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: А. К. Болдырев
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: Карманная сортировка.

Вариант ключа: `Вещественные числа в промежутке [-100, 100].`

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

1 Описание

Как сказано в книге «Т. Кормен, Алгоритмы. Построение и анализ»: «Идея алгоритма состоит в том, что промежуток [0; 1) делится на n равных частей, после чего для чисел из каждой части выделяется свой ящик-черпак (bucket), и n подлежащих сортировке чисел раскладываются по этим ящикам. Поскольку числа равномерно распределены на отрезке [0; 1), следует ожидать, что в каждом ящике их будет немного. Теперь отсортируем числа в каждом ящике по отдельности и пройдемся по ящикам в порядке возрастания, выписывая попавшие в каждый из них числа также в порядке возрастания.»
Будем считать, что на вход подается n-элементный массив A, причем $0 <= A[i] < 1$ для всех i. Используется также вспомогательный массив $B[0..n-1]$, состоящий из списков, соответствующих ящикам.
Математическое ожидание времени работы карманной сортировки линейно зависит от количества чисел.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру TDats, в которой будем хранить ключ(key) в переменной типа double и значение(val) в переменной типа val[2049]. Реализуем основные функции для вектора TVector.
Функция TVector<TData> BucketSort (TVector<TData> const& vect, double n) выполняет соритровку. На вход ей подается вектор, который нужно отсортироваеть и его размер. В самой функции заводится вектор векторов, в конце происходит вывод отсортированного вектора. Принцип сортировки описан в предыдущем пункте.
В функции main происходит ввод пар ключей-значений и запись их в вектор. После вызова функции сортировки происходит вывод отсортированного вектора.

main.cpp	
TVector<TData> BucketSort(TVector<TData> const vect, double n)	Функция карманной сортировки.
int main()	В функции main происходит ввод пар ключей-значений и запись их в вектор.

```
1  #include <iostream>
2  #include <cstdio>
3  #include <limits>
4  #include <cstdint>
5  #include <iomanip>
6
7  struct TData{
8      double key;
9      char val[2049] = {'\0'};
10 };
11
12 template<class T>
13 class TVector {
14 private:
15     size_t size_;
16     size_t cap_;
17     T *data_;
18 public:
19     TVector() :
20             size_(0), cap_(0), data_(nullptr) {};
21
22     TVector(double n) :
23             size_(n), cap_(n), data_(new T[cap_]) {};
24
```

```
25      TVector(double n, T x){
26          size_ = n;
27          cap_ = n;
28          data_ = new T[cap_];
29          for (size_t i = 0; i < size_; i++)
30              data_[i] = x;
31      }
32
33      TVector(const TVector<T>& other){
34          if (data_)
35              delete[] data_;
36          data_ = new T[other.cap_];
37          for (size_t i = 0; i < other.size_; ++i) {
38              data_[i] = other.data_[i];
39          }
40          size_ = other.Size();
41          cap_ = other.cap_;
42      }
43
44      ~TVector(){
45          delete[] data_;
46      }
47
48      T& operator[] (const int id) const{
49          return data_[id];
50      }
51
52      TVector<T>& operator= (const TVector<T>& other) {
53          if (this != &other) {
54              T* tmp = new T[other.size_];
55              for (size_t i = 0; i < other.size_; ++i) {
56                  tmp[i] = other.data_[i];
57              }
58              delete[] data_;
59              data_ = tmp;
60              size_ = other.size_;
61              cap_ = other.cap_;
62          }
63          return *this;
64      }
65
66      void PushBack(const T& newdata_) {
67          if (cap_ == size_){
68              cap_ *= 2;
69              if (cap_ == 0)
70                  cap_ = 1;
71              T *tmp = new T[cap_];
72              for (size_t i = 0; i < size_; ++i) {
73                  tmp[i] = data_[i];
```

```
74              }
75              delete[] data_;
76              data_ = tmp;
77          }
78          data_[size_++] = newdata_;
79      }
80
81      size_t Size() const {
82          return size_;
83      }
84  };
85
86
87  void InsSort(TVector<TData>& vect, double n) {
88      for (int i = 1; i < n; i++) {
89          TData now = vect[i];
90          int j = i - 1;
91          while(j >= 0 && vect[j].key > now.key) {
92              vect[j + 1] = vect[j];
93              --j;
94          }
95          vect[j + 1] = now;
96      }
97  }
98
99  TVector<TData> BucketSort(TVector<TData> const& vect, double n) {
100     TVector<TVector<TData> > buckets(n);
101
102     double min_el = std::numeric_limits<double>::max();
103     double max_el = std::numeric_limits<double>::min();
104
105     for (int i = 0; i < n; i++) {
106         TData elem = vect[i];
107         min_el = std::min(min_el, elem.key);
108         max_el = std::max(max_el, elem.key);
109     }
110
111     double len = (max_el - min_el + 1e-9);
112
113     for (int i = 0; i < n; i++) {
114         size_t num = ((vect[i].key - min_el) / len) * n;
115
116         if (num >= (size_t) n)
117             num = n - 1;
118
119         buckets[num].PushBack(vect[i]);
120
121     }
122
```

```
123     for (int i = 0; i < n; i++) {
124         InsSort(buckets[i], buckets[i].Size());
125     }
126 
127     TVector<TData> res(n);
128     int ind = 0;
129     for (int i = 0; i < n; i++) {
130         for (size_t j = 0; j < buckets[i].Size(); j++) {
131             res[ind++] = buckets[i][j];
132         }
133     }
134     return res;
135 }
136 
137 
138     int main() {
139         std::ios::sync_with_stdio(false);
140         TVector<TData> vect;
141         TData in;
142 
143 
144 
145         while(std::cin >> in.key >> in.val) {
146             vect.PushBack(in);
147         }
148 
149         TVector<TData> res;
150         res = BucketSort(vect, vect.Size());
151 
152         for (size_t i = 0; i < res.Size(); i++) {
153             std::cout << std::fixed << std::setprecision(6) << res[i].key << '\t' <<
                    res[i].val << "\n";
154         }
155         return 0;
156     }
```

3 Консоль

```
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$ ./a.out
-100 adsdsfa
-99 asffa
76 afdsdfa
-99 afdsfagfgfgs
78 dgasgsdag
12 easdfgfads
1 adsfgfasdf
-2 adsggasdg
99 dsagfdsagdsf
100 dasgdsfgasd


-100.000000 adsdsfa
-99.000000 asffa
-99.000000 afdsfagfgfgs
-2.000000 adsggasdg
1.000000 adsfgfasdf
12.000000 easdfgfads
76.000000 afdsdfa
78.000000 dgasgsdag
99.000000 dsagfdsagdsf
100.000000 dasgdsfgasd
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$
```

4 Тест производительности

Тест производительности представляет из себя следующее: сортировка ключей и значений происходит при помощи 3 сортировок, первая - BubbleSort, вторая - QuickSort, третья BucketSort. Для начала запустим сортировку для файла, состоящего из 100 строк.

```
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$ ./a.out < 01_100.t
BubbleSort time: 0.000374 sec
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$ g++ std.cpp
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$ ./a.out < 01_100.t
QuickSort time: 9.1e-05 sec
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$ g++ lab1.cpp
anton@anton-Lenovo-ideapad-320-15IKB:~/DA/Lab_1_BucSo$ ./a.out < 01_100.t
BucketSort time: 0.020776 sec
```

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я реализвоал карманную сортировку (BucketSort) на языке C++, которая работает за линейное время (O(N)). Также провел сравнение со стандартной сортировкой в C++ и соритровкой пузырьком, и сделал вывод, что карманная сортировка лучше всего подходит для вещественных чисел. Применить сортировку можно, например в базе данных, для того, чтобы осортировать поля по ключам, которые повторяются и при этом максимальное значение ключа является вещественным числом. Как например в моем варианте лабораторной работы, где ключами являлись вещественные числа в промежутке [-100, 100].

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] *Блочная сортировка — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Блочная_сортировка` .

[3] Макаров Н.К. Лекции по курсу «Дискретный анализ»