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: И. Д. Черненко
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

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. Основная идея сортировки, как сказано в [2]: «Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.» Кроме того, важно отметить, что все промежуточные сравнения стоит проводить с использованием сортировки с эффективностью не менее чем O(n), иначе поразрядная сортировка не сможет работать за линейное время.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру Tkv, в которой будем хранить ключ и значение. Далее перед написанием сортировки был реализован вектор $TKeyValueVector$, для возможности динамического ввода данных. Его реализация включает в себя функции $TCreate()$(создает динамический массив), $TDelete()$(удаляет динамический массив), $Tset()$(присваивает значение элементу вектора), $TGet()$(возвращает значение элемента вектора), $TGetSize()$ (возвращает кол-во элементов вектора), $TGetCap()$(возвращает максимально возможный размер вектора, при данных ограничениях).
$TSetSize()$(устанавливает размер вектора), $TSetCap()$(устанавливает максимальный размер вектора, при заданных условиях), далее были реализованы функции, участвующие непосредственно в процессе сортировки $CharToInteger()$ (интерпертирует подаваемые на вход символы от 0 до 9 и от a до f соответственно в числа формата int от 0 до 9 и от 10 до 15 соответственно, а потом возвращает полученное значение), далее была написана вспомогательная сортировка(чтобы сортировать разряды ключей), причем была выбрана сортировка подсчетом так как она работает за O(n) $CountSort()$, далее был реализован сам алгоритм поразрядной сортировки $RadixSort()$, далее была описано основное тело программы с вводом пар типа «ключ-значение» и созданием вектора для динамического ввода данных. Кроме того, стоит отметить, что изначально алгоритм был написан с использованием статических массивов для хранения значений, которые позже были заменены на вектора.

```
#include <iostream>
#include <ctype.h>
typedef struct {
    char value[65] = {0};
    char key[33] = {0};
} Tkv;
typedef struct {
    Tkv* body;
    int size;
    int cap;
} TKeyValueVector;
TKeyValueVector* TCreate() {
    TKeyValueVector* v = (TKeyValueVector*)malloc(sizeof(TKeyValueVector));
    v->body = nullptr;
    v->size = 0;
    v->cap = 1;
    return v;
}
void TDelete(TKeyValueVector* v) {
    free(v->body);
    free(v);
```

```
}
void TSet(TKeyValueVector* v, int i, Tkv val) {
    v->body[i] = val;
}
Tkv TGet(TKeyValueVector* v, int i) {
    return v->body[i];
}
int TGetSize(TKeyValueVector* v) {
    return v->size;
}
int TGetCap(TKeyValueVector* v) {
    return v->cap;
}
void TSetSize(TKeyValueVector* v, int new_size) {
    v->size = new_size;
}
void TSetCap(TKeyValueVector* v, int new_size) {
    Tkv* reBody = (Tkv*)realloc(v->body, new_size * sizeof(Tkv));
    v->body = reBody;
}
int CharToInteger(char a) {
    if (isdigit(a)) {
        return (a-'0');
    } else {
       return (a-'W');
    }
}
void CountSort(TKeyValueVector* v, int k) {
    const int NUMBER_OF_HEX_DIGITS = 16;
    int n = TGetSize(v);
    TKeyValueVector* output = TCreate();
    TSetCap(output, n+1);
    TSetSize(output, n);
    int c[NUMBER_OF_HEX_DIGITS] = {0};
    for (int i = 0; i < n; i++) {
        c[CharToInteger(TGet(v, i).key[k])]++;
    }
    for (int i = 1; i < NUMBER_OF_HEX_DIGITS; i++) {
        c[i] += c[i - 1];
    }
    for (int i = n-1; i >= 0; i--) {
        TSet(output,c[CharToInteger(TGet(v, i).key[k])]-1, TGet(v, i));
        c[CharToInteger(TGet(v, i).key[k])]--;
    }
    for (int i = 0; i < n; i++) {
        TSet(v, i, TGet(output, i));
    }
    TDelete(output);
}
```

```
71 void RadixSort(TKeyValueVector* arr, int k) {
72     for (int i = k; i >= 0; i--) {
73         CountSort(arr, i);
74     }
75 }
76 int main() {
77     int posForRadix = 31;
78     std::ios_base::sync_with_stdio(false);
79     std::cin.tie(NULL);
80     TKeyValueVector* v = TCreate();
81     Tkv elem;
82     while (std::cin >> elem.key >> elem.value) {
83         if (TGetSize(v) + 1 == TGetCap(v)) {
84             v->cap *= 2;
85             TSetCap(v, v->cap);
86         }
87         int size = TGetSize(v);
88         TSet(v, size, elem);
89         TSetSize(v, size + 1);
90     }
91     RadixSort(v, posForRadix);
92     int size = TGetSize(v);
93     for (int i = 0; i < size; i++) {
94         std::cout << TGet(v, i).key << '\t' << TGet(v, i).value << std::endl;
95     }
96     TDelete(v);
97     return 0;
98 }
```

3 Консоль

```
D:\Study\Labs2course\DA_lab1\cmake-build-debug\DA_lab1.exe
00000000000000000000000000000000  n399tann9nnt3ttnaaan9nann93na9t3
affffffffffffffffffffffffffffff  n399tann9nnt3ttnaaan9nann9
afffffffffffffffffffffffffffff  n399tann9nnt3ttnaaan
00000000000000000000000000000000  n399ta

^D
00000000000000000000000000000000 n399tann9nnt3ttnaaan9nann93na9t3
00000000000000000000000000000000 n399ta
affffffffffffffffffffffffffffff n399tann9nnt3ttnaaan9nann9
afffffffffffffffffffffffffffff n399tann9nnt3ttnaaan

Process finished with exit code 0
```

4 Тест производительности

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

Тест производительности представляет из себя следующее: поразрядная сортировка строк с использ, 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.

```
Тест производительности представляет из себя следующее: сортировка пары «ключ-значение
ключ является строкой длиной 32,а значение строкой длиной не больше 64,сравнивается
с сортировкой функцией sort из стандартной библиотеки «C++»,при этом время
создания динамического массива не учитывается. Текст состоит из 1000000 образцов.
The time: 10557 ms

Process finished with exit code 0

The time: 17216 ms

Process finished with exit code 0
```

Как видно, стандартная сортировка, работающая за O(nlog(n)) выиграла у написанного мною алгоритма поразрядной сортировки, хотя на то есть несколько причин. В реализации через «sort» строки сравниваются напрямую через внутренние ф-ции языка, без привязки к разрядности, у меня же сравнение образцов происходит посимвольно (целых 32 разряда), при этом само сравнение происходит в отдельной функции, что существенно замедляет работу программы. Кроме того, нельзя не упомянуть неэффективность написанного мной прототипа вектора, по сравнению с «vector» из стандартной библиотеки. Таким образом, можно сделать вывод о том, что при использовании поразрядной сортировки на строках теряется значительная её эффективность, что не позволило мне достичь линейного времени для данной сортировки.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился писать поразрядную сортировку и сортировку подсчетом. Приобрел навыки работы со скриптами на языке программирования «Python», а также с «Си-строками», в дальнейшем данная сортировка может пригодится при написании других работ, а навыки работы с «Python», при написании тестов в будущем.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2013. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 226 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Поразрядная сортировка — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 28.09.2019).