Федеральное государственное бюджетное образовательное учреждение высшего образования «Национальный исследовательский университет «МЭИ»

Институт информационных и вычислительных технологий

Кафедра Управления и интеллектуальных технологий

**Отчёт по лабораторной работе № 3**

# По курсу «Разработка ПО систем управления»

# «Декомпозиция программы»

# Выполнил студент группы А-01-19

# Нецветай Д.О.

# Проверили

# Мохов А. С

# Козлюк Д. А

Москва 2020

**Задание.**

# 1) Написать программу для построения гистограммы массива чисел как изображения в формате SVG

2) Доработать программу в соответствии с вариантом.

#### Вариант 12

#### Добавьте на ось подписей границы столбцов по аналогии с заданием этого варианта в лабораторной работе 1. Добавьте на ось подписей границы столбцов. Например, если в первый столбец отнесены элементы от наименьшего до 1,23, во второй — от 1,23 до 2,34 и т. д., желаемый результат:

8|\*\*\*\*\*\*\*\*

1.23

11|\*\*\*\*\*\*\*\*\*\*\*

2.34

6|\*\*\*\*\*\*

**Пояснение к программе.**

Границы столбцов определяются следующим образом: нижняя граница первого столбца равняется минимальному элементу массиву чисел, а верхняя граница верхнего столбца равна нижней границе + шаг, следовательно, нижней границей следующего столбца будет верхняя предыдущего, а верхней границей будет нижняя + шаг. Вывод осуществляется через функцию svg\_text(), в условии задания не требуется выводить верхнюю границу последнего столбца, поэтому ставится следующее условие: border(граница) < max(максимальный элемент).

Также в программу внесено масштабирование при условии, что размер максимального столбца превышает 75 единиц.

**Текст программы.**

**1)main.cpp**

#include <iostream>

#include <vector>

#include "histogram.h"

#include "svg.h"

using namespace std;

vector<double> input\_numbers(size\_t count)

{

vector<double> result(count);

for (size\_t i = 0; i < count; i++)

{

cin >> result[i];

}

return result;

}

vector <size\_t> make\_histogram(const vector<double>& numbers,size\_t bin\_count)

{

double min;

double max;

find\_minmax(numbers,min,max);

vector<size\_t> bins(bin\_count,0);

for (double number : numbers)

{

size\_t bin;

bin = (number - min) / (max - min) \* bin\_count;

if (bin == bin\_count)

{

bin--;

}

bins[bin]++;

}

return bins;

}

int main()

{

size\_t number\_count;

cerr << "Enter number count: ";

cin >> number\_count;

cerr << "Enter numbers: ";

const auto numbers = input\_numbers(number\_count);

size\_t bin\_count;

cerr << "Enter column count: ";

cin >> bin\_count;

double min, max;

find\_minmax(numbers, min, max);

const auto bins = make\_histogram(numbers, bin\_count);

double wag=((max-min)/bin\_count);

show\_histogram\_svg(bins,wag,min,max);

return 0;

}

**2)histogram.cpp**

#include "histogram.h"

#include <vector>

void find\_minmax(const vector<double>& numbers, double& min, double& max)

{

if(numbers.size()==0)

{

return;

}

else

{

min = numbers[0];

max = numbers[0];

for (double number : numbers)

{

if (number < min)

{

min = number;

}

if (number > max)

{

max = number;

}

}

}

}

**3)histogram.h**

#ifndef HISTOGRAM\_H\_INCLUDED

#define HISTOGRAM\_H\_INCLUDED

#include <vector>

using namespace std;

void find\_minmax(const vector<double>& numbers, double& min, double& max);

#endif // HISTOGRAM\_H\_INCLUDED

**4)svg.h**

#ifndef SVG\_H\_INCLUDED

#define SVG\_H\_INCLUDED

#include <vector>

using namespace std;

void svg\_rect(double x, double y, double width, double height,string stroke,string fill );

void svg\_text(double left, double baseline, string text);

void svg\_begin(double width, double height);

void svg\_end();

void show\_histogram\_svg(const vector<size\_t>& bins,double wag,double min,double max);

#endif // SVG\_H\_INCLUDED

**5)svg.cpp**

#include <iostream>

#include <vector>

#include "svg.h"

using namespace std;

double fun\_wag(double& min,double& max,size\_t& bin\_count)

{

if(max == min || bin\_count==0)

{

return 0;

}

else

{

return (max-min)/bin\_count;

}

}

void max\_width(const vector<size\_t>& bins,double& maximum)

{

if (bins.size()==0)

{

return;

}

else

{

for (size\_t bin : bins)

{

if (bin > maximum)

{

maximum = bin;

}

}

}

}

void svg\_rect(double x, double y, double width, double height,string stroke = "black",string fill = "black")

{

cout << "<rect x='"<<x<< "' y='" <<y<<"' width='" <<width <<"' height='" <<height <<"' stroke='"<<stroke<<"' fill='"<<fill<<"'/>";

}

void

svg\_text(double left, double baseline, string text)

{

cout << "<text x='" << left << "' y='" << baseline <<"'>" <<text <<"</text>";

}

void

svg\_begin(double width, double height)

{

cout << "<?xml version='1.0' encoding='UTF-8'?>\n";

cout << "<svg ";

cout << "width='" << width << "' ";

cout << "height='" << height << "' ";

cout << "viewBox='0 0 " << width << " " << height << "' ";

cout << "xmlns='http://www.w3.org/2000/svg'>\n";

}

void

svg\_end()

{

cout << "</svg>\n";

}

void show\_histogram\_svg(const vector<size\_t>& bins,double min,double max,size\_t bin\_count)

{

const auto IMAGE\_WIDTH = 400;

const auto IMAGE\_HEIGHT = 300;

const size\_t MAX\_ASTERISK=76;

const auto TEXT\_LEFT = 20;

const auto TEXT\_BASELINE = 20;

const auto TEXT\_WIDTH = 50;

const auto BIN\_HEIGHT = 30;

const auto BLOCK\_WIDTH = 10;

const auto GRAPH\_WIDTH = IMAGE\_WIDTH-TEXT\_WIDTH;

double top = 0;

double maximum=bins[0];

max\_width(bins,maximum);

svg\_begin(IMAGE\_WIDTH, IMAGE\_HEIGHT);

double wag= fun\_wag(min,max,bin\_count);

double border=min+wag;

size\_t max\_count=0;

for(size\_t b:bins)

{

if(b>max\_count)

{

max\_count=b;

}

}

if (max\_count > MAX\_ASTERISK)

{

const double scaling\_multiplier = (double)MAX\_ASTERISK / max\_count;

if(max\_count\*BLOCK\_WIDTH/GRAPH\_WIDTH>1)

{

for (size\_t bin : bins)

{

const double new\_scaling\_multiplier=(double)GRAPH\_WIDTH/(max\_count\*BLOCK\_WIDTH\*scaling\_multiplier);

const size\_t new\_bin\_width = (size\_t)BLOCK\_WIDTH\*bin\*scaling\_multiplier\*new\_scaling\_multiplier;

svg\_text(TEXT\_LEFT+25, top + TEXT\_BASELINE, to\_string(bin));

svg\_rect(TEXT\_WIDTH+25, top, new\_bin\_width, BIN\_HEIGHT,"lawngreen","#7CFC00");

top += BIN\_HEIGHT;

if(border<max)

{

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(border));

top += BIN\_HEIGHT;

border=border+wag;

}

}

}

else

{

for (size\_t bin : bins)

{

const double bin\_width = BLOCK\_WIDTH\*bin\*scaling\_multiplier;

svg\_text(TEXT\_LEFT+25, top + TEXT\_BASELINE, to\_string(bin));

svg\_rect(TEXT\_WIDTH+25, top,bin\_width, BIN\_HEIGHT,"lawngreen","#7CFC00");

top += BIN\_HEIGHT;

if(border<max)

{

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(border));

top += BIN\_HEIGHT;

border=border+wag;

}

}

}

}

else

{

if(max\_count\*BLOCK\_WIDTH/GRAPH\_WIDTH>1)

{

for (size\_t bin : bins)

{

const double new\_scaling\_multiplier=(double)GRAPH\_WIDTH/(max\_count\*BLOCK\_WIDTH);

const size\_t new\_bin\_width = (size\_t)BLOCK\_WIDTH\*bin\*new\_scaling\_multiplier;

svg\_text(TEXT\_LEFT+25, top + TEXT\_BASELINE, to\_string(bin));

svg\_rect(TEXT\_WIDTH+25, top,new\_bin\_width, BIN\_HEIGHT,"lawngreen","#7CFC00");

top += BIN\_HEIGHT;

if(border<max)

{

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(border));

top += BIN\_HEIGHT;

border=border+wag;

}

}

}

else

{

for (size\_t bin : bins)

{

const double bin\_width = BLOCK\_WIDTH \* bin;

svg\_text(TEXT\_LEFT+25, top + TEXT\_BASELINE, to\_string(bin));

svg\_rect(TEXT\_WIDTH+25, top,bin\_width, BIN\_HEIGHT,"lawngreen","#7CFC00");

top += BIN\_HEIGHT;

if(border<max)

{

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(border));

top += BIN\_HEIGHT;

border=border+wag;

}

}

}

}

svg\_end();

}

**6)test-dz**

#include "C:\Users\nedeo\Desktop\прога\lab03\12\histogram.h"

#include "C:\Users\nedeo\Desktop\прога\lab03\12\svg.h"

#include <cassert>

void test\_positive()

{

double min = 0;

double max = 0;

find\_minmax({1, 2, 3}, min, max);

assert(min == 1);

assert(max == 3);

}

void test\_otricala()

{

double min = 0;

double max = 0;

find\_minmax({-1, -2, -3}, min, max);

assert(min == -3);

assert(max == -1);

}

void test\_same()

{

double min = 0;

double max = 0;

find\_minmax({3, 3, 3}, min, max);

assert(min == 3);

assert(max == 3);

}

void test\_ones()

{

double min = 0;

double max = 0;

find\_minmax({2}, min, max);

assert(min == 2);

assert(max == 2);

}

void test\_emptyarr()

{

double min = 0;

double max = 0;

find\_minmax({}, min, max);

}

void test\_1()

{

double min=2;

double max=4;

size\_t bin\_count=2;

assert( fun\_wag(min,max,bin\_count) == 1.0);

}

void test\_2()

{

double min=2;

double max=10;

size\_t bin\_count=4;

assert( fun\_wag(min,max,bin\_count) == 2.0);

}

void test\_3()

{

double min=2;

double max=2;

size\_t bin\_count=2;

assert(fun\_wag(min,max,bin\_count) == 0);

}

void test\_4()

{

double min=2;

double max=2;

size\_t bin\_count=0;

assert( fun\_wag(min,max,bin\_count) == 0);

}

int main()

{

test\_positive();

test\_otricala();

test\_same();

test\_ones();

test\_emptyarr();

test\_1();

test\_2();

test\_3();

test\_4();

}

**Ссылка на репозитарий.**

https://github.com/NetsvetayDO/lab03