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: И. О. Ильин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата: 01.10.2020
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: Поразрядная сортировка.

Вариант ключа: `даты в формате DD.MM.YYYY`.

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

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки. В качестве ключа выступают даты в формате DD.MM.YYYY.

Как сказано в [2]: «По аналогии с разрядами чисел будем называть элементы, из которых состоят сортируемые объекты, разрядами. Сам алгоритм состоит в последовательной сортировке объектов какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего разряда к старшему, после чего последовательности будут расположены в требуемом порядке».

Сортировать разряды будем сортировкой подсчётом. На лекции Н. К. Макарова было выведено, что сложность такой сортировки будет $O(\frac{b}{R}(n + 2^R))$, где b – длина числа, R – длина разряда, n – количество чисел. Однако, данная сложность относится к сортировке двоичных чисел, тогда как в ключе содержаться десятичные. В плане реализации удобней пользоваться десятичной системой счисления, тогда сложность алгоритма сортировки составит $O(\frac{b}{R}(n + 10^R))$.

Так как на вход подаются даты, то $b = 8$. Тогда R может принимать значения равные $1, 2, 4, 8$. Рассмотрев все случаи было обнаружено, что наименьшая сложность в рамках данной задачи достигается при $R = 4$, тогда сортировка будет производится по разрядам DDMM и YYYY.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $NSort :: TKeyValue$, в которой будем хранить ключ и значение. Так как ключ является датой, то создадим класс $NDate :: TDate$.

NDate::TDate включает в себя приватные приватные поля *date, month, year, dateString*, которые представляют из себя целые числа: день, месяц, год и строку даты в естественном представлении. Также класс *TDate* имеет методы получения, изменения атрибутов, а также два конструктора: *TDate()* – конструктор по-умолчанию, создаёт объект с нулевой датой *"0.0.0"*, *TDate(std::string inputLine)* – создаёт объект из строки с датой. Метод *void SetDate(const std::string newDate)* изменяет объект, в соответствии с входной строкой. Также, были реализованы перегрузки операторов ввода и вывода для потоков: *std::ostream operator<<(std::ostream& output, const TDate& date)* и *std::istream operator>>(std::istream& input, TDate& date)*

Для хранения объектов структуры $TKeyValue$ был написан класс *TVector* в *namespace NVector*. Имеет приватные поля *size_t bufferSize, size_t bufferUsed, T *buffer*: размер буфера, размер заполненной части и указатель на тип T вектора (динамический массив). Описания методов вектора приведены в таблице ниже. Стоит отметить, что метод *void PushBack(const T & value)* изменяет размер буфера следующем образом: новый размер становится равным 1, если до этого он был нулевым, либо увеличивается в 2 раза. А метод *T PopBack()* делает размер буфера равным нулю, если до этого размер заполненной части был равен 1, либо уменьшает его в 2 раза, если размер заполненной части становится ровно в 2 раза меньше размера буфера, после удаления последнего элемента вектора.

Функция поразрядной сортировки имеет сигнатуру *void NSort::RadixSort(NVector::TVector<TKeyValue>& data, TDate& maxKey)*, где *maxKey* представляет собой ключ с максимальными значениями разрядов, *data* – входной вектор. Производит сортировку, согласно раннее описанному алгоритмту. В конце производится копирование из результирующего вектора *result* во входной *data*.

В функции *main* сначала отключаем синхронизацию между стандартными потоками *C* и *C++* используя *std::ios_base::sync_with_stdio(false)*, что позволяет ускорить ввод. Использование этого вызова целесообразно, поскольку в рамках решения данной задачи для ввода данных использовались только потоки *C++*.

Затем, создаём *NSort::TKeyValue elem*, который будет хранить в себе введенную пару «ключ-значение», также *TDate maxKey*, который будет хранить в себе максимальные день, месяц, год и вектор *NVector::TVector<NSort::TKeyValue> elems* для хранения введённых данных.

В цикле *while* вводим ключ и значение в *elem*, находим *TDate maxKey* и добавляем текущую пару «ключ-значение» в вектор. Затем, сортируем с помощью *NSort::RadixSort(elems, maxKey)* и выводим результат в цикле *for*.

sort.hpp	
void RadixSort(NVector::TVector<TKeyValue>& data, NDate::TDate& maxKey)	Функция поразрядной сортировки
date.hpp	
TDate()	Конструктор по умолчанию.
TDate(std::string& inputLine)	Конструктор, использующий строку с датой
int GetDigit(const int& idx)	Метод получения разряда даты
int GetDay() const	Метод получения дня
int GetMonth() const	Метод получения номера месяца
int GetYear() const	Метод получения год
std::string GetDate() const	Метод получения строки даты
void SetDay(const int& newDay)	Метод изменения дня
void SetMonth(const int& newMonth)	Метод изменения месяца
void SetYear(const int& newYear)	Метод изменение год
void SetDate(const std::string& newDate)	Метод изменение объекта в соответствии со входной строкой
std::ostream& operator<<(std::ostream& output, const NDate::TDate& date)	Перегруженный оператор вывода для объекта класса *TDate*
std::istream& operator>>(std::istream& input, NDate::TDate& date)	Перегруженный оператор ввода для объекта класса *TDate*
vector.hpp	
TVector()	Конструктор по умолчанию
TVector(size_t size)	Конструктор, задающий размер вектора
TVector(size_t size, const T & initial)	Конструктор, который создаёт вектор определенного размера, заполненный элементами переданного значения
TVector(const TVector<T> & vector)	Конструктор, создающий вектор из другого вектора
TVector()	Деструктор
T* Begin()	Указатель на начало вектора
T* End()	Указатель на конец вектора
size_t Capacity() const	Метод получения размера буфера вектора
size_t Size() const	Метод получения размер заполненной части вектора
bool Empty() const	Метод проверки вектора на пустоту

void PushBack(const T & value)	Метод добавление элемента в конец вектора
T & PopBack()	Метод удаление последнего элемента с конца вектора, возвращает его значение
T & operator[](size_t idx)	Метод изменение элемента на заданной позиции
T operator[](size_t idx) const	Метод получения значения элемента на заданной позиции
void Copy(TVector<T> & vector)	Метод, копирующий вектор, в текущий вектор
benchmark.cpp	
bool operator<(const NSort::TKeyValue& lhs, const NSort::TKeyValue& rhs)	Оператор сравнения для двух элементов вектора

```
namespace NSort {
    struct TKeyValue {
        NDate::TDate key;
        unsigned long long value;
    };

    const int keyLength = 8;
    const int digitLength = 4;
} // namespace NSort

namespace NDate {
    class TDate {
    public:
        TDate();
        TDate(std::string& inputLine);
        ~TDate();

        int GetDigit(const int& idx);
        int GetDay() const;
        int GetMonth() const;
        int GetYear() const;
        std::string GetDate() const;

        void SetDay(const int& newDay);
        void SetMonth(const int& newMonth);
        void SetYear(const int& newYear);
        void SetDate(const std::string& newDate);
    private:
        int day;
        int month;
        int year;
```

```
32          std::string dateString;
33      };
34  } // namespace NDate
35
36  namespace NVector {
37      template<class T>
38      class TVector{
39      public:
40          TVector();
41          TVector(size_t size);
42          TVector(size_t size, const T & initial);
43          TVector(const TVector<T> & vector);
44          ~TVector();
45
46          T* Begin();
47          T* End();
48
49          size_t Capacity() const;
50          size_t Size() const;
51          bool Empty() const;
52
53          void PushBack(const T & value);
54          T PopBack();
55          T & operator[](size_t idx);
56          T operator[](size_t idx) const;
57          void Copy(TVector<T> & vector);
58
59      private:
60          size_t bufferSize;
61          size_t bufferUsed;
62          T *buffer;
63  };
```

3 Консоль

```
MacBook-Pro:lab1 mr-ilin$ make
g++ -std=c++11 -pedantic -Wall -c main.cpp -o main.o
g++ -std=c++11 -pedantic -Wall -c date.cpp -o date.o
g++ -std=c++11 -pedantic -Wall -c sort.cpp -o sort.o
g++ -std=c++11 -pedantic -Wall main.o date.o sort.o -o solution
MacBook-Pro:lab1 mr-ilin$ cat tests/01.t
12.3.3009       14777361747327323373
29.11.2872      17604535573458183758
12.11.2707      3603119951011250069
8.5.2086        16871884052628989897
25.7.1966       6910674171499412178
21.9.2506       10579329736579662162
2.8.2053        2325070095917824646
25.7.1966       7126833875939297803
30.4.2976       4015673070521422780
29.11.2495      5546458575163770494
MacBook-Pro:lab1 mr-ilin$ ./solution <tests/01.t
25.7.1966       6910674171499412178
25.7.1966       7126833875939297803
2.8.2053        2325070095917824646
8.5.2086        16871884052628989897
29.11.2495      5546458575163770494
21.9.2506       10579329736579662162
12.11.2707      3603119951011250069
29.11.2872      17604535573458183758
30.4.2976       4015673070521422780
12.3.3009       14777361747327323373
```

4 Тест производительности

Тест производительности представляет из себя следующее: сортировку 1 миллионах входных данных с помощью реализованной поразрядной сортировки и *std::stable_sort*. При этом, измерения времени ввода и вывода данных не производятся.

```
MacBook-Pro:lab1 mr-ilin$ make benchmark
g++ -std=c++11 -pedantic -Wall -c benchmark.cpp -o benchmark.o
g++ -std=c++11 -pedantic -Wall -c sort.cpp -o sort.o
g++ -std=c++11 -pedantic -Wall -c date.cpp -o date.o
g++  sort.o date.o benchmark.o -o benchmark
MacBook-Pro:lab1 mr-ilin$ ./benchmark <one_million/01.t
Radix Sort Time: 0.347304
STL Stable Sort Time: 0.707575
```

Очевидно, что поразрядная сортировка сортирует быстрее, чем *stable_sort*, поскольку поразрядная сортировка относится к линейным сортировкам, а *stable_sort* имеет временную сложность $O(n * log(n))$.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать алгоритм линейной поразрядной сортировки. В ходе выполенения данной работы я стокнулся с несколькими проблемами, постараюсь описать каждую.

Изначально, для обработки строки "DD.MM.YYYY"использовалась функция *sscanf*, в дальнейшем обнаружил, что скорость ее выполнения не удволетворяет поставленным рамкам, поэтому обработку входной строки переписал вручную, что уменьшило время выполенения на 0.05 с на 1 миллионе входных данных.

Также, научился использовать классы на практике, в том числе, не с первой попытки, получилось реализовать достаточно быстрый вектор.

Следующем, на пути ускорения программы, было переосмысление алгоритма сортировки. Было выведено, что быстрее всего поразрядная сортировка работает, если разбивать дату не на 3 разряда (DD, MM, YYYY), а на 2 (DDMM, YYYY), что тоже помогло ускорить выполнение программы.

Пригодились навыки олимпиадного программирования, связанные с ускорением ввода данных. Ограничился выключением синхронизации стандартных потоков *C* и *C++*.

В ходе тестирование программы овладел базовыми навыками программирования на *Python*, для редактирования генератора тестов. Также научился тестировать скорость выполнения программы с помощью *std::chrono* и сравнивать её со встроенными алгоритмами сортировок.

Научился тому, как можно использовать свой контейнер в *std* сортировках.

Для себя сделал выводы, что необходимо сразу структурировать свою программу на отдельные файлы, чтобы потом было легче переписывать код. Также, в самом начале, необходимо основательно обдумывать все производимые действия и оценивать сложность их выполнения, это позволит на конечно этапе избежать поисков путей ускорения программы.

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] *Поразрядная сортировка — Вики университета ITMO.* URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 31.09.2020).