Лабораторная работа № 1 по курсу дискретного анализа: сортировка за линейное время

Выполнил студент группы М8О-208Б-20 *Борисов Ян*.

# Условие

Кратко описывается задача:

1. Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.
2. Вариант задания: 7-1. Поразрядная сортировка. Тип ключа: автомобильные номера в формате A 999 BC (используются буквы латинского алфавита). Тип значения: строки фиксированной длины 64 символа.

# Метод решения

В самом начале работы программы в функции main создается std::vector, в который считываются пары ключ-значение. Для этих пар я создал структуру TObject. Если этот вектор не пуст, то вызывается функция поразрядной сортировки, в которую передается вектор данных. Поразрядная сортировка базируется на сортировке подсчетом, которая сортирует за линейное время числа по определенному разряду. Так как номер состоит из 7 индексов (индексация с нуля), то мы запускаем поразрядную сортировку с 7 разряда. После каждой итерации, массив предстает в новом обличии, отсортированном по определенному разряду. В заключение, в цикле мы выводим у каждого элемента массива два его поля: ключ и значение.

# Описание программы

Программа состоит из файла main.cpp.

Структура **TObject**:

struct TObject{  
 std::string key;  
 std::string value;  
};

Функция **void RadixSort(std::vector<TObject\*>& keyValues)**:

В функции создается массив count для подсчета вхождений каждого элемента и массив для отсортированной последовательности. Затем начиная с младшего разряда, я запускаю сортировку подсчетом для массива. Так же после сортировки подсчетом массив count обнуляется для сортировки следующего разряда.

int count[AMOUNT\_OF\_LETTERS] = {};  
std::vector<TObject\*> sorted(keyValues.size());  
for(short int i = AMOUNT\_OF\_DIGITS; i >= 0; --i){  
 if (keyValues[0]->key[i] == ' '){  
 continue;  
 }  
 else if(keyValues[0]->key[i] >= '0' && keyValues[0]->key[i] <= '9'){  
 CountSort(count, "number", keyValues, sorted, i);  
 }  
 else{  
 CountSort(count, "letter", keyValues, sorted, i);  
 }  
 for (short int j = 0; j < AMOUNT\_OF\_LETTERS; ++j) {  
 count[j] = 0;  
 }  
 std::swap(keyValues, sorted);  
}

Функция **void CountSort(int\* count, const std::string& flag, std::vector<TObject\*>& keyValues, std::vector<TObject\*>& sorted, short int rank)** реализует алгоритм сортировки подсчетом, при чем на вход она принимает int flag, который обозначает, разряд букв или цифр сейчас сортируется, и в зависимости от этого функция реализует свою логику: сначала подсчитывает количество вхождений каждого символа, затем увеличивает количество вхождений элемента на количество вхождений предыдущего элемента и расставляет элементы в сортированном порядке.

char c;  
if(flag == "number") {  
 c = '0';  
}  
else {  
 c = 'A';  
}  
for(int k = 0; k < keyValues.size(); ++k){  
 ++count[keyValues[k]->key[rank] - c];  
}  
if(flag == "number") {  
 for (short int j = 1; j < AMOUNT\_OF\_NUMBERS; ++j) {  
 count[j] += count[j - 1];  
 }  
}  
else {  
 for(short int j = 1; j < AMOUNT\_OF\_LETTERS; ++j){  
 count[j] += count[j - 1];  
 }  
}  
for(int m = keyValues.size(); m != 0; --m){  
 --count[keyValues[m - 1]->key[rank] - c];  
 sorted[count[keyValues[m - 1]->key[rank] - c]] = keyValues[m - 1];  
}

**Дневник отладки**

Присутствовали проблемы с производительностью, но move семантика решила данный вопрос.

# Тест производительности

Пусть n – количество входных данных. Тогда:

1. При n = 10000 время работы программы составляет ~ 0.028 секунд.
2. При n = 100000 время работы программы составляет ~ 0.18 секунд.
3. При n = 1000000 время работы программы составляет ~ 1.906 секунд.
4. При n = 10000000 время работы программы составляет ~ 23 секунд.

Из этих тестов видно, что программы работает за линейную сложность.

# Недочёты

Программа работает корректно только при правильных входных данных.

**Выводы**

Поразрядная сортировка может иметь большую область применения, так как она является устойчивой и её сложность O(d\*n), где d – количество разрядов ключа, а n – количество входных данных. Устойчивость алгоритма означает, что если на вход программе подаётся одинаковые ключи с разными значениями, то в конце сортировки их порядок не поменяется.

Можно представить такую задачу: есть очередь на запись в барбершоп в разное время и нужно отсортировать её по возрастанию времени. Клиенты записались к разным барберам на одно и то же время. Тогда, чтобы построить очередь, можно представить эти данные так: ключ – время, значение – ФИ. После применения поразрядной сортировки, мы получим отсортированные данные по времени с сохранением порядка.