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: М. С. Лагуткина
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 27.10.2020
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: Числа от 0 до $2^{64} - 1$.

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки. В качестве разрядов выступают последовательности элементов из которых состоят сортируемые объекты. При этом объекты сортируются последовательно от младшего разряда к старшему. В результате чего объекты будут расположены в требуемом порядке. В качестве внутренней устойчивой сортировки используется сортировка подсчётом.

Алгоритм поразрядной сортировки относится к сортировкам за линейное время, которые достигают такой сложности благодаря отсутствию в них операций сравнения.

2 Исходный код

Для решения задачи потребовалось реализовать шаблонный класс вектора, класс массива для хранения строки, структуру для хранения пары ключ-значение, а также функцию сортировки вектора.

Класс массива представляет из себя массив указанной длинны, содержит конструктор по умолчанию, конструктор через ввод, деструктор, указатели на начало и за конец массива, перегруженный оператор индексирования элементов.

Реализован класс вектор (массив, который может расширятся посредством добавления в него новых элементов), который содержит конструктор, деструктор, добавление нового элемента в конец вектора, получение размера вектора, указатели на начало и за конец вектора, перегруженный оператор индексирования элементов.

Также реализована структура для хранения ключа и значения.

```
1  #include <iostream>
2  #include <cstring>
3  #include <chrono>
4  #include <algorithm>
5
6  using namespace std;
7
8  template <typename T, size_t size = 0>
9  class TArray {
10 protected:
11     T array_[size];
12 public:
13     TArray() {}
14     TArray(size_t defValue) {
15         for (auto& i : array_) {
16             i = defValue;
17         }
18     }
19     const size_t Size() {
20         return size;
21     }
22
23     T* begin() {
24         return &array_[0];
25     }
26
27     T* end() {
28         return &array_[size];
```

```
29      }
30
31      const T& operator[](size_t index) const {
32          return array_[index];
33      }
34      T& operator[](size_t index) {
35          return array_[index];
36      }
37  };
38
39  template <typename T>
40  class TVector {
41  public:
42      TVector() : buf(nullptr), size(0), capacity(0) {}
43      TVector(size_t newSize) {
44          if (newSize == 0) {
45              return;
46          }
47          size = newSize;
48          capacity = newSize;
49          buf = new T[size];
50      }
51
52      ~TVector() {
53          delete[] buf;
54          size = 0;
55          capacity = 0;
56          buf = nullptr;
57      }
58
59      void PushBack(T& value) {
60          if (size < capacity) {
61              buf[size++] = value;
62              return;
63          }
64          size_t newCapacity = 1;
65          if (capacity > 0) {
66              newCapacity = capacity * 2;
67          }
68          capacity = newCapacity;
69          T* newBuf = new T[capacity];
70          memcpy(newBuf, buf, sizeof(T)*size);
71          delete[] buf;
72          buf = newBuf;
73          buf[size++] = value;
74          return;
75      }
76
77      const size_t Size() {
```

```
78          return size;
79      }
80
81      T* begin() const {
82          return buf;
83      }
84
85      T* end() const {
86          if (buf) {
87              return buf + size;
88          }
89          return nullptr;
90      }
91
92      T& operator[](size_t i) {
93          return buf[i];
94      }
95
96      T& Get(size_t i) {
97          return this->buf[i];
98      }
99
100     void Set(size_t i, T& pair) {
101         this->buf[i] = pair;
102     }
103
104     T* GetBuf() {
105         return this->buf;
106     }
107
108     void ReplaceBuf(T* newBuf) {
109         memcpy(buf, newBuf, sizeof(T)*size);
110     }
111
112 private:
113     T *buf;
114     size_t size;
115     size_t capacity;
116 };
117
118 template <typename K, typename V>
119 struct TPair {
120     K key;
121     V value;
122 };
123
124 bool operator<(const TArray<char, 32> & lhs, const TArray<char, 32>& rhs) {
125     int i = 0;
126     while (lhs[i] == rhs[i]) {
```

```
127         if (i == 31) {
128             break;
129         }
130         i++;
131     }
132     return lhs[i] < rhs[i];
133 }
134 
135 template <typename K, typename V>
136 bool operator<(const TPair<K, V>& lhs, const TPair<K, V>& rhs) {
137     return lhs.key < rhs.key;
138 }
139 
140 template <typename K, typename V>
141 istream& operator >> (istream& input, TPair<K, V>& pair) {
142     for (int i = 0; i < pair.key.Size(); ++i) {
143         input >> pair.key[i];
144     }
145     input >> pair.value;
146     return input;
147 }
148 
149 template <typename K, typename V>
150 ostream& operator<< (ostream& output, TPair<K, V>& pair) {
151     for (int i = 0; i < pair.key.Size(); ++i) {
152         output << pair.key[i];
153     }
154     output << '\t' << pair.value;
155     return output;
156 }
157 
158 template <typename K, typename V>
159 void CountingSort(int i, TVector<TPair<K, V>>& v) {
160     TVector <TPair< TArray<char, 32>, uint64_t>> res(v.Size());
161     int count[16];
162     for (int j = 0; j < 16; ++j) {
163         count[j] = 0;
164     }
165     for (int j = 0; j < v.Size(); ++j) {
166         if (v[j].key[i] - '0' - 49 >= 0) {
167             count[v[j].key[i] - '0' - 39]++;
168         }
169         else {
170             count[v[j].key[i] - '0']++;
171         }
172     }
173     for (int j = 1; j < 16; ++j) {
174         count[j] += count[j - 1];
175     }
```

```
176     for (int j = v.Size() - 1; j >= 0; --j) {
177         if (v[j].key[i] - '0' - 49 >= 0) {
178             count[v[j].key[i] - '0' - 39]--;
179             res.Set(count[v[j].key[i] - '0' - 39], v[j]);
180         }
181         else {
182             count[v[j].key[i] - '0']--;
183             res.Set(count[v[j].key[i] - '0'], v[j]);
184         }
185     }
186     v.ReplaceBuf(res.GetBuf());
187 }
188 
189 template <typename K, typename V>
190 void RadixSort(TVector<TPair<K, V>>& v) {
191     for (int i = 31; i >= 0; --i) {
192         CountingSort(i, v);
193     }
194 
195 }
196 
197 int main() {
198     ios_base::sync_with_stdio(false);
199     cin.tie(nullptr);
200     TVector<TPair<TArray<char, 32>, uint64_t>> v;
201     TPair<TArray<char, 32>, uint64_t> pair;
202     pair.value = 0;
203     //auto start = chrono::steady_clock::now();
204     while (cin >> pair) {
205         v.PushBack(pair);
206         if (v.Size() == 0) {
207              return 0;
208         }
209     }
210     //auto finish = chrono::steady_clock::now();
211     //auto dur1 = finish - start;
212     //start = chrono::steady_clock::now();
213     //stable_sort(v.begin(), v.end());
214     RadixSort(v);
215     //finish = chrono::steady_clock::now();
216     //auto dur2 = finish - start;
217     // = chrono::steady_clock::now();
218     for (size_t i = 0; i < v.Size(); ++i) {
219         cout << v[i] << '\n';
220     }
221     //finish = chrono::steady_clock::now();
222     //auto dur3 = finish - start;
223     //cerr << "input " << chrono::duration_cast<chrono::milliseconds>(dur1).count() <<
        " ms" << endl;
```

```
    //cerr << "sort " << chrono::duration_cast<chrono::milliseconds>(dur2).count() << "
        ms" << endl;
    //cerr << "output " << chrono::duration_cast<chrono::milliseconds>(dur3).count() <<
        " ms" << endl;
    return 0;
}
}
```

3 Консоль

```
Mari@Mari:~/lr_da$ cat t1.txt
99999999999999999999999999989999 41
acaabd0aaaaaaaaaaaa9999999989998 18467
99999999999999979999999999989997 6334
79999999999999969999999999989996 26500
b99909999999999599999999999989995 19169
00999999999999949999999999989994 15724
99999999999999999999999999989999 11478

Mari@Mari:~/lr_da$ ./solution <1.txt
00999999999999949999999999989994        15724
79999999999999969999999999989996        26500
99999999999999979999999999989997        6334
99999999999999999999999999989999        41
99999999999999999999999999989999        11478
acaabd0aaaaaaaaaaaa9999999989998        18467
b99909999999999599999999999989995        19169
```

4 Тест производительности

Тест производительности представляет из себя следующее: сортировку 80 тысяч входных данных с помощью реализованной поразрядной сортировки и $std :: stable\_sort$.

Моя реализация:

```
Mari@Mari:~/lr_da$ ./solution <test.txt >res.txt
input 184902 ms
sort 2153 ms
output 200914 ms
```

$std :: stable\_sort$:

```
Mari@Mari:~/lr_da$ ./solution <test.txt >res.txt
input 248025 ms
sort 2203 ms
output 197710 ms
```

Как видно, поразрядная сортировка работает не намного быстрее, чтобы была видна разница между сортировками с ассимптотикой между $O(n)$ и $O(\log_2 n)$.

5 Выводы

Выполнив лабораторную работу я поняла, что поразрядная сортировка полезна, однако из теста на производительность можно сделать вывод, что она не настолько эффективна, как хотелось бы. Так же для данной лабораторной работы я научилась использовать шаблоны для классов, а также узнала о различных утилитах для тестирования 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.))

[2] *Поразрядная сортировка — Вики университета ITMO.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 01.10.2020).