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: К. С. Саженов
Преподаватель: Н. С. Капралов
Группа: М8О-208Б

Дата: 5 октября 2020 г.
Оценка:
Подпись:

Москва, 2020

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: Карманная сортировка.

Вариант ключа: `Числа от 0 до` $2^{64} - 1$.

Вариант значения: `Числа от 0 до` $2^{64} - 1$.

1 Описание

Требуется написать реализацию алгоритма карманной сортировки.

Как сказано в [2]: «карманная сортировка – алгоритм сортировки, основанный на предположении о равномерном распределении входных данных.».

Идея карманной сортировки состоит в том, чтобы разбить исходные возможные значения в массиве на n равных интервалов, где n – количество элементов в массиве, а затем пройтись по массиву, перемещая в созданные интервалы(называемые карманами), элементы, которые в них попадают, а карманы, в свою очередь, отсортировать обычной сортировкой вставки.

2 Исходный код

Написание кода состояло из нескольких этапов:

1. Прописать объявления основных функций, классов, методов(функций-членов) и полей класса
2. Написать прототип функции $main$, а именно:
   (a) Ввод-вывод данных
   (b) Запуск сортировки
3. Реализовать основные функции и функции-члены для работы с векторами
4. Реализовать сортировку вставками
5. Реализовать карманную сортировку(используя сортировку вставками)
6. Написать Makefile

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новые структуры $TKey$ и $TValue$, которые будут являться псевдонимами для входных типов данных(это сделано для универсальности кода), также, т.к. тип $NPair :: TPair < NSort :: TKey, NSort :: TValue >$ является слишком длинным, я решил сделать для него псевдоним $TData$.

Файл `main.cpp`:

```
1  #include <iostream>
2  #include <algorithm>
3
4  #include "vector.h"
5  #include "pair.h"
6  #include "sort.h"
7
8
9  using TData = NPair::TPair<NSort::TKey, NSort::TValue>;
10
11 int main() {
12   std::ios::sync_with_stdio(false);
13   std::cin.tie(nullptr);
14   std::cout.tie(nullptr);
15
16   NVector::TVector<TData> vector;
17
18   NSort::TKey key;
```

```
  NSort::TValue value;
  while(std::cin >> key >> value){
    vector.PushBack(NPair::TPair<NSort::TKey, NSort::TValue>(key, value));
  }

  NSort::BucketSort(vector);

  for (int i = 0; i < vector.Size(); ++i) {
    std::cout << vector[i].First << ' ' << vector[i].Second << std::endl;
  }

  return 0;
}
```

Файл `vector.h`:

```
#pragma once

#include <stdexcept>
#include <cstring>


namespace NVector {
  const int DEFAULT_CAPACITY_MULTIPLIER = 2;

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

    explicit TVector(size_t newSize, const T &defaultValue = T());

    [[nodiscard]] size_t Size() const;

    [[nodiscard]] bool Empty() const;

    [[maybe_unused]] T *Begin() const;

    [[maybe_unused]] T *End() const;


    TVector(const TVector &other);

    TVector &operator=(const TVector &other);

    ~TVector();

    void Clear();

    [[maybe_unused]] void PushBack(T &element);
```

```
35
36        void PushBack(const T &&element);
37
38        const T &At(size_t index) const;
39
40        T &At(size_t index);
41
42        const T &operator[](size_t index) const;
43
44        T &operator[](size_t index);
45
46      private:
47        size_t Len{};
48        size_t Capacity{};
49        T *Arr;
50      };
51    }
```

Файл pair.h:

```
1  //
2  // Created by sakost on 02.10.2020.
3  //
4
5  #ifndef LAB1_PAIR_H
6  #define LAB1_PAIR_H
7
8
9  namespace NPair {
10
11     template<typename F, typename S>
12     class TPair {
13     public:
14         TPair();
15
16         TPair(F first, S second);
17
18         TPair(const TPair &other);
19
20         ~TPair()= default;
21
22         bool operator<(const TPair &other) const;
23
24         TPair &operator=(const TPair &other);
25
26
27         F First;
28         S Second;
29     };
30 }
```

```
31
32
33 #endif //LAB1_PAIR_H
```

Файл sort.h

```
1  //
2  // Created by sakost on 02.10.2020.
3  //
4
5  #ifndef LAB1_SORT_H
6  #define LAB1_SORT_H
7
8  #include <cinttypes>
9  #include <limits>
10 #include <cassert>
11
12 #include "vector.h"
13 #include "pair.h"
14
15 namespace NSort{
16     using TKey = std::uint64_t;
17     using TValue = std::uint64_t;
18
19     NPair::TPair<TKey, TValue> MaxElement(const NVector::TVector<NPair::TPair<TKey,
           TValue>> &vector);
20     NPair::TPair<TKey, TValue> MinElement(const NVector::TVector<NPair::TPair<TKey,
           TValue>> &vector);
21
22     void InsertionSort(NVector::TVector<NPair::TPair<TKey, TValue>> &vector);
23
24     void BucketSort(NVector::TVector<NPair::TPair<TKey, TValue>> &vector);
25 }
26
27
28 #endif //LAB1_SORT_H
```

Файл benchmark.cpp

```
1  //
2  // Created by sakost on 03.10.2020.
3  //
4
5
6  #include "pair.h"
7  #include "vector.h"
8  #include "sort.h"
9
10 #include <iostream>
11 #include <cstdint>
12 #include <chrono>
```

```cpp
#include <algorithm>
#include <random>

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

    NVector::TVector<NPair::TPair<NSort::TKey , NSort::TValue>> v;
    NPair::TPair<NSort::TKey , NSort::TValue> pair;


    auto start = std::chrono::steady_clock::now();
    NSort::TKey key;
    NSort::TValue value;
    while (std::cin >> key >> value) {
        pair.First = key;
        pair.Second = value;
        v.PushBack(pair);
    }
    auto finish = std::chrono::steady_clock::now();
    auto dur = finish - start;
    std::cerr << "input " << std::chrono::duration_cast<std::chrono::milliseconds>(dur)
        .count() << " ms" << std::endl;

    start = std::chrono::steady_clock::now();
    NSort::BucketSort(v);
    finish = std::chrono::steady_clock::now();
    dur = finish - start;
    std::cerr << "custom bucket sort " << std::chrono::duration_cast<std::chrono::
        milliseconds>(dur).count() << " ms" << std::endl;

    std::random_device rd;
    std::mt19937 g(rd());
    std::shuffle(v.Begin(), v.End(), g);

    start = std::chrono::steady_clock::now();
    std::stable_sort(v.Begin(), v.End());
    finish = std::chrono::steady_clock::now();

    dur = finish - start;
    std::cerr << "stable sort from std " << std::chrono::duration_cast<std::chrono::
        milliseconds>(dur).count() << " ms" << std::endl;

    start = std::chrono::steady_clock::now();
    for (size_t i = 0; i < v.Size(); i++) {
        std::cout << v[i].First << ' ' << v[i].Second << '\n';
    }
    finish = std::chrono::steady_clock::now();
```

```
59      dur = finish - start;
60      std::cerr << "output " << std::chrono::duration_cast<std::chrono::milliseconds>(dur
            ).count() << " ms" << std::endl;
61
62      return 0;
63  }
```

Таблица 1, описывающая структуру программы:

vector.h	
class TVector	Класс, реализующий тип данных «вектор»
TVector()	Конструктор по умолчанию
explicit TVector(size_t newSize, const T &defaultValue = T())	Конструктор с заданием размера или и размера, и элемента по умолчанию
size_t Size() const	Функция-член, возвращающая размер вектора
bool Empty() const	Функция-член, возвращающая значение, которое указывает, пустой ли вектор
T *Begin() const	Функция-член, возвращающая указатель на первый элемент вектора в памяти
T *End() const	Функция-член, возвращающая указатель на элемент, следующий за последним, в векторе в памяти
TVector(const TVector &other)	Конструктор копирования
TVector &operator=(const TVector &other)	Оператор копирования
~ TVector()	Деструктор класса
void Clear()	Функция-член очистки объекта
void PushBack(T &element)	Функция-член добавления элемента в конец вектора(по ссылке)
void PushBack(T &&element)	Функция-член добавления элемента в конец вектора(по r-value)
const T &At(size_t index) const	Константная функция-член доступа к элементам данной коллекции
T &At(size_t index)	Функция-член доступа к элементам данной коллекции
const T &operator[](size_t index)const	Константный оператор доступа к элементам данной колекции(аналогично `At`)

T &operator[](size_t index)	Оператор доступа к элементам данной колекции(аналогично `At`)
pair.h	
class TPair	Класс, реализующий тип данных «пара»
TPair()	Конструктор по-умолчанию
TPair(F first, S second)	Конструктор, явно указывающий элементы пары
TPair(const TPair &other)	Конструктор копирования
~ TPair() = default	Деструктор класса(по-умолчанию)
bool operator<(const TPair &other) const	Оператор сравнения «меньше»
TPair &operator=(const TPair &other)	Оператор копирования
F first	Первый элемент пары
S second	Второй элемент пары
sort.h	
TKey	Тип ключа
TValue	Тип значения
NPair::TPair<TKey, TValue> MaxElement(const NVector::TVector<NPair::TPair<TKey, TValue» &vector)	Возвращает максимальный элемент вектора(по ключу)
NPair::TPair<TKey, TValue> MinElement(const NVector::TVector<NPair::TPair<TKey, TValue» &vector)	Возвращает минимальный элемент вектора(по ключу)
void BucketSort(NVector::TVector< NPair::TPair< TKey, TValue>> &vector)	Карманная сортировка
void InsertionSort(NVector::TVector< NPair::TPair<TKey, TValue >> &vector)	Сортировка вставками

3 Консоль

```
sakost$ g++ -pedantic -Wall -std=c++17 -Werror -Wno-sign-compare lab1.cpp vector.h
pair.h sort.h -o lab1
sakost$ cat test1
1 4
2 7
1 8
sakost$ ./lab1 <test1
1 4
1 8
2 4
```

4 Тест производительности

Тест производительности представляет из себя следующее: имеется 100000 пар «ключ-значение» и осуществляется ввод этих данных в программу, затем происходит сортировка с помощью карманной сортировки:

```
sakost $ make benchmark
g++ -std=c++17 -O3 -c benchmark.cpp -o benchmark.o
g++ benchmark.o -o benchmark

sakost $./benchmark <tests/06.t >/dev/null
input 124 ms
custom bucket sort 36 ms
stable sort from std 24 ms
output 28 ms

sakost $./benchmark <tests/07.t >/dev/null
input 601 ms
custom bucket sort 578 ms
stable sort from std 205 ms
output 269 ms

sakost $ ./benchmark <tests/08.t >/dev/null
input 5413 ms
custom bucket sort 6519 ms
stable sort from std 2976 ms
output 2741 ms
```

std :: *stable_sort* чуть более, чем в два раза быстрее карманной сортировки, написанной мною. Данный разрыв можно объяснить тем, что стандартная сортировка работает в более оптимизированном варианте, ориентированном на скорость, а не на просто реализацию, как в моей версии. Также в карманной сортировке происходит множество операций копирования и вставки(в конец вектора), чего нельзя сказать о стандартной сортировке, в которой все сортируется in-place.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать классы из STL, понял, как работают те, или иные сортировки, в частности «карманная сортировка». Я научился отлаживать утечки памяти, создавать проект, уже похожий на те, которые создаются в больших компаниях. Также меня ещё больше заинтересовала реализация стандартного вектора, который я буду изучать в ближайшее время.

Список литературы

[1] Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. *Алгоритмы: построение и анализ, 2-е издание.*, — Издательский дом «Вильямс», 2005, стр. 220-239, глава 8, «Сортировка за линейное время»

[2] *Карманная сортировка - Викиконспекты Университета ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Карманная_сортировка` (дата обращения: 03.10.2020)

[3] *Блочная сортировка — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Блочная_сортировка` (дата обращения: 03.10.2020).