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: Т. Д. Голубев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

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: `Автомобильные номера в формате A 999 BC (используются буквы латинского алфавита).`

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

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки.

Как сказано в [?]: «основная идея поразрядной сортировки в том, что число разбивается на цифры (разряды), а затем все числа сортируются по каждому разряду устойчивой сортировкой».

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру *TPair*, в которой будем хранить ключ и значение. Для неё перегрузим *operator»* для удобства считывания и *operator«* для вывода. Чтобы хранить пары, напишем класс *TVector*, который будет выполнять функцию динамического массива. Для данной задачи метод *PopBack* для вектора реализовывать не нужно. Напишем фукнцию поразрядной сортировки *RadixSort* и функцию сортировки подсчётом *CountingSort*, которая будет использоваться для сортровки разрядов.

main.cpp	
TVector<TPair> CountingSort(TVector<TPair>& array, int pos)	Функция сортировки подсчётом по букве с номером *pos* из ключа пары.
void RadixSort(TVector<TPair>& array)	Функция поразрядной сортировки.

```
template <class T>
class TVector {
private:
  int capacity;
  int size;
  T* data;
public:
  TVector(int size = 0);
  TVector(int size, T element);
  TVector(const TVector<T>& other) = delete;
  TVector(TVector<T>&& other);
  ~TVector();
  void PushBack(T& element);
  void PushBack(T&& element);
  T& operator[](int idx);
  const T& operator[](int idx) const;
  TVector<T>& operator=(const TVector<T>& other);
  TVector<T>& operator=(TVector<T>&& other) noexcept;
  int Size() const;
};

struct TPair {
  std::string key;
  std::string value;
  friend std::istream& operator>>(std::istream& is, TPair& rhs);
  friend std::ostream& operator<<(std::ostream& os, const TPair& rhs);
  TPair() = default;
  TPair(const TPair& other) = delete;
```

```
29 ||   TPair(TPair&& other);
30 ||   TPair& operator=(const TPair& other);
31 ||   TPair& operator=(TPair&& other) noexcept;
32 || };
```

3 Консоль

```
cat_mood@nuclear-box:~/programming/mai-da-labs/lab01$ make
g++ -std=c++20 -pedantic -Wall -Wextra -Wno-unused-variable main.cpp -o lab01
cat_mood@nuclear-box:~/programming/mai-da-labs/lab01$ cat tests/03.t
J 213 DF        QfR
A 335 MG        JeqOCF
G 124 EJ        fijFYoKLhcaSBGMnPt
O 451 YI        zL
Z 063 JW        c
W 632 QA        CREMkwavGJbb
O 760 UP        JncI
C 221 QC        okFceGcCGjzSzyNwkpH
B 047 WE        mlznVqGoWOwD
S 081 LO        PQMN
cat_mood@nuclear-box:~/programming/mai-da-labs/lab01$ ./lab01 <tests/03.t
A 335 MG        JeqOCF
B 047 WE        mlznVqGoWOwD
C 221 QC        okFceGcCGjzSzyNwkpH
G 124 EJ        fijFYoKLhcaSBGMnPt
J 213 DF        QfR
O 451 YI        zL
O 760 UP        JncI
S 081 LO        PQMN
W 632 QA        CREMkwavGJbb
Z 063 JW        c
```

4 Тест производительности

Тест производительности представляет из себя следующее: алгоритм поразрядной сортровки сравнивается с *std::stable_sort* на 7 тестах с разным количеством входных данных, входные данные из себя представляют случайный набор ключей и значений.

```
[info] [2024-03-10 21:21:15] Running tests/01.t
Count of lines is 0
Radix sort time: 7us
STL stable sort time: 0us
[info] [2024-03-10 21:21:15] Running tests/02.t
Count of lines is 1
Radix sort time: 8us
STL stable sort time: 1us
[info] [2024-03-10 21:21:15] Running tests/03.t
Count of lines is 10
Radix sort time: 22us
STL stable sort time: 9us
[info] [2024-03-10 21:21:15] Running tests/04.t
Count of lines is 100
Radix sort time: 147us
STL stable sort time: 134us
[info] [2024-03-10 21:21:15] Running tests/05.t
Count of lines is 1000
Radix sort time: 1315us
STL stable sort time: 1840us
[info] [2024-03-10 21:21:15] Running tests/06.t
Count of lines is 10000
Radix sort time: 13636us
STL stable sort time: 22433us
[info] [2024-03-10 21:21:15] Running tests/07.t
Count of lines is 100000
Radix sort time: 140472us
STL stable sort time: 279561us
```

Как видно, алгоритм поразрядной сортировки выигрывает по времени у *std::stable_sort*, так как её асимптотическая сложность $O(n \log n)$, когда у поразрядной сортировки – $O(kn)$, где k – количество разрядов. При выбранном варианте ключей (автомобильные номера) $k = 6$, поэтому сложность приближается к $O(n)$.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я реализовал алгоритм поразрядной сортировки. В процессе написания я столкнулся с проблемой Memory Limit, которую решал заменой операций копирования на операции перемещения.

Список литературы

[1] Кормен, Томас Х., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. *Алгоритмы: построение и анализ. – 2-е изд.* — М.: Издательский дом «Вильямс», 2011. — 1296 с.