counters[i] += counters[i-1];
131         }
132         for(int i = dataKeeper->Length-1; i >= 0; --i){
133             key_to_int = ((*dataKeeper)[i].Key3[0] - VALUE_LENGTH) * NUMBER_OF_LETTER + ((*dataKeeper)[i].
                    Key3[1] - VALUE_LENGTH);
134             data[counters[key_to_int] - 1]=(*dataKeeper)[i];
135             counters[key_to_int]--;
136         }
137         delete [] counters;
138     }
139     delete [] dataKeeper->Exar;
140     dataKeeper->Exar = data;
141 }
142 
143 void RadixSort(struct TExpandingArray *dataKeeper){
144     for(int i = 3; i > 0; --i){
145         DigitCountingSort(dataKeeper, i);
146     }
147 }
148 
149 int main(){
150 
151     ios::sync_with_stdio(false);
152 
153     struct TKVPair temper;
154     struct TExpandingArray dataKeeper;
155 
156     while(cin >> temper){
157         dataKeeper.InsertElement(temper);
158     }
159     RadixSort(&dataKeeper);
160     cout<<'\n';
161     for (int i = 0; i < dataKeeper.Length; ++i){
162         cout << dataKeeper[i] << endl;
163     }
```

3 Консоль

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ ./lab1
A 000 AA  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
Z 999 ZZ  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
A 000 AA  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
Z 999 ZZ  n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na

A 000 AA n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
A 000 AA n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
Z 999 ZZ n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
Z 999 ZZ n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na
```

4 Тест производительности

Для теста произваодительности была написана программа ортировки с использованием стандартных контейнеров pair, string, vector и стандартного алгоритма stable_sort - stlreal.cpp и программа написанная согласно варианту, но измененная для подавления вывода - lab1.cpp. Так же для генерации тестов был использован скрипт gen.py на языке Python.

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ python3 gen.py 10000 > test
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ time ./lab1 < test


real    0m0.025s
user    0m0.000s
sys     0m0.010s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ time ./stlreal < test

real    0m0.032s
user    0m0.014s
sys     0m0.000s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ python3 gen.py 100000 > test
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ time ./lab1 < test


real    0m0.189s
user    0m0.056s
sys     0m0.019s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ time ./stlreal < test

real    0m0.237s
user    0m0.112s
sys     0m0.014s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ python3 gen.py 1000000 > test
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ time ./lab1 < test


real    0m1.686s
user    0m0.485s
sys     0m0.137s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab1$ time ./stlreal < test

real    0m2.507s
user    0m1.229s
sys     0m0.188s
```

Очевидно поразрядная сортировка работает быстрее стандартной в среднем в 1,33 раза на тестах с 10000, 100000 и 1000000 парк «Ключ-Значение».

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я реализовал поразрядную сортировку, которая имеет линейную сложность. Приобретенные в ходе данной работы навыки и знания можно применить для реализации сортировки элементов, которые можно естественно разбить на одинаковое количество разрядов, которые были бы ограничены сверху и снизу.