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. М. Семенов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-18

Дата:
Оценка:
Подпись:

Москва, 2019

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: `Почтовые индексы(000000 - 999999).`

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

1 Описание

Необходимо реализовать алгоритм сортировки подсчётом. Как известно любой алгоритм сортировки, в котором данные о порядке элементов извлекаются из непосредственного сравнения элементов, требует $\Omega(nlgn)$.

> Для доказательства данного утверждения достаточно определить высоту дерева, в котором каждая перестановка представлена достижимым листом.[1]

Сортировка подсчётом не использует непосредственное сравнение элементов, вместо этого, предполагается, что ключи, по которым производится сортировка, принадлежат некоторому множеству $0..k$ целых чисел. Алгоритм сортировки подсчётом основан на подсчёте количества элементов во входном массиве, которые меньше или равны данному. Благодаря этому, если $k = o(N)$, то сложность всего алгоритма становится равна $O(N)$. Также, данный алгоритм можно немного модифицировать, уменьшив количество ключей для подсчёта. Это можно сделать, если обобщить алгоритм на произвольный целочисленный диапазон и определить минимальный и максимальный ключи из тех, что попадаются во входных данных.

> Обобщение происходит следующим образом : если min больше нуля, то следует при работе с массивом C из A[i] вычитать min, а при обратной записи прибавлять. При наличии отрицательных чисел нужно при работе с массивом C к A[i] прибавлять |min|, а при обратной записи вычитать.[2]

2 Исходный код

1 Описание программы

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим структуру $TSortType$, в которой будем хранить ключ и значение. Для хранения ключа подойдет почти любой целочисленный тип, а вот для значения необходимо выбрать 64-битный беззнаковый(unsigned long long или uint64_t). В целях повышения удобства работы со структурой, реализуем перегрузки операторов ввода/вывода.
Экземпляры структуры $TSortType$ перед сортировкой необходимо поместить в какой-нибудь контейнер, позволяющий обратиться к произвольному элементу за $O(1)$. Для этого напишем шаблон класса $TVector$, который будет предоставлять примерно тот же функционал, что и $std :: vector$.
Далее, рассмотрим функцию для сортировки подсчётом. Функция принимает вектор структур «ключ-значение» по константной ссылке, минимальное и максимальное значения переданных ключей, а возвращает новый, отсортированный вектор. Внутри функции, создаются два вектора - результирующий вектор структур $TSortType$ и временный вектор целых чисел, используемый для подсчета ключей. Последний сначала инициализируется нулями. После этого, в цикле $for(i)$ просматривается каждый элемент входного массива. Если значение данного элемента равно i, то значение временного массива в ячейке $i - minKey$ увеличивается на 1. Далее, для каждого $i = 0, 1...maxKey - minKey$ с помощью суммирования определяется, сколько элементов входного массива не превышают $i + minKey$. Наконец, в соответствии со значениями, хранящимися во временном векторе, заполняется результирующий массив, который и возвращается из функции.

2 Таблица функций и методов

main.cpp	
TVector<TSortType> countingSort(const TVector<TSortType>& v, int maxKey, int minKey)	Функция сортировки подсчётом
std::istream& operator » (std::istream& is, TSortType& elem)	Перегрузка оператора ввода для TSortType
std::ostream& operator « (std::ostream& is, TSortType& elem)	Перегрузка оператора вывода для TSortType
int main()	Обрабатывает ввод, создавая вектор из структур, созданных на основе входных данных. Вызывает функцию сортировки. Осуществляет вывод результирующего вектора.
main.cpp, template<typename T> class TVector	
TVector() = default;	Конструктор без параметров
TVector(size_t newSize)	Конструктор, принимающий размер нового вектора
TVector(size_t newSize, T defaultVal)	Конструктор, принимающий размер и значение по умолчанию
TVector(const TVector& other)	Конструктор копирования, копирует значения из переданного вектора
TVector(TVector&& other)	Конструктор перемещения, перемещает в создаваемый объект указатель из other, в other указатель становится nullptr
TVector()	Деструктор, освобождает выделенную память.
T& operator[] (size_t index)	Оператор доступа по индексу, возвращает data[index].
const T& operator[] (size_t index) const	Константная версия оператора доступа по индексу.
void PushBack(const T& elem)	Добавляет elem в конец вектора, при необходимости совершая реаллокацию памяти

T* begin()	Возвращает указатель на память, хранящуюся внутри вектора, название данной и трёх последующих функций не соответствует codestyle из-за того, что функции begin() и end() используются в range-based for.
T* end()	Возвращает указатель на хранящуюся память, увеличенный на размер вектора.
const T* begin() const	Константная версия метода begin().
const T* end() const	Константная верстия метода end().
TVector& operator=(const TVector& other)	Копирующий оператор присваивания, возвращает ссылку на текущий объект.
TVector& operator=(TVector&& other)	Перемещающий оператор присваивания, перемещает указатель на память из other, указатель в other становится nullptr. Возвращает ссылку на текущий объект.
void ShrinkToFit()	Сжимает кусок памяти, которым владеет вектор до его реального размера.
size_t Size() const	Возвращает размер вектора.

3 Код программы

```
#include <iostream>
#include <iomanip>
#include <algorithm>

template <typename T>
class TVector {
public:
    TVector() = default;
    TVector(size_t newSize);
    TVector(size_t newSize, T defaultVal);
    TVector(const TVector& other);
    TVector(TVector&& other);
    ~TVector();
    T& operator[] (size_t index);
    const T& operator[] (size_t index) const;
    void PushBack(const T& elem);
    T* begin();
    T* end();
    const T* begin() const;
```

```
20      const T* end() const;
21      TVector& operator=(const TVector& other);
22      TVector& operator = (TVector&& other);
23      void ShrinkToFit();
24      size_t Size() const;
25
26  private:
27      T* data = nullptr;
28      size_t size = 0;
29      size_t capacity = 0;
30  };
31
32  struct TSortType {
33      int key;
34      uint64_t value;
35  };
36
37  std::istream& operator >> (std::istream& is, TSortType& elem) {
38      return is >> elem.key >> elem.value;
39  }
40
41  std::ostream& operator << (std::ostream& is, TSortType& elem) {
42      return is << std::setw(6) << std::setfill('0') << elem.key << '\t' << elem.value;
43  }
44
45  TVector<TSortType> CountingSort(const TVector<TSortType>& v, int minKey, int maxKey) {
46      if(v.Size() == 0) {
47          return {};
48      }
49      TVector<uint64_t> counts(maxKey - minKey + 1, 0);
50      for (TSortType element : v) {
51          counts[element.key - minKey]++;
52      }
53      for (size_t i = 1; i < counts.Size(); ++i) {
54          counts[i] += counts[i - 1];
55      }
56      TVector<TSortType> result(v.Size());
57      for (size_t i = v.Size(); i > 0; --i) {
58          result[--counts[v[i - 1].key - minKey]] = v[i - 1];
59      }
60      return result;
61  }
62
63  int main() {
64      std::ios::sync_with_stdio(false);
65      std::cin.tie(nullptr);
66      TVector<TSortType> inputVector;
67      TSortType temp;
68      int maxKey = -1;
```

```
69      int minKey = 1000000;
70      while(std::cin >> temp) {
71          inputVector.PushBack(temp);
72          if (temp.key > maxKey) {
73              maxKey = temp.key;
74          }
75          if (temp.key < minKey) {
76              minKey = temp.key;
77          }
78      }
79      inputVector.ShrinkToFit();
80      TVector<TSortType> result = CountingSort(inputVector, minKey, maxKey);
81      for (TSortType i : result) {
82          std::cout << i << "\n";
83      }
84      return 0;
85  }
```

3 Консоль

```
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  da_01  da_01.cbp  empty  empty_test
generate  Makefile  sort  test  test_01  test_02  test_03
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ cmake --build . --target
da_01
[100%] Built target da_01
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ cat empty_test
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ./da_01 <empty_test
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ cat test_01
1 4
5 8
2 11
8 74
4 12
5 2
5 1
1 5
10 6

ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ./da_01 <test_01
000001 4
000001 5
000002 11
000004 12
000005 8
000005 2
000005 1
000008 74
000010 6
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ cat test_02
000000 31241243123
999999 45345412314
992329 9173241519274
452312 243252341
553322 71212358762984
342348 127389123
121231 5237412
523343 213
234252 24534
```

```
345345 342738
251932 3402730472
241939 43058209
532423 24876384723
123412 7543792
999999 10182310
999999 151094823
999999 25103750929
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ./da_01 <test_02
000000 31241243123
121231 5237412
123412 7543792
234252 24534
241939 43058209
251932 3402730472
342348 127389123
345345 342738
452312 243252341
523343 213
532423 24876384723
553322 71212358762984
992329 9173241519274
999999 45345412314
999999 10182310
999999 151094823
999999 25103750929
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ cat test_03
000007 3484614493785666169
532887 4828641007683365971
679449 13055523280712283859
053473 12318502244045278956
383502 1925314227934318260
589109 9647485219232062217
416093 18033171869389351789
262512 3394578529375205246
632781 18405411276370922156
247094 16839657125260286356
631777 12758205263944437784
237827 6268503287710306687
486626 4171976010561805431
516408 9161769635579038968
```

```
266204 4370791132210130574
500819 5889500272216580892
529866 13562876108260841219
761685 17652691491696419685
015871 17839141181180571478
736390 18444243949387268945
233247 6230458118142611781
591246 11701249246842320528
287277 708908267150101213
802586 2464714855437598181
955575 16392624604709550746
624990 7907703968673471409
091010 4625519226348829401
026882 18160345884143606829
180936 8702208438469711566
150368 15614314300537846604
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ./da_01 <test_03
000007 3484614493785666169
015871 17839141181180571478
026882 18160345884143606829
053473 12318502244045278956
091010 4625519226348829401
150368 15614314300537846604
180936 8702208438469711566
233247 6230458118142611781
237827 6268503287710306687
247094 16839657125260286356
262512 3394578529375205246
266204 4370791132210130574
287277 708908267150101213
383502 1925314227934318260
416093 18033171869389351789
486626 4171976010561805431
500819 5889500272216580892
516408 9161769635579038968
529866 13562876108260841219
532887 4828641007683365971
589109 9647485219232062217
591246 11701249246842320528
624990 7907703968673471409
631777 12758205263944437784
```

632781 18405411276370922156
679449 13055523280712283859
736390 18444243949387268945
761685 17652691491696419685
802586 2464714855437598181
955575 16392624604709550746

4 Тест производительности

Существует также алгоритм сортировки подсчётом, работающий за квадратичное время. В нем не используется массив, содержащий информацию о элементах, меньших или равных данному. В данном алгоритме используется лишь один вспомогательный массив - результирующий.

Псевдокод алгоритма[2]

```
SquareCountingSort
    for i = 0 to n - 1
        c = 0;
        for j = 0 to i - 1
            if A[j] <= A[i]
                c = c + 1;
        for j = i + 1 to n - 1
            if A[j] < A[i]
                c = c + 1;
        B[c] = A[i];
```

Напишем простую программу generate для генерации тестов(принимает количество тестовых строк, минимальный и максимальный ключи, которые гарантированно будут в тесте). Также добавим в программу код, выводящий в $std :: cerr$ время выполнения в миллисекундах(используется библиотека std::chrono). Сравним данным способом время, за которое выполняют свою задачу $std :: sort$, сортировка подсчётом и квадратичная сортировка подсчётом.

```
#include <ctime>
#include <random>
#include <limits>
#include <cstdlib>
#include <iostream>
#include <iomanip>
using namespace std;

default_random_engine rng;

uint64_t my_random(uint64_t max) {
    uniform_int_distribution<unsigned long long> dist_ab(0, max);
    return dist_ab(rng);
}

int main(int argc, char** argv) {
    if (argc < 4) {
        return 1;
    }
    long long count = stoll(argv[1]);
```

```
21      long long min_key = stoll(argv[2]);
22      long long max_key = stoll(argv[3]);
23      cout << setw(6) << setfill('0') << min_key << my_random(numeric_limits<uint64_t>::
            max()) << "\n";
24      cout << setw(6) << setfill('0') << max_key << "\t" << my_random(numeric_limits<
            uint64_t>::max()) << "\n";
25      for (int i = 0; i < count; ++i) {
26          cout << setw(6) << setfill('0') << my_random(999999 - min_key - max_key) +
                min_key << "\t" <<  my_random(numeric_limits<uint64_t>::max()) << "\n";
27      }
28  }
```

1 Протокол тестирования производительности

```
ilya@ilya-lenovo:~/test_dir$ ./generate 1000 >test_1000
ilya@ilya-lenovo:~/test_dir$ ./generate 10000 >test_10000
ilya@ilya-lenovo:~/test_dir$ ./generate 100000 >test_100000
ilya@ilya-lenovo:~/test_dir$ ./generate 1000000 >test_1000000

ilya@ilya-lenovo:~/test_dir$ ./std_sort <test_1000 >result_1000
Std_sort_time: 1ms
ilya@ilya-lenovo:~/test_dir$ ./std_sort <test_10000 >result_10000
Std_sort_time: 5ms
ilya@ilya-lenovo:~/test_dir$ ./std_sort <test_100000 >result_100000
Std_sort_time: 59ms

ilya@ilya-lenovo:~/test_dir$ ./square_sort <test_1000 >result_1000
Square_counting_sort_time: 22ms
ilya@ilya-lenovo:~/test_dir$ ./square_sort <test_10000 >result_10000
Square_counting_sort_time: 1424ms
ilya@ilya-lenovo:~/test_dir$ ./square_sort <test_100000 >result_100000
Square_counting_sort_time: 125425ms

ilya@ilya-lenovo:~/test_dir$ ./da_01 <test_1000 >result_1000
Counting_sort_time: 21ms
ilya@ilya-lenovo:~/test_dir$ ./da_01 <test_10000 >result_10000
Counting_sort_time: 28ms
ilya@ilya-lenovo:~/test_dir$ ./da_01 <test_100000 >result_100000
Counting_sort_time: 49ms

ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ./da_01 <test_1000000
```

```
>result
Counting_sort_time: 366ms
ilya@ilya-lenovo:~/CLionProjects/da_01/cmake-build-debug$ ./std_sort <test_1000000
>result
Std_sort_time: 653ms
```

Как видно, время работы сортировки за квадратичное время очень быстро растёт с увеличением количества тестовых данных, в то время как сортировка подсчётом за линейное время работает крайне быстро. Как видно из последнего теста, время сортировки подсчётом для больших наборов тестовых данных становится значительно быстрее стандартной сортировки(ожидаемо, $std :: sort$ имеет оценку $O(nlogn)$.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ» я разобрался в тонкостях работы алгоритма сортировки подсчётом за линейное время. Также я приобрёл немного опыта в создании шаблонов классов и поиска утечек памяти в своей программе. Благодаря прочитанной литературе я узнал о наименьшей возможной асимптотической оценке снизу количества операций для алгоритмов сортировки, использующих сравнение элементов.

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: 22.10.2019).