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: Р. С. Лисин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-20
Дата:
Оценка:
Подпись:

Москва, 2022

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: `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.

Как сказано в [1]: «Сам алгоритм состоит в последовательной сортировке объектов какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего разряда к старшему, после чего последовательности будут расположены в требуемом порядке». В качестве устойчивой сортировки для разряда использую устойчивую версию сортировки подсчётом.

2 Исходный код

Для хранения пар «ключ-значение» будем использовать структуру $TPair$, так как это удобно. Ключ будем хранить в статическом массиве на 32 элемента, так как считываем его по одному символу. А для значения будем выделять динамическую память, так как там могут быть строки переменной длины до 64 символов.

Поразрядная сортировка $RadixSort$ реализуется итерацией по каждому символу ключа элементов массива, начиная справа. Эти символы сортируются с помощью сортировки подсчётом $CountingSort$, чтобы программа работала за линейное время.

Опишем функцию $CountingSort$. Устойчивая сортировка подсчётом осуществляется с помощью двух вспомогательных массивов: один - для счётчика, другой - для записи отсортированного результата. Для удобства обозначим: v - исходный массив, res - результирующий массив, $count$ - массив для подсчёта вхождений.

Сначала заполняем $count$ нулями. Затем для каждого $v[i]$ увеличиваем $count[v[i]]$ на единицу. Так, мы посчитаем количество вхождений каждого элемента. Суммируем каждый $count[i]$ с $count[i-1]$, кроме $count[0]$ (насчитываем префиксные суммы). Далее читаем входной массив с конца, записываем в $res[count[v[i]]]$ $v[i]$ и уменьшаем $count[v[i]]$ на единицу.

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  struct TPair {
7      char Key[32];
8      char* Value;
9  };
10
11 istream& operator>> (istream& input, TPair& pair) {
12     for (int i = 0; i < 32; ++i) {
13         input >> pair.Key[i];
14     }
15     input >> pair.Value;
16     return input;
17 }
18
19 ostream& operator<< (ostream& output, TPair& pair) {
20     for (int i = 0; i < 32; ++i) {
21         output << pair.Key[i];
22     }
23     output << '\t' << pair.Value;
24     return output;
25 }
26
27 void CountingSort(int i, vector<TPair>& v) {
```

```
28      vector<TPair> res(v.size());
29      int count[16] = { 0 };
30      for (long long j = 0; j < v.size(); ++j) {
31          if (v[j].Key[i] - '0' - 49 >= 0) { // ASCII codes: 'a' = 97, '0' = 48
32              ++count[v[j].Key[i] - '0' - 39];
33          }
34          else {
35              ++count[v[j].Key[i] - '0'];
36          }
37      }
38      for (int j = 1; j < 16; ++j) {
39          count[j] += count[j - 1];
40      }
41      for (long long j = v.size() - 1; j >= 0; --j) {
42          if (v[j].Key[i] - '0' - 49 >= 0) {
43              --count[v[j].Key[i] - '0' - 39];
44              res[count[v[j].Key[i] - '0' - 39]] = v[j];
45          }
46          else {
47              --count[v[j].Key[i] - '0'];
48              res[count[v[j].Key[i] - '0']] = v[j];
49          }
50      }
51      v = move(res);
52  }
53
54  void RadixSort(vector<TPair>& v) {
55      for (int i = 31; i >= 0; --i) {
56          CountingSort(i, v);
57      }
58  }
59
60  int main() {
61      ios::sync_with_stdio(false);
62      cin.tie(nullptr);
63      cout.tie(nullptr);
64      vector<TPair> v;
65      TPair pair;
66      pair.Value = (char*)malloc(sizeof(char) * 64);
67      while (cin >> pair) {
68          v.push_back(pair);
69          pair.Value = (char*)malloc(sizeof(char) * 64);
70      }
71      if (v.size() == 0) {
72          return 0;
73      }
74      RadixSort(v);
75      for (int i = 0; i < v.size(); ++i) {
76          cout << v[i] << '\n';
```

```
77      }
78      return 0;
79  }
```

3 Консоль

```
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ g++ lab1.cpp
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ cat test1
00000000000000000000000000000010 hello
00000000000000000000000000000001 hi
00000000000000000000000000010010 are
00000000000000000000000000000100 what
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ ./a.out <test1
00000000000000000000000000000001 hi
00000000000000000000000000000010 hello
00000000000000000000000000000100 what
00000000000000000000000000010010 are
```

4 Тест производительности

Тест производительности представляет из себя следующее: поразрядная сортировка сравнивается с $std :: stable\_sort$. Время на ввод данных не учитывается. Количество пар «ключ-значение» для каждого файла равно десять в степени номер теста минус один. Например, $02.t$ содержит десять пар, а $07.t$ миллион.

```
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ ./benchmark <tests/04.t >04.a
radix sort 6 ms
stable sort from std 0 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ ./benchmark <tests/05.t >05.a
radix sort 9 ms
stable sort from std 6 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ ./benchmark <tests/06.t >06.a
radix sort 37 ms
stable sort from std 168 ms
roma@DESKTOP-JD58QU2:~/Diskran/lab1$ ./benchmark <tests/07.t >07.a
radix sort 478 ms
stable sort from std 2371 ms
```

Глядя на результаты, видно, что $std :: stable\_sort$ выигрывает больше всего на самых маленьких тестах, а $RadixSort$ на самых больших. Сложность $std :: stable\_sort$ $O(n * log\ n)$, а сложность $RadixSort$ $O(m * n)$, где m - количество разрядов в числе. Так как логарифм является возрастающей функцией, переломный момент наступает, когда $log\ n$ становится больше, чем постоянная m.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать сортировки поразрядную и подсчётом, вспомнил работу с памятью. Это поможет мне в ситуации, когда нужно будет написать быструю сортировку, которая будет работать за линейное время. Также я узнал, что системные вызовы, которые выделяют динамическую память, к примеру *malloc*, работают достаточно долго, и это нужно всегда учитывать.

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