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. Т. Бахарев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2018

# 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.

Идея сортировки заключается в разбиении сортируемых элементов на разряды. Затем выполняется устойчивая сортировка подсчетом для каждого разряда. При этом, для строк подходит версия MSD(Most Significant Digit) - сортировка начинается от самого старшего разряда. Для чисел же нужно использовать LSD-версию(Least Significant Digit). В данной версии, сортировка начинается от самого младшего разряда.

# 2 Исходный код

Jopa

```
1  void sort(TElement*, int& amount_of_elems);
2  int main(int argc, char *argv[])
3  {
4      int size_of_array = 1; // default value
5      int amount_of_elems = 0;
6      TElement* array = new TElement[size_of_array]; // array of elems
7      while(1)
8      {
9          char ch = getchar();
10         if(ch == EOF || ch == '\0')
11             break;
12         else
13             ungetc(ch, stdin);
14         if(amount_of_elems == size_of_array)
15         {
16             TElement* tmp_buffer = new TElement[size_of_array];
17             memcpy(tmp_buffer, array, size_of_array*sizeof(TElement));
18             array = new TElement[size_of_array * 2];
19             memcpy(array, tmp_buffer, size_of_array*sizeof(TElement));
20             size_of_array *= 2;
21             delete[] tmp_buffer;
22         }
23         scanf("%s", array[amount_of_elems].Buffer);
24         scanf("%llu", &array[amount_of_elems].n);
25         ++amount_of_elems;
26         getchar();
27     }
28     sort(array, amount_of_elems);
29     for(int i = 0; i < amount_of_elems; ++i)
30     {
31         printf("%s\t", array[i].Buffer);
32         printf("%llu\n", array[i].n);
33     }
34     delete[] array;
35     return 0;
36 }
37 void sort(TElement* array, int& amount_of_elems)
38 {
39     const int radix = 16;
40     const int strLen = 32;
41     TElement* A = new TElement [amount_of_elems];
42     for(int digit = strLen - 1; digit >= 0; --digit)
43     {
44         int C[radix] = {0};
45         for(int i = 0; i < amount_of_elems; ++i)
46         {
```

```
47              A[i] = array[i];
48              int c = array[i].Buffer[digit];
49              if (c >= '0' && c <= '9') c = c - '0';
50              else c = c - 'a' + 10; // According to ASCII table
51              ++C[c];
52          }
53          for(int i = 1; i < radix; ++i)
54              C[i] = C[i] + C[i - 1];
55          for(int i = amount_of_elems - 1; i >= 0; --i)
56          {
57              TElement val = A[i];
58              int c = A[i].Buffer[digit];
59              if (c >= '0' && c <= '9') c = c - '0';
60              else c = c - 'a' + 10; // According to ASCII table
61              int position = C[c] - 1;
62              array[position] = val;
63              --C[c];
64          }
65      }
66      delete[] A;
67  }
```

# 3 Консоль

```
a.kukhticev$ gcc -pedantic -Wall -std=c99 -Werror -Wno-sign-compare -lm da10.c
-o da10 --some_long_argument=true
a.kukhticev$ cat test1
87 a
13 b
89 c
13 d
a.kukhticev$ ./da10 <test1
13 b
13 d
87 a
89 c
```

# 4 Тест производительности

*Тут Вы описываете собственно тест производительности, сравнение Вашей реализации с уже существующими и т.д.*

Тест производительности представляет из себя следующее: поиск образцов с помощью суффиксного массива сравнивается с поиском алгоритма КМП, но время на построение суффиксного массива не учитывается. Текст состоит из 1 миллиона букв: а образцов около 200 штук, длина которых может быть от 2 до 100 букв.

```
Andys-MacBook-Pro:kmp Andy$ g++ main.cpp
Andys-MacBook-Pro:kmp Andy$ ./a.out <../in.txt >out2.txt
Andys-MacBook-Pro:kmp Andy$ cat out2.txt | grep "time"
KMP search time: 1.639993 sec
Andys-MacBook-Pro:sa Andy$ make
g++ -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -O2 -o lab5
main.cpp suffix_tree.cpp suffix_array.cpp
Andys-MacBook-Pro:sa Andy$ ./lab5 <../in.txt >out1.txt
Andys-MacBook-Pro:sa Andy$ cat out1.txt | grep "time"
Suffix array build time: 2.179744 sec
Suffix array search time: 0.003511 sec
```

Как видно, что суффиксный массив выиграл у КМП, так как и т.д.

# 5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать и применять на практике эффективные сортировки за линейное время. Те алгоритмы, которые я знал до этого, значительно проигрывают по времени и по памяти. Также, я реализовал свой класс для строк, узнав при этом о внутреннем устройстве этого контейнера. Я научился писать бенчмарки для оценки времени работы программы, а также определять сложность работы алгоритмов.

# 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