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. В. Полей-Добронравова
Преподаватель: Д. Е. Ильвохин
Группа: М8О-307Б

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: Сортировка подсчётом.

Вариант ключа: `Почтовые индексы: шестизначные числа.`

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 сортировки подсчётом. Используемая память и скорость этой сортировки O(n + k), где n - количество элементов массива, k - максимальный элемент массива.

Как сказано в [1]: «основная идея сортировки подсчетом заключается в том, чтобы для каждого входного элемента x определить количество элементов, которые меньше x».

В этом алгоритме используют три массива (вектора): исходный массив A, массив C для подсчета того, сколько раз встретился элемент от 0 до максимального, массив B для промежуточного результата сортировки.
Сначала следует заполнить массив C нулями, и для каждого A[i] увеличить C[A[i]] на 1. Далее подсчитывается количество элементов меньших или равных k-1. Для этого каждый C[j], начиная с C[1], увеличивают на C[j-1]. Таким образом в последней ячейке будет находиться количество элементов от 0 до k-1 существующих во входном массиве. На последнем шаге алгоритма читается входной массив с конца, значение C[A[i]] уменьшается на 1 и в каждый B[C[A[i]]] записывается A[i]. Алгоритм устойчив.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру *TElement*, в которой будем хранить: index - ключ в первоначальном виде строки из 6 символов, key - целочисленное представление ключа для удобства сортировки по ключу, word - строка значения.
TVector это собственная реализация коллеции вектор. Он состоит из вместительности capacity, текущего размера vSize и указателя на последовательно расположенные в памяти элементы вектора data типа TElement. Методы TVector классические, есть метод изменения размера вектора.
Функция CountingSort осуществляет непосредственно сортировку, подходящую только для входных данных варианта. Функция типа void, т.к. изменяет исходный массив в памяти компьютера. Массив B из алгоритма здесь заменяет вектор temp, исходный вектор передается под именем v, вектор c под тем же именем.
В main программы осуществляется формирование исходного вектора. Считываются строки из input длиной максимум 72, записываются в переменную str. Если строка пустая, то условный оператор это видит и продолжает считывание до непустой строки. Далее из этой строки копируются значения нового TElement, считается целочисленное значение ключа. Далее идет вталкивание нового элемента в вектор, очистка вспомогательных строк. После того, как входной файл закончился, идет проверка на то, не пустой ли исходный вектор. Если не пустой, то он передаётся сортировке и выводится на экран.

```
#include <iostream>
#include <cstring>

const int LWORD = 65;
const int LKEY = 7;
struct TElement {
    char index[LKEY] = {};
    unsigned long long key = 0;
    char word[LWORD] = {};
};
class TVector {
    size_t vSize;
    size_t capacity;
    TElement* data;
public:
    TVector() {
        vSize = 0;
        capacity = 0;
        data = new TElement[2]();
    }
    TVector(size_t s) {
        vSize = 0;
        capacity = s;
```

```
24          data = new TElement[s]();
25      }
26      TVector(size_t s, TElement* d) {
27          vSize = s;
28          capacity = s;
29          data = new TElement[s]();
30          if (s != 0) {
31              for (size_t i = 0; i < s; i++) {
32                  data[i] = d[i];
33              }
34          }
35      }
36      TElement& operator[] (size_t i) {
37          return data[i];
38      }
39      size_t Size() {
40          return vSize;
41      }
42      size_t Capacity() {
43          return capacity;
44      }
45      void Resize() {
46          if (capacity == 0) {
47              capacity = 1;
48          }
49          capacity *= 2;
50          TElement* temp = new TElement[capacity]();
51           if(temp == 0x0 ) {
52               exit(-1);
53           }
54          for (size_t i = 0; i < vSize; i++) {
55              temp[i] = data[i];
56          }
57          delete [] data;
58          data = temp;
59      }
60      void PushBack(TElement elem) {
61          if (vSize == capacity) {
62              Resize();
63          }
64          data[vSize++] = elem;
65      }
66      ~TVector() {
67          delete [] data;
68      }
69  };
70
71  void CountingSort(TVector &v) {
72      TVector temp(v.Size());
```

```
73      unsigned long long max = v[0].key;
74      int i;
75      for (size_t j = 1; j < v.Size(); j++) {
76          if (v[j].key > max) {
77              max = v[j].key;
78          }
79      }
80      max++;
81      TVector c((size_t)max);
82      for (size_t j = 0; j < c.Capacity(); j++) {
83          c[j].key = 0;
84      }
85      for (size_t j = 0; j < v.Size(); j++) {
86          ++c[v[j].key].key;
87      }
88      for (size_t j = 1; j < c.Capacity(); j++) {
89          c[j].key = c[j].key + c[j - 1].key;
90      }
91      for (int j = v.Size() - 1; j >= 0; j--) {
92          c[v[j].key].key--;
93          temp[c[v[j].key].key] = v[j];
94      }
95      for (size_t m = 0; m < v.Size(); m++) {
96          v[m] = temp[m];
97      }
98  }
99
100 int main(void) {
101     std::ios::sync_with_stdio(false);
102     TVector dataVector;
103     TElement element;
104     size_t i = 0;
105     char str[72] = {};
106     while (true) {
107         if (std::cin.eof()) {
108             break;
109         }
110         str[0] = '\0';
111         std::cin.getline(str, 72);
112          if (str[0] != '\0') {
113             element.key = 0;
114             for (i = 0; i < 6; i++) {
115                 element.index[i] = str[i];
116                 element.key = element.key * 10 + str[i] - '0';
117             }
118             i++;
119             for (; i < 72; i++) {
120                 if (str[i] == '\0') {
121                     break;
```

```
122                 }
123                 element.word[i - 7] = str[i];
124             }
125             element.word[64] = '\0';
126             element.index[6] = '\0';
127             dataVector.PushBack(element);
128             for (i = 0; i < 72; i++) {
129                 str[i] = 0;
130             }
131              for (i = 0; i < 64; i++) {
132                  element.word[i] = 0;
133              }
134              for (i = 0; i < 6; i++) {
135                  element.index[i] = 0;
136              }
137         }
138     }
139     if (dataVector.Size() != 0) {
140         CountingSort(dataVector);
141         for (size_t j = 0; j < dataVector.Size(); j++) {
142             std::cout << dataVector[j].index << "\t"<< dataVector[j].word << std::endl;
143         }
144     }
145     return 0;
146 }
```

3 Консоль

```
MacBook-Pro-Amelia:1 amelia$ cat test1.txt
999999 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na


011111 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
000000 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
999997 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat

999999 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
000100 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
999999 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na
MacBook-Pro-Amelia:1 amelia$ ./dat < test1.txt
000000 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
000100 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
011111 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
999997 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
```

999999 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na
999999 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
999999 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na

4 Тест производительности

Тест производительности представляет из себя следующее: при тесте с максимальным значением ключа 999999 сравним разницу в скорости сортировки подсчетом и быстрой сортировки(Хоара).

```
MacBook-Pro-Amelia:1 amelia$ ./dat < test.txt
The time of counting sort: 226 ms
The time of quick sort: 8 ms
MacBook-Pro-Amelia:1 amelia$ ./dat < test1.txt
The time of counting sort: 225 ms
The time of quick sort: 0 ms
```

В первом тесте было 17000 элементов для анализа, во втором - 9 элементов. Время сортировки Хоара в среднем O(n log n), в худшем случае - O(n^2). Т.е. время сортировки Хоара зависит только от количества элементов для сортировки, чем их меньше, тем быстрее эта сортировка.
Сортировка подсчетом же имеет скорость O(n + k), т.е. критически зависит от диапазона значений сортируемого вектора. В обоих тестах максимальное значение элемента вектора 999999, что заставляет сортировку тратить много времени и памяти, количество элементов в этих двух тестах практически не влияют на скорость сортировки. В то же время сортировка Хоара во-первых в несколько раз быстрее на данных тестах, в первом случае:
n log n = 71917;
n + k = 1016999;
Во втором случае:
n log n = 8.5;
n + k = 1000008;
Из вышесказанного следует, что нужно использовать сортировку подсчетом в ситуациях, когда диапазон значений не велик, или гораздо меньше числа сортируемых объектов.

Так же прилагаю код функции сортировки Хоара, адаптированный под мой вариант входных значений.

```
1 void quicksort(TVector &mas, int first, int last)
2 {
3     int mid;
4     TElement count;
5     int f = first, l = last;
6     mid = mas[(f + l) / 2].key; //
7     do
8     {
```

```
 9          while (mas[f].key < mid) f++;
10          while (mas[l].key > mid) l--;
11          if (f <= l) //
12          {
13              count = mas[f];
14              mas[f] = mas[l];
15              mas[l] = count;
16              f++;
17              l--;
18          }
19      } while (f<l);
20      if (first<l) quicksort(mas, first, l);
21      if (f<last) quicksort(mas, f, last);
22  }
```

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научилась правильной работе со строками в C++. Попробовала работать с valgrind, оказалось, что под операционную систему macOS эта утилита очень плохо работает, но поняла её принцип. Так же я научилась обращать внимание на экономный выбор типа данных и ошибки, связанные с неправильным выбором типа. Например, цикл $for(size_t k = 1000; k >= 0; k--)$ будет работать неправильно, т.к. k никогда не примет отрицательное значение, в отличии от например int.

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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008