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: П. А. Мохляков
Преподаватель: Н. С. Капралов
Группа: М8О-208Б

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: 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

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.

2 Исходный код

Для начала напишем шаблонный класс вектора, полями вектора являются динамический массив, количество занятых ячеек и общее количество ячеек. При добавлении нового элимента в конец массива мы проверярем, есть ли в массиве еще место, если его нет, то увеличивает массив в 2 раза.

Для описания элементов массива я создал класс, хранящий в себе пару автомабольный номер и строка. И описал для него операцию присваевания и сравнения. Операция сравнения необходима для сравнения скорости моей сортировки и стандартной сортировки слиянием.

Далее мы считываем все данные которые нам доступны из потока, и помещаем их в вектор.

Сортировка разбита на 2 функции. Первая сортирует элементы вектора по возврастанию буквы в алфавитном порядке, вторая по возврастанию чисел от нуля до тысячи. Для ускорения работы программы и сокращения количество копирований мы создаем копию вектора той же длины, что и исходный, в который будет складываться ответ. После чего для следующей функции уже второй вектор будет исходным, а в первый будет складываться ответ. Таким образом мы их меняем местами нужное количество раз.

Сами сортировки являются сортировками подсчета. То есть мы создаем массим состоящий из количества элиментов в разряде. После чего считаем сколько раз на встетился тот, или иной элимент. Потом мы прибавляем к каждому элименту этого массива сумму всех предидущих элиментов, тем самым получая положения последнего элемента вектора для каждого значения разряда. Далее мы с конца проходим начальный вектор и копируем значения его элимента в ячейку второго вектора, номер которой задан в массиве. После добавления элимента мы уменьшаем на 1 значение в этой ячейке массива.

```
#ifndef VECTOR
#define VECTOR

#include <cstring>
#include <cstddef>
#include <iostream>

namespace NVector
{
    int min(int a,int b);


    template <typename T>
    class TVector
    {
        protected:
        T *Data;
        size_t Size_data;
        size_t Size_malloc;

        void Resize(size_t newsize);

        void Copy(T*& data1, T*& data2, size_t begin, size_t end);

        void Clear();

        public:

        TVector();

        ~TVector();

        const int Size();

        int Size();

        void Renew(size_t new_size);

        T* Begin();

        T* End();

        void Push_back(const T& item);

        T& operator[](size_t itr);

        const T &operator[](size_t itr);

        void operator=(const TVector<T> &second);
```

```
50
51      }; //class TVector
52
53  } //namespace NVector
54
55  #endif
56
57  #ifndef SORTBASE
58  #define SORTBASE
59
60  #include "vector.hpp"
61  #include "base.hpp"
62
63  namespace NSort_base
64  {
65      const void Sort_ch( NVector::TVector<NBase::TBase_elem> &base,NVector::TVector<
            NBase::TBase_elem> &newbase, int num)
66      {
67          int k[26] = {};
68          //newbase.Renew(base.Size());
69          int size = base.Size();
70          for(int i = 0;i < size;++i)
71          {
72              int l = (int)base[i].Carnum.Sym[num]-65;
73              ++k[l];
74          }
75          for(int i=1;i<26;++i)
76          {
77              k[i]+=k[i-1];
78          }
79          for(int i = size - 1;i >= 0;--i)
80          {
81              int l = (int)base[i].Carnum.Sym[num]-65;
82              newbase[k[l]-1] = base[i];
83              --k[l];
84          }
85
86      }
87
88
89      void Sort_int(const NVector::TVector<NBase::TBase_elem> &base,NVector::TVector<
            NBase::TBase_elem> &newbase)
90      {
91          int k[1000] = {};
92          //newbase.Renew(base.Size());
93          int size = base.Size();
94          for(int i = 0;i < size;++i)
95          {
96              int l = base[i].Carnum.Num;
```

```
97              ++k[l];
98          }
99          for(int i=1;i<1000;++i)
100         {
101             k[i]+=k[i-1];
102         }
103         for(int i = size - 1;i >= 0;--i)
104         {
105             int l = base[i].Carnum.Num;
106             newbase[k[l]-1] = base[i];
107             --k[l];
108         }
109     }
110
111     void Sort(NVector::TVector<NBase::TBase_elem> &base1)
112     {
113         NVector::TVector<NBase::TBase_elem> base2;
114         base2.Renew(base1.Size());
115         Sort_ch(base1,base2,2);
116         Sort_ch(base2,base1,1);
117         Sort_int(base1,base2);
118         Sort_ch(base2,base1,0);
119     }
120
121 } // namespace NSort_base
122
123 #endif
124
125 #ifndef BASE
126 #define BASE
127
128 #include <iostream>
129 #include <iomanip>
130 #include <cstring>
131
132 namespace NCarnum
133 {
134     class TCarnum
135     {
136         public:
137         int Num = 0;
138         char Sym[3]={'A','A','A'};
139
140         void operator=(const TCarnum &second_elem);
141
142         friend std::istream& operator>>(std::istream &in, TCarnum &carnum);
143
144         friend std::ostream& operator<< (std::ostream &out, const TCarnum &carnum);
145
```

```
146         bool operator<(const TCarnum &b);
147 
148     }; // class TCarnum
149 
150 } // namespace NCarnum
151 
152 
153 namespace NBase
154 {
155     class TBase_elem
156     {
157         public:
158         NCarnum::TCarnum Carnum;
159         char Str[65] = "";
160 
161         void operator=(const TBase_elem &second_elem);
162 
163         friend std::istream& operator>>(std::istream &in, TBase_elem &elem);
164 
165         friend std::ostream& operator<< (std::ostream &out, const TBase_elem &elem);
166 
167         bool operator< (const TBase_elem &b);
168     }; // class TCarnum
169 } // namespace NCarnum
170 
171 #endif
```

main.cpp	
int main()	Считывает данные в вектор, сортирует и выводит
vector.hpp	
void Resize(size_t newsize)	Меняет размер массива с данными
void Copy(T*& data1, T*& data2, size_t begin, size_t end)	Копирует данные из одного массива в другой
void Clear()	Очищает массив
int Size()	Возвращает размер массива
void Renew(size_t new_size)	Изменяет размер массива и записывает этот размер в поля
T* Begin()	Возвращает указатель на первый элимент массива
T* End()	Возвращает указатель на элимент масива после последнего
void Push_back(const T& item)	Добавляет элимент в конец вектора, если это необходимо, то меняет его размер
sort_base.hpp	
const void Sort_ch(NVector:: TVectorNBase:: TBase_elem &base,NVector:: TVectorNBase:: TBase_elem &newbase, int num)	Сортирует элименты по определённой букве в номере
void Sort_int(const NVector:: TVectorNBase:: TBase_elem &base,NVector:: TVectorNBase:: TBase_elem &newbase)	Сортирует элименты по числу
void Sort(NVector:: TVectorNBase:: TBase_elem &base1)	Вызывает сортировки в нужном порядке

3 Консоль

```
pavel@pc ~/Projects/mai/2_course/DA/LB1/solution cat tests/test_1.t
A 159 PW vnooqchksmimoghvylzenlpinabmekwqtgkrlrobqslxysucapzhyoeplbjtszge
G 265 SA xjcwlteiidgberucfaqegemae
L 431 HA vysiuauvmpgdyalgccevgaikpwjzkhhftblzhdkcqtszivdmhm
L 759 JJ kdcpzfqzhqlozuvnutrdlhsnfkiqpqbisafzeivoyfrkvgwunbqcyzpimsi
O 072 VE dduvzhnfhvdeymgfgu
O 511 TC gqsgphojrlkdz
Q 902 LQ omypycnyvcibio
T 916 ZH inezpvokworktbaobbdlbejhqjvipyqdbslecedrasduhllavkiknc
U 482 CI iregithcmyzujjqkjdpqqmfdzjaelsqdnhjqobutpojrl
V 084 BM fhoobacktutcvmbvbrcileqybzlczytlleucqnlnpbyiyblsf
W 031 GY wovxkbohipmgdlrabpvaeoznuxkvhitehlvsbgxafghe
pavel@pc ~/Projects/mai/2_course/DA/LB1/solution  g++ main.cpp -o main
pavel@pc ~/Projects/mai/2_course/DA/LB1/solution  cat tests/test_1.t | ./main
A 159 PW vnooqchksmimoghvylzenlpinabmekwqtgkrlrobqslxysucapzhyoeplbjtszge
G 265 SA xjcwlteiidgberucfaqegemae
L 431 HA vysiuauvmpgdyalgccevgaikpwjzkhhftblzhdkcqtszivdmhm
L 759 JJ kdcpzfqzhqlozuvnutrdlhsnfkiqpqbisafzeivoyfrkvgwunbqcyzpimsi
O 072 VE dduvzhnfhvdeymgfgu
O 511 TC gqsgphojrlkdz
Q 902 LQ omypycnyvcibio
T 916 ZH inezpvokworktbaobbdlbejhqjvipyqdbslecedrasduhllavkiknc
U 482 CI iregithcmyzujjqkjdpqqmfdzjaelsqdnhjqobutpojrl
V 084 BM fhoobacktutcvmbvbrcileqybzlczytlleucqnlnpbyiyblsf
W 031 GY wovxkbohipmgdlrabpvaeoznuxkvhitehlvsbgxafghe
```

4 Тест производительности

```
[info] [2020-10-08 09:36:49] Running tests/test_0.t
Read time: 49us
Count of lines is 100
Counting sort time: 16us
STL stable sort time: 65us
[info] [2020-10-08 09:36:49] Running tests/test_1.t
Read time: 309us
Count of lines is 1000
Counting sort time: 186us
STL stable sort time: 375us
[info] [2020-10-08 09:36:49] Running tests/test_2.t
Read time: 3899us
Count of lines is 10000
Counting sort time: 1730us
STL stable sort time: 3812us
[info] [2020-10-08 09:36:49] Running tests/test_3.t
Read time: 30414us
Count of lines is 100000
Counting sort time: 20370us
STL stable sort time: 52669us
[info] [2020-10-08 09:36:50] Running tests/test_4.t
Read time: 268699us
Count of lines is 1000000
Counting sort time: 181746us
STL stable sort time: 612597us
```

Как видно, поразрядная сортировка быстрее стабильной сортировки слиянием из стандартной библиотеки. Это связанно с тем, что сортировка слиянием является сортировкой сравнения, то есть в лучшем случае ее сложность $O(N * lg(N))$. Поразрядная сортировка же работет за линейное время, поэтому быстрее.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился базово искать утечки памяти, скрытые копирования и делать профиллирование программы.

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.))