МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ»
ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент гр. 9303 _____ Эйсвальд М.И.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить алгоритмы сортировки, научиться оценивать эффективность различных сортировок на разных наборах входных данных, реализовать один из алгоритмов сортировки.

Задание.

Вариант **21**: Соломонова сортировка.

Основные теоретические положения.

Алгоритм сортировки — алгоритм для упорядочивания элементов коллекции. Очевидно, что такие алгоритмы имеют смысл только для упорядоченных коллекций.

Соломонова сортировка — алгоритм сортировки, основанный на принципе приблизительного распределения и последующей вставки элементов в коллекцию. Является сортировкой распределением, а также сортировкой без сравнений.

Описание алгоритма

Принцип соломоновой сортировки состоит в следующем: для каждого элемента коллекции определяется его примерный индекс в упорядоченном массиве, после чего уже упорядочиваются элементы, «конкурирующие» между собой за один индекс (возможно, тем же способом). Сам алгоритм несложен: вычисляется особая величина Δ, равная отношению размаха последовательности к количеству элементов данной последовательности. Потом предполагаемый индекс j каждого элемента A_i (пусть массив индексируется с единицы) вычисляется как

$$j = \frac{A_i - A_{min}}{\Delta} + 1, \tag{1}$$

где A_{min} — наименьший элемент массива, а результат вычислений приводится к целому.

Тогда j можно интерпретировать как количество шагов длины Δ, которое надо отступить от минимального элемента, чтобы примерно достичь A_i.

Отсюда легко видеть, что при равноотстоящих элементах последовательности: $A_{min}, A_{min} + \Delta, A_{min} + 2\Delta, \ldots$ индексы j будут равны 1, 2, 3 и т. д., а в случае, когда один элемент коллекции много больше или меньше остальных, многие элементы получат один индекс. Эти случаи являются соответственно лучшим и худшим для соломоновой сортировки.

После назначения индексов элементам упорядочиваются элементы, претендующие на один индекс. Автор сортировки предлагает при 5 и более претендентах на одно место вызывать алгоритм рекурсивно для нового массива, при меньшем количестве элементов с одним индексом — сравнивать элементы непосредственно другими способами [1].

К достоинствам данной сортировки можно отнести скорость — $O(n)$ в лучшем случае, когда нет необходимости сортировать числа с совпадающими индексами. Если считать, что нам требуется сортировать относительно «равномерную» последовательность, большинство случаев будут близки к лучшему. В общем случае скорость алгоритма составит порядка $O(n \times k)$, где k — количество кандидатов на одно место. Отсюда, соответственно, в худшем случае — все элементы, кроме одного, вновь и вновь получают один индекс — сложность сортировки равна $O(n^2)$. Скорость работы в худшем случае уже можно назвать недостатком алгоритма: существуют алгоритмы сортировки, гарантирующие скорость $O(n \log n)$ даже на «плохих» данных. Тем не менее, как справедливо отмечено в комментариях к [1], если откуда-то заранее известно, что входная последовательность очень неравномерна, можно вычисления (1) производить не с самими числами последовательности, а с их логарифмами (правда, это имеет не очень широкое приенение).

Очевидно, алгоритм требует много дополнительной памяти: претендентов на разные индексы надо где-то хранить. Константным количеством дополнительной памяти никак не обойтись: сортировка требует $O(n \times k)$.

Стоит отметить не имеющее воздействия на эффективность, но всё же не очень приятное обстоятельство. Применим формулу (1) для наибольшего элемента: $j = N\Delta/\Delta + 1 = N + 1$. По нашему предположению, массив нумеруется с единицы, так что индекса $N+1$ быть не может. Для максимального элемента индекс, возможно, придётся уменьшить на 1. Из-за неточности представления вещественных чисел в памяти, возможно и такое, что отношение размаха выборки к Δ окажется меньше N. Кстати, подобное справедливо

для всех $A_i : A_{max} - A_{min} = m(A_i - A_{min})$ и N кратно m ($m \in \mathbb{Z}$). Однако влияния на эффективность алгоритма это не оказывает.

Выполнение работы.

Шаблонная функция `solomonSort()` принимает на вход сортируемый массив и его длину. В целом функция реализует описанный выше алгоритм, но с некоторыми изменениями.

Сначала ищутся максимум и минимум. Зная их, можно посчитать величину Δ. Если $\Delta = 0$, то можно сразу сделать вывод, что все элементы массива равны.

Примерные индексы элементов вычисляются по формуле (1), но без сдвига на 1, поскольку массивы в C++ принято индексировать с нуля.

В предварительно созданный массив списков той же длины, что и полученный функцией буфер, записываются обработанные элементы: элемент с индексом j пишется в j-й список. Списки постоянно упорядочены: меньшие элементы записываются перед большими. Выбор односвязных списков из стандартной библиотеки обусловлен следующими причинами:

- Ручное управление динамическими массивами для претендентов на одно место неудобно: надо хранить длину массива отдельно, предусмотреть расширение массивов (или сразу выделить память «по максимуму», что приведёт к затратности $O(n^2)$ по памяти);
- Односвязные списки из стандартной библиотеки разрабатывались не только удобными, но и эффективными [2];
- Сортировка образующихся групп тем же методом — не единственный вариант.

Таким образом, собственно соломонова сортировка применяется к массиву лишь однажды, а полученные подклассы сортируются простыми вставками в список.

После описанных операций остаётся лишь пройтись по каждому списку массива по порядку, заменив элементы исходного массива на вновь считанные.

Полученный алгоритм в общем случае требует $O(n \times k)$ памяти и $O(n \times k)$ времени: вычисление индексов для n элементов, итерирование по списку длины порядка k для каждого из них и вставка за константное время (помимо поиска максимума и минимума за $O(n)$ и копирования значений в исходный массив).

Тестирование.

Результаты тестирования программы представлены в таблицах ниже. Для удобства восприятия некоторые строки вывода, не относящиеся к сути, были удалены.

Вывод.

В ходе выполнения лабораторной работы была изучена соломонова сортировка, её преимущества и недостатки. Результатом работы стала программа, использующая соломонову сортировку для упорядочивания массива целых чисел.

Таблица 1 – Тестирование на корректных входных данных

Входные данные	Вывод
-6 8 4 0 2 -3	Min: -6 Max: 8 delta = 2.33333 For -6 estimated position is (-6- min)/delta = 0 of 5 For 8 estimated position is (8-min)/delta = 5 of 5 For 4 estimated position is (4-min)/delta = 4 of 5 For 0 estimated position is (0-min)/delta = 2 of 5 For 2 estimated position is (2-min)/delta = 3 of 5 For -3 estimated position is (-3- min)/delta = 1 of 5 Finished sorting... -6 -3 0 2 4 8
23 5 1 8 17 11	Min: 1 Max: 23 delta = 3.66667 For 23 estimated position is (23- min)/delta = 5 of 5 For 5 estimated position is (5-min)/delta = 1 of 5 For 1 estimated position is (1-min)/delta = 0 of 5 For 8 estimated position is (8-min)/delta = 1 of 5 For 17 estimated position is (17- min)/delta = 4 of 5 For 11 estimated position is (11- min)/delta = 2 of 5 Finished sorting... 1 5 8 11 17 23
2 100 1	Min: 1 Max: 100 delta = 33 For 2 estimated position is (2-min)/delta = 0 of 2 For 100 estimated position is (100- min)/delta = 2 of 2 For 1 estimated position is (1-min)/delta = 0 of 2 Finished sorting... 1 2 100

Таблица 2 – Тестирование на последовательности равных элементов

Входные данные	Вывод
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Min: 2 Max: 2 delta = 0 All numbers are equal!

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: sort.h

```
#ifndef SORT_H
#define SORT_H
```

```
#include <cassert>

#include <fstream>
#include <iostream>
#include <forward_list>
#include <utility>

#define llu long long unsigned

#define EMPTY_ARRAY 1
#define ALL_EQUAL 2

template <typename T>
int solomonSort(T* array, llu length, std::ofstream& out){
if(!length) return EMPTY_ARRAY;
T min = array[0];
T max = array[0];
for(llu i = 0; i < length; ++i){
if(array[i] < min) min = array[i];
if(array[i] > max) max = array[i];
}
std::cout << ”Min: ” << min << ”\t”;
out << ”Min: ” << min << ”\t”;
std::cout << ”Max: ” << max << ”\n”;
out << ”Max: ” << max << ”\n”;

double delta = (max - min)/(double)length;
std::cout << ”delta = ” << delta << ”\n”;
out << ”delta = ” << delta << ”\n”;

if(delta < 0.00001) return ALL_EQUAL;

std::forward_list<T>* cand = new std::forward_list<T>[length];

llu index;
for(llu i = 0; i < length; ++i){
index = (array[i] - min)/delta;
if(index == length) --index;
std::cout<<”For ”<<array[i]<<” estimated position is (”
<<array[i]<<”-min)/delta = ”<<index<<
” of ”<<length-1<<”\n”;
out<<”For ”<<array[i]<<” estimated position is (”
                        <<array[i]<<”-min)/delta = ”<<index<<
                        ” of ”<<length-1<<”\n”;

//if(cand[index].empty()) cand[index].push_front(array[i]);
for(auto iter = cand[index].before_begin(); iter != cand[index].end(); ++iter){
auto tmp = iter;
++tmp;
if(tmp == cand[index].end() || *(tmp) >= array[i]){
cand[index].insert_after(iter, array[i]);
break;
}
}
}

llu i = 0;
for(llu index = 0; index < length; ++index){
for(auto iter = cand[index].begin(); iter != cand[index].end(); ++iter){
```

```
array[i++] = *iter;
}
}

delete[] cand;
return 0;
}

#endif
```

Название файла: main.cpp

```
#include <cstdlib>
#include <cstring>
#include <cstdio>
#include <cctype>

#include <iostream>
#include <fstream>
#include <string>

#include ”sort.h”

#define OUTFILE ”output.txt”
#define NUMBER_CHARS ”+-”

int main(){
int* buffer = nullptr;
int index;
const char* source;
int elem;
int sort_result;

std::ofstream out(OUTFILE);
if(!out.is_open()){
std::cout << ”Failed to create output file. Exiting...\n”;
return 0;
}
std::string str;
std::cout << ”Specify input file name: ”;
getline(std::cin, str);
std::ifstream infile(str.c_str());
if(!infile.is_open()){
std::cout << ”Failed to open file. Expecting input from console...\n”;
std::cout << ”Enter EOF (i.e. Ctrl+D) to finish.\n”;
}
do{
index = 0;
getline( (infile.is_open() ? infile : std::cin) , str);
if( (infile.is_open() && infile.eof()) || (std::cin.eof()) ) break;
std::cout << ”Read line: \”” << str << ”\”\n”;
out << ”Read line: \”” << str << ”\”\n”;
buffer = (int*)realloc(buffer , sizeof(int)*(str.length()/2 + 1) );
//source = (char*)realloc(reserved, sizeof(char)*(str.length()+1) );
//strcpy(source , str.c_str());
//reserved = source;
//puts(source);
source = str.c_str();
while(source[0] && !isdigit(source[0]) && !strchr(NUMBER_CHARS,source[0])) ++source;
int result;
```

```
do{
result = sscanf(source , "%d" , &elem);
if(result == 1){
   buffer[index++] = elem;
   while((source[0])&&(isdigit(source[0]) || strchr(NUMBER_CHARS,source[0])))
++source;
   while(source[0] && !isdigit(source[0]) && !strchr(NUMBER_CHARS,source[0]))
++source;
}
}while(result == 1);
std::cout << "Extracted numbers: ";
out << "Extracted numbers: ";
for(int i = 0; i < index; ++i) std::cout << buffer[i] << " ";
for(int i = 0; i < index; ++i) out << buffer[i] << " ";
std::cout << "\n";
out << "\n";
sort_result = solomonSort(buffer , index , out);
switch(sort_result){
case EMPTY_ARRAY:
out << "Error: array is empty!\n";
std::cout << "Error: array is empty!\n";
break;
case ALL_EQUAL:
out << "All numbers are equal!\n";
std::cout << "All numbers are equal!\n";
break;
default:
out << "Finished sorting...\n";
std::cout << "Finished sorting...\n";
for(int i = 0; i < index; ++i) std::cout << buffer[i] << " ";
for(int i = 0; i < index; ++i) out << buffer[i] << " ";
std::cout << "\n";
out << "\n";
}
}while((infile.is_open() && !infile.eof()) || (!(infile.is_open()) && !std::cin.eof()) )

free(buffer);
return 0;
}
```

СПИСОК ЛИТЕРАТУРЫ

[1] Хабр: сообщество для IT-специалистов: соломонова сортировка https://habr.com/ru/post/208088/ (дата обращения: 26.11.2020).

[2] C++ reference: Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples: http://www.cplusplus.com/reference/forward_list/forward_list/ (дата обращения: 26.11.2020).

[3] Algolab: сортировка царя Соломона http://algolab.valemak.com/solomon (дата обращения: 26.11.2020).