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: Е. А. Айрапетова
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
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: Сортировка подсчётом.

Вариант ключа: Числа от 0 до 65535.

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 сортировки подсчётом. В качестве ключа выступают числа от 0 до 65535.

На вход программе подается неизвестное заранее количество данных, которые ситываются с помощью цикла $while$. В этом же цикле определяется max — значение самого большого ключа из полученных на вход.

Как сказано в [1]: «основная идея сортировки подсчётом заключается в том, чтобы для каждого входного элемента x определить количество элементов, которые меньше x». Для этого создается вспомогательный массив $count[]$ размером max. Также для хранения отсортированыых данных создается массив $result[]$ равный по размеру исходному неотсортированному массиву $unsortedVector[]$.

Сложность сортировки подсчётом равна $O(n + max)$, где n — количество элементов.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $TPair$, в которой будут храниться эти пары:

```
#include <iostream>

struct TPair {
  unsigned short key;
  char value[65];
  TPair();
  TPair(int i, char* str);
  void Reset(char ch = '\0');
};
```

pair.h	
TPair()	Конструктор по умолчанию
TPair(int i, char* str)	Конструктор от двух аргументов: ключ и значение
void Reset(char ch = '\ 0')	Функция, заполняющая значение нулевыми символами

Также создадим шаблонный класс $TVector < T >$, в котором память выделяется динамически, так как мы не знаем заранее количество данных, которые программа получит на вход.

```

#include <iostream>

template <typename T>
class TVector {
public:
  using TValueType = T;
  using TIterator = TValueType *;
  TVector();
  TVector(unsigned int size);
  unsigned int Size() const;
  bool Empty() const;
  TIterator Begin() const;
  TIterator End() const;

  template <typename U>
    friend void VecSwap(TVector<U>& first, TVector<U>& second);

  TVector& operator=(TVector other);
  ~TVector();
  TValueType& operator[](int index) const;
```

```
22    void PushBack(TValueType& value);
23
24    template <typename U>
25      friend TVector<U> CountingSort(const TVector<U>& unsortedVector, unsigned int max);
26
27  private:
28    int storageSize;
29    int capacity;
30    TValueType *storage;
31  };
```

vector.h	
TVector()	Конструктор по умолчанию
TVector(unsigned int size)	Конструктор, выделяющий память под конкретное число элементов
unsigned int Size() const	Функция, возвращающая размер вектора
bool Empty() const	Функция, проверяющая, является ли вектор пустым
TIterator Begin() const	Функция, возвращающая указатель на начало вектора
TIterator End() const	Функция, возвращающая указатель на последний элемент вектора
template <typename U> friend void VecSwap(TVector<U>& first, TVector<U>& second);	Функция, меняющая два вектора местами
TVector& operator=(TVector other)	Перегрузка оператора присваивания
TVector()	Деструктор
TValueType& operator[](int index) const	Перегрузка оператора []
void PushBack(TValueType& value)	Функция добавления элемента в конец вектора

Функция сортировки подсчётом:

```
1  #include "vector.h"
2
3  template <typename U>
4  TVector<U> CountingSort(const TVector<U> &unsortedVector, unsigned int max) {
5    TVector<unsigned int> count{max + 1};
6    for (unsigned int i = 0; i <= max; i++) {
7      count[i] = 0;
8    }
9    for (unsigned int i = 0; i < unsortedVector.Size(); ++i) {
10     ++count[unsortedVector[i].key];
11   }
```

```
  for (unsigned int i = 1; i <= max; i++) {
    count[i] += count[i-1];
  }
  TVector<U> result{unsortedVector.Size()};
  for (int i = unsortedVector.Size() - 1; i >= 0; i--) {
    result[--count[unsortedVector[i].key]] = unsortedVector[i];
  }

  return result;
}
```

Функция для оценки времени работы [4]:

```
#include "pair.h"
#include "sort.h"
#include <chrono>
#include <algorithm>

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);

  TVector<TPair> vec;
  TPair p;
  unsigned int maxKey = 0;

    auto start = std::chrono::steady_clock::now();
  while(std::cin >> p.key >> p.value) {
    vec.PushBack(p);
    if (p.key > maxKey) {
      maxKey = p.key;
    }
    p.Reset();
  }
    auto finish = std::chrono::steady_clock::now();
  auto dur1 = finish - start;
  std::cerr << "input " << std::chrono::duration_cast<std::chrono::milliseconds>(dur1)
      .count() << " ms" << std::endl;

    std::cout << "Number of elems is " << vec.Size() << std::endl;

    start = std::chrono::steady_clock::now();
  TVector<TPair> sortedVector = CountingSort(vec, maxKey);
    // std::stable_sort(vector.Begin(), vector.End());
  finish = std::chrono::steady_clock::now();
  auto dur2 = finish - start;
  std::cerr << "sort " << std::chrono::duration_cast<std::chrono::milliseconds>(dur2).
      count() << " ms" << std::endl;

    start = std::chrono::steady_clock::now();
```

```
36      std::cout << sortedVector;
37      finish = std::chrono::steady_clock::now();
38    auto dur3 = finish - start;
39    auto dur = dur1 + dur2 + dur3;
40    std::cerr << "output " << std::chrono::duration_cast<std::chrono::milliseconds>(dur3
          ).count() << " ms" << std::endl;
41    std::cerr << "all " << std::chrono::duration_cast<std::chrono::milliseconds>(dur).
          count() << " ms" << std::endl;
42    return 0;
43  }
```

Основная функция, в которой происходит считывание данных, а также вывод результата:

```
1   #include "pair.h"
2   #include "sort.h"
3   #include <algorithm>
4
5   int main() {
6     std::ios::sync_with_stdio(false);
7     std::cin.tie(nullptr);
8
9     TVector<TPair> vec;
10    TPair p;
11      unsigned int maxKey = 0;
12
13    while(std::cin >> p.key >> p.value) {
14      vec.PushBack(p);
15      if (p.key > maxKey) {
16        maxKey = p.key;
17      }
18      p.Reset();
19    }
20
21    TVector<TPair> sortedVector = CountingSort(vec, maxKey);
22      std::cout << sortedVector;
23    return 0;
24  }
```

3 Консоль

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР1/solution$ make
g++ -std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable -c main.cpp -o
main.o
g++ -std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable main.o -o solution
jane@Evgenia:/mnt/c/Files/ДА/ЛР1/solution$ ./solution
5       qqqqqqqqqqqqqqqqqqwefff3]
1       erwefwggggwrgwrgw0.,3
14      fjjdwe
1       mm
1       111111111
14      6erij222pfj[][wggg
1       rrrrrrrrrrrrrrrrrrrr

1       erwefwggggwrgwrgw0.,3
1       mm
1       111111111
1       rrrrrrrrrrrrrrrrrrrr
5       qqqqqqqqqqqqqqqqqqwefff3]
14      fjjdwe
14      6erij222pfj[][wggg
```

4 Тест производительности

Тест производительности представляет из себя следующее: скорость моей реализации сортировки подсчетом сравнивается с устройчивой сортировкой, взятой из библиотеки $C++$ $std::stable\_sort()$. Тест состоит из 1000 строк.

Моя реализация:

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР1/solution$ make benchmark
g++ -std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable -c benchmark.cpp
-o benchmark.o
g++  benchmark.o -o benchmark
jane@Evgenia:/mnt/c/Files/ДА/ЛР1/solution$ ./benchmark <test.txt >output.txt
input 6 ms
sort 7 ms
output 1 ms
all 14 ms
```

Стабильная сортировка:

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР1/solution$ make benchmark
g++ -std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable -c benchmark.cpp
-o benchmark.o
g++  benchmark.o -o benchmark
jane@Evgenia:/mnt/c/Files/ДА/ЛР1/solution$ ./benchmark <test.txt >output.txt
input 6 ms
sort 4 ms
output 1 ms
all 11 ms
```

Как видно, стабильная сортировка работает несколько быстрее. Это обусловлено тем, что в моей программе довольно много времени занимает копирование данных из одного вектора в другой. Скорость работы программы можно бы было усовершенствовать засчет создания отдельного вектора, в который записывался бы конечный результат, но это заняло бы дополнительную память.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я применила свои теоретические знания по сортировкам на практике, научилась работать с инструментами отладки памяти (в частности, *valgrind*) и утилитой *make*, попрактиковалась в написании программ на языке $C++$, освоила некоторые тонкости этого языка, а также вспомнила, как работать в $LaTeX$.

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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2020).

[3] *Шаблоны C++ — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Шаблоны_C%2B%2B` (дата обращения: 16.12.2020).

[4] *Chrono.*
URL: `https://www.cplusplus.com/reference/chrono/` (дата обращения: 18.12.2020).