Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу «Дискретный анализ»**

Студент: И. С. Глушатов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19
Дата:
Оценка:
Подпись:

Москва, 2020

# Лабораторная работа №1

**Задача:** Требуется разработать программу на языке C++, 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 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

Поразрядная сортировка - один из видов сортировок за линейное время. Общая сложность алгоритма $O(b * n)$, где $b$ - колличество разрядов в самом длинном числе, а $n$ - количество элементов. Предполагается, что $b$ намного меньше $n$ ($b << n$), только в этом случае данная сортировка обеспечивает настоящую эффективность.

Если сортируемые элементы - числа, то вместо сортировки по десятичным разрядам, реализуется сортировка по n'ому количеству битов. Внутри первого цикла, который проходится по всем разрядам, элементы сортируются с помощью сортировки подсчетом.

# 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. Так как аргументы структуры - базовые типы языка и массив символов определенной константной длины, деструктор нам не нужен. Для удобства написания кода я создал конструктор, хотя можно было обойтись и без него.

```
const int STRING_SIZE = 64;

struct TPair {
  unsigned long long key;
  char value[STRING_SIZE];
  TPair();
  TPair(const unsigned long long k, const char v[STRING_SIZE]);
};
```

В данной лабораторной работе запрещалось использовать вектор из стандартной библиотеки C++, поэтому пришлось реализовывать его самим. Для удобного обращения к элементам вектора я переопределил оператор [].
Константа $BASIC\_VECTOR\_BUFFER$ используется для метода $Append()$ - если вектор изначально пустой, то начальный размер буфера выставляется равным этой константе.

```
const int BASIC_VECTOR_BUFFER = 124;

template<typename T>
class TVector {
  private:
    int size;
    int buffer;
    T *massive;
  public:
    TVector();
    TVector(const int lenth);
    ~TVector();
    void Append(const T &element);
    void Resize(const int sz);
    void ChangeSize(const int sz);
    int Len();
    T& operator[](const int id);
};
```

Главный файл *main.cpp* представляет из себя ввод данных из стандартного входного потока с помощью цикла *while* и метода $std :: cin$. Далее следует функция поразрядной сортировки, после чего ответ выводится на экран.

```cpp
#include <iostream>
#include "pair.hpp"
#include "vector.hpp"

T Rank(const T number, int i);

void RadixSort(TVector<TPair> &massive);

int main() {

  TVector<TPair> vec;
  unsigned long long key;
  char value[STRING_SIZE];

  while (std::cin >> key >> value) {
    vec.Append(TPair(key, value));
  }

  RadixSort(vec);

  for (int i = 0; i < vec.Len(); ++i) {
    std::cout << vec[i].key << " " << vec[i].value << "\n";
  }

  return 0;
}
```

| main.cpp | |
|---|---|
| T Rank(const T number, int i) | Функция возвращает цифру i'ого разряда. |
| void RadixSort(TVector<TPair> & massive) | Функция поразрядной сортировки. |
| pair.cpp | |
| TPair::TPair(): key(0)<br>TPair::TPair(const unsigned long long k, const char v[STRING_SIZE]): key(k) | Конструкторы TPair. |
| vector.cpp | |
| TVector<T>::TVector(): size(0), buffer(0), massive(0)<br>TVector<T>::TVector(const int lenth): size(0), buffer(lenth)<br>TVector<T>::~ TVector() | Конструкторы и деструктор TVector. |
| void TVector<T>::Append(const T &element) | Функция вставки в вектор. |
| void TVector<T>::Resize(const int sz) | Изменяет размер вектора (если уменьшается, то элементы стираются). |
| void TVector<T>::ChangeSize(const int sz) | Изменяет размер вектора (не трогая буфер и элементы вообще). |
| int TVector<T>::Len() | Возвращает количество элементов вектора. |
| T& TVector<T>::operator[](const int id) | Переопределение оператора возврата элемента по индексу. |

# 3 Консоль

```
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ g++ -Werror -Wno-sign-compare
-Wall -Wextra -pedantic main.cpp
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ cat test0.txt
7 Sa
8 d
5 d
25 GPsIKqE
27 DSjYzxRa
23 RxBx
15 bknTDiRI
0 joFclaCc
19 ZdALsZ
15 NC
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ ./a.out <test0.txt
0 joFclaCc
5 d
7 Sa
8 d
15 bknTDiRI
15 NC
19 ZdALsZ
23 RxBx
25 GPsIKqE
27 DSjYzxRa
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$
```

# 4 Тест производительности

Тест производительности представляет из себя следующее: всего в тесте учавствуют четыре файла на 1000, 10000, 100000 и 1000000 строчек. С помощью библиотеки *chrono* я замеряю время начала и конца функции поразрядной сортировки и устойчивой из библиотеки *algorithm* ($std :: stable\_sort$).

```
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ ./a.out <test0.txt
Поразрядная: 0.0026
Стабильная из std: 0.00127379
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ ./a.out <test1.txt
Поразрядная: 0.00447372
Стабильная из std: 0.00436133
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ ./a.out <test2.txt
Поразрядная: 0.0450362
Стабильная из std: 0.0513525
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$ ./a.out <test3.txt
Поразрядная: 0.483629
Стабильная из std: 0.669035
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab1$
```

Из приведенных тестов видно, что после 10000 элементов поразрядная сортировка начинает выигрывать у устойчивой, временная сложность которой, из источников, $O(nlog^2n)$

# 5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился многим аспектам такого языка программмирования, как C++. Создал template класс, реализовал поразрядную сортировку. Попрактиковался в управлении памяти. Занимался переопределением операторов, что было самым интересным в создании нового класса, но, к сожалению, только один из четырех операторов оказался в итоге полезным. Учитывая разработку данной программы на разных компьютерах, я осознал, насколько важную роль играет железо в производительности, и это помогло мне в оптимизации. К слову, самая большая проблема была не в реализации классов и поразрядной сортировки, а медленные скорость работы этой самой сортировки и вывод ответа на экран пользователя. Эта лабораторная работа дала хорошее представление о работе на C++ и о том, насколько скрупулёзным надо быть при разработке проекта на этом языке.

# Список литературы


[1] *Поисковик - Google.*
URL: https://www.google.com/

[2] *Поразрядная сортировка — Algolist.*
URL: http://algolist.ru/sort/radix_sort.php

[3] *Про операторы C++ и их переопределение*
URL: https://en.cppreference.com/w/cpp/language/operators

[4] *Про шаблоны C++*
URL: https://docs.microsoft.com/ru-ru/cpp/cpp/templates-cpp?view=vs-2019