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: Д. С. Пивницкий
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 01.01.2021
Оценка:
Подпись:

Москва, 2021

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: Числа от 0 до $2^{64} - 1$.

Вариант значения: Строки переменной длины (до 2048 символов).

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки. В качестве ключа выступают строки переменной длины (до 2048 символов).

Как сказано в Wikipedia: ««По аналогии с разрядами чисел будем называть элементы, из которых состоят сортируемые объекты, разрядами. 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 внутренней устойчивой сортировки будем использовать сортитовку подсчётом. Поразрядная сортировка в таком случае работает за $O(\frac{b}{R}(n + 2^R)$, причем $O(n + 2^R)$ - сложность сортировки подсчётом, где b - длина числа (в нашем случае в битах, то есть $b = 64$), R - длина (размер) разряда, по которому мы будем сортировать, n - кол-во пар «ключ-значение» в сортируемой последовательности. Так как R явно не задается, вычислять его будем следующим образом:

- При $b \leq \log_2 n$: $R = b$, при этом сложность выполнения программы сводится к $O(n + 2^b)$, причем $n + 2^b \leq 2n$
- При $b > \log_2 n$: $R = \log_2 n$, при этом сложность выполнения программы сводится к $O(\frac{2bn}{\log_2 n})$

2 Исходный код

Разобьем процесс написания программы на несколько этапов

1. Реализация необходимых новых типов (пара «ключ-значение» и вектор)
2. Реализация алгоритма сортировки подсчетом вектора пар по ключу по заданному разряду заданной длины
3. Реализация алгоритма поразрядной сортировки подсчетом вектора пар
4. Реализация ввода-вывода

Код:

```
#include <iostream>
#include <algorithm>
#include <cmath>

template <typename T>
class TVector {
public:
TVector() = default;

TVector(size_t newSize) {
capacity = newSize;
size = 0;
storage = new T[newSize];
}

TVector(const TVector& v) {
size = v.size;
capacity = v.capacity;
storage = new T[size];
for (size_t i = 0; i < size; i++) {
storage[i] = v.storage[i];
}
}

const T& operator[](size_t index) const {
return storage[index];
}

T& operator[](size_t index) {
return storage[index];
}

~TVector() {
```

```
34 delete[] storage;
35 }
36
37 size_t sizeM() const {
38 return size;
39 }
40
41 const T* begin() const {
42 return storage;
43 }
44
45 const T* end() const {
46 return storage + size;
47 }
48
49 T* Begin() {
50 return storage;
51 }
52
53 T* End() {
54 return storage + size;
55 }
56
57 void AddLast(const T& value) {
58 if (size < capacity) {
59 storage[size] = value;
60 ++size;
61 return;
62 }
63 int next_size = 10000;
64 if (capacity > 0)
65 next_size = capacity * 10;
66
67 T* tempvec = new T[next_size];
68 std::copy(begin(), end(), tempvec);
69 delete[] storage;
70 storage = tempvec;
71 capacity = next_size;
72 storage[size] = value;
73 ++size;
74 }
75
76 TVector& operator=(TVector& other) {
77 if (&other == this) {
78 return *this;
79 }
80 if (other.size <= capacity) {
81 std::copy(other.begin(), other.end(), begin());
82 size = other.size;
```

```
}
else {
delete[] storage;
storage = new T[other.capacity];
std::copy(other.begin(), other.end(), begin());
size = other.size;
capacity = other.capacity;
}
return *this;
}

TVector& operator = (TVector&& other) {
if (&other == this) {
return *this;
}
delete[] storage;
storage = other.storage;
other.storage = nullptr;
size = other.size;
other.size = 0;
capacity = other.capacity;
other.capacity = 0;
return *this;

}

private:
std::size_t capacity = 0;
std::size_t size = 0;
T* storage = nullptr;

};

struct TElem {
std::uint64_t Key;
char Value[2049];
};

struct TSortElem {
std::uint64_t Key;
std::uint64_t Ind;
};


TVector<TSortElem> Sortvec(TVector<TSortElem>vec, std::uint64_t maxKey) {
std::uint64_t h = pow(10, 19);
int i, count[10] = { 0 };
int k = vec.sizeM();
TVector<TSortElem> res(k);
```

```
for (std::uint64_t exp = 1; maxKey / exp > 0 && exp <= h; exp *= 10) {
for (i = 0; i <= 9; i++)
count[i] = 0;
for (i = 0; i < k; i++) {
count[(vec[i].Key / exp) % 10]++;
}
for (i = 1; i < 10; i++)
count[i] += count[i - 1];
for (i = k - 1; i >= 0; i--) {
res[count[(vec[i].Key / exp) % 10] - 1] = vec[i];
count[(vec[i].Key / exp) % 10]--;
}
for (i = 0; i < k; i++) {
vec[i] = res[i];
}
if (exp == h)
break;
}
return vec;
}

int main() {
std::ios_base::sync_with_stdio(false);
std::cin.tie(NULL);
TVector<TElem> vec;
TElem temp;
TVector<TSortElem> vecSort;
TVector<TSortElem> Result;
TSortElem tempSort;
int i = 0;
std::uint64_t maxKey = 0;
while (std::cin >> temp.Key >> temp.Value) {
vec.AddLast(temp);
if (temp.Key > maxKey) {
maxKey = temp.Key;
}
tempSort.Key = vec[i].Key;
tempSort.Ind = i;
vecSort.AddLast(tempSort);
i++;
}

Result = Sortvec(vecSort, maxKey);

int M = vec.sizeM();

for (i = 0; i < M; i++) {
tempSort = Result[i];
std::cout << tempSort.Key << " " << vec[tempSort.Ind].Value << "\n";
```

```
181 ||
182 || }
183 || }
```

vector.h	
TVector()	Конструктор по умолчанию
explicit TVector(size_t newSize, const T& defaultValue = T()) : TVector()	Конструктор от двух аргументов: размер вектора и значения по умолчанию
TVector(const TVector& other) : TVector();	Конструктор копирования
TVector& operator=(TVector other)	Оператор присваивания с копированием
TVector()	Деструктор
size_t Size() const	Размер вектора
bool Empty() const	Проверка на пустоту
T* begin() const	Итератор на начало
T* end() const	Итератор за конец
static void Swap(TVector& lhs, TVector& rhs)	Обмен векторов значениями
void PushBack(const T& value)	Добавление элемента в конец
const T& At(size_t index) const	Получение константной ссылки на элемент по индексу
T& At(size_t index)	Получение ссылки на элемент по индексу
const T& operator[](size_t index) const	Получение константной ссылки на элемент по индексу
T& operator[](size_t index)	Получение ссылки на элемент по индексу

3 Консоль

```
(py37)  ~ cd da1
(py37)  ~ g++ -pedantic -Wall -Wextra main.cpp -o out
(py37)  ~ da1 ./out <test.txt >res.txt
(py37)  ~ da1 cat test.txt
0 xGfxrxGGxrxMMMMfrrrG
18446744073709551615  xGfxrxGGxrxMMMMfrrr
0 xGfxrxGGxrxMMMMfrr
18446744073709551615  xGfxrxGGxrxMMMMfr
(py37)  ~ da1 cat res.txt
0 xGfxrxGGxrxMMMMfrrrG
0 xGfxrxGGxrxMMMMfrr
18446744073709551615 xGfxrxGGxrxMMMMfrrr
18446744073709551615 xGfxrxGGxrxMMMMfr
```

4 Выводы

К сожалению, реализация алгоритма поразрядной сортировки оказалась не настолько эффективной, насколько хотелось бы. Тем не менее, работа оказалась достаточно занимательной: я ознакомился с азами использования утилиты *make*, утилитами для контроля утечек памяти, освоил новый Code-Style а так же вспонил, как работать в LaTeX-е.

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] *Поразрядная сортировка — Вики университета ITMO.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 01.10.2020).