

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 сортировка для каждого разряда. Алгоритм для сортировки разрядов может быть выбран любым, но для максимальной эффективности нужно использовать сортировки за линейное время. 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). Для выполнения данной лабораторной работы была выбрана устойчивая сортировка подсчетом. В данной версии, сортировка начинается от самого младшего разряда.

Свойства сортировки подсчетом:

- Не является сортировкой сравнением: ни одна пара элементов не сравнивается друг с другом
- Линейная (вернее, $O(k + n)$, но при $k = O(n)$ время выполнения $O(n)$)
- Устойчивая (стабильная)
- Не используются обмены(swap)
- Требует дополнительную память под массивы C и B размером k и n соответственно

Теорема 1 (О времени работы программы)

для n b -битовых чисел и натурального числа $r \leq b$ (цифры из r битов) алгоритм Radix-Sort выполнит сортировку за время $\Theta(\frac{b}{r})$

Описание работы программы

Сначала происходит ввод данных. Так как заранее неизвестно, сколько элементов будет обрабатываться, то исходный массив динамически расширяется. Как только встречается символ EOF, ввод считается завершенным и массив передается функции *sort*, которая сортирует данные. Затем происходит печать того же массива, который уже отсортирован.

2 Исходный код

```
1 #include <iostream>
2 #include <cstdio>
3 #include <cstdlib>
4 #include <cstring>
5 typedef struct TElement TElement;
6 struct TElement
7 {
8     char Buffer[33];
9     unsigned long long int n;
10 };
11 void sort(TElement*, int& amount_of_elems);
12 int main(int argc, char *argv[])
13 {
14     int size_of_array = 1; // default value
15     int amount_of_elems = 0;
16     TElement* array = new TElement[size_of_array]; // array of elems
17     while(1)
18     {
19         char ch = getchar();
20         if(ch == EOF || ch == '\0')
21             break;
22         else
23             ungetc(ch, stdin);
24         if(amount_of_elems == size_of_array)
25         {
26             TElement* tmp_buffer = new TElement[size_of_array];
27             memcpy(tmp_buffer, array, size_of_array*sizeof(TElement));
28             array = new TElement[size_of_array * 2];
29             memcpy(array, tmp_buffer, size_of_array*sizeof(TElement));
30             size_of_array *= 2;
31             delete[] tmp_buffer;
32         }
33         scanf("%s", array[amount_of_elems].Buffer);
34         scanf("%llu", &array[amount_of_elems].n);
35         ++amount_of_elems;
36         getchar();
37     }
38     sort(array, amount_of_elems);
39     for(int i = 0; i < amount_of_elems; ++i)
40     {
41         printf("%s\t", array[i].Buffer);
42         printf("%llu\n", array[i].n);
43     }
44     delete[] array;
45     return 0;
46 }
47
```


4 Тест производительности

Тест производительности представляет из себя следующее: Производится замер времени для работы поразрядной сортировки и сравнивается со временем работы сортировки `std::sort()`.

```
alex$ bash benchmark.sh
g++ -std=c++11 -o da_1 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
-lm main.cpp sort.cpp
Time for Radix Sort

real 0m20.225s
user 0m19.748s
sys 0m0.472s
Time for std::sort()

real 0m28.851s
user 0m27.836s
sys 0m1.000s
alex$
```

Как видно, поразрядная сортировка выиграла по времени у быстрой сортировки(`std::sort()`). Это связано с тем, что сложность первой сортировки - линейная, а у `std::sort()` она равна $O(N\log N)$.

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. Однако чтобы это увидеть, надо обрабатывать большие объемы данных, иначе преимущество алгоритма не так очевидно. Также, я научился писать бенчмарки для оценки времени работы программы и определять сложность работы алгоритмов, а это, на мой взгляд, очень ценные знания.

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: 16.12.2017).