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 студентка группы А-01-20

Яковлева А. Л.

Проверили

Мохов А. С.

Москва 2021

**Вариант 7**

Вычислите среднюю высоту столбца. Если столбец выше, цвет столбца должен быть красным, если ниже или равен средней высоте, цвет зеленый.

**Логика решения**

Мною были написаны 3 функции:

* bin\_height

В ней создаётся вектор высот столбцов с проверкой условия масштабирования.

* avg

Используется для расчета среднего значения вектора.

* color

Выполняет основную часть задачи – выдаёт цвет столбца гистограммы в зависимости от величины столбца и среднего значения. (Если столбец выше среднего значения, то он становится красным, если ниже или равен – зеленым)

Ссылка на репозитарий:

<https://github.com/YakovlevaAL/cs-lab03>

**Код**

**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, const size\_t count)

{

vector<size\_t> result(count);

double min;

double max;

find\_minmax(numbers, min, max);

for (double number : numbers)

{

size\_t bin = (size\_t)((number - min) / (max - min) \* count);

if (bin == count)

{

bin--;

}

result[bin]++;

}

return result;

}

void

show\_histogram\_text(vector<size\_t> bins)

{

const size\_t SCREEN\_WIDTH = 80;

const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 4 - 1;

size\_t max\_count = 0;

for (size\_t count : bins)

{

if (count > max\_count)

{

max\_count = count;

}

}

const bool scaling\_needed = max\_count > MAX\_ASTERISK;

for (size\_t bin : bins)

{

if (bin < 100)

{

cout << ' ';

}

if (bin < 10)

{

cout << ' ';

}

cout << bin << "|";

size\_t height = bin;

if (scaling\_needed)

{

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

height = (size\_t)(bin \* scaling\_factor);

}

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

{

cout << '\*';

}

cout << '\n';

}

}

int

main()

{

// Ввод данных

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;

//Обработка данных

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

// Вывод данных

show\_histogram\_svg(bins);

return 0;

}

**histogram.h**

#pragma once

#include <vector>

using namespace std;

void

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

**histogram.cpp**

#include "histogram.h"

void

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

{

if (numbers.size() != 0)

{

min = numbers[0];

max = numbers[0];

for (double number : numbers)

{

if (number < min)

{

min = number;

}

if (number > max)

{

max = number;

}

}

}

}

**svg.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include "avg\_colour.h"

using namespace std;

void

svg\_begin(double width, double height);

void

svg\_end();

void

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

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

void

show\_histogram\_svg(const vector<size\_t>& bins);

**svg.cpp**

#include "svg.h"

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

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

{

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

}

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

{

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

}

void

show\_histogram\_svg(const vector<size\_t>& bins)

{

const auto IMAGE\_WIDTH = 400;

const auto IMAGE\_HEIGHT = 300;

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 size\_t MAX\_ASTERISK = IMAGE\_WIDTH - TEXT\_LEFT - TEXT\_WIDTH;

size\_t max\_count = 0;

for (size\_t count : bins) {

if (count > max\_count) {

max\_count = count;

}

}

const bool scaling\_needed = max\_count \* BLOCK\_WIDTH > MAX\_ASTERISK;

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

double top = 0;

const auto heights = bin\_height(bins, scaling\_needed, max\_count, MAX\_ASTERISK, BLOCK\_WIDTH);

double heights\_average = avg(heights);

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

{

string colour = color(heights[i], heights\_average);

const double bin\_width = BLOCK\_WIDTH \* heights[i];

svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bins[i]));

svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "black", colour);

top += BIN\_HEIGHT;

}

svg\_end();

}

**avg\_colour.h**

#pragma once

#include <vector>

#include <string>

using namespace std;

vector<size\_t>

bin\_height(const vector<size\_t>& bins, bool scaling\_needed, size\_t max\_count, size\_t MAX\_ASTERISK, int BLOCK\_WIDTH);

double

avg(const vector<size\_t>& heights);

string

color(size\_t height, double heights\_average);

**avg\_colour.cpp**

#include "avg\_colour.h"

vector<size\_t>

bin\_height(const vector<size\_t>& bins, bool scaling\_needed, size\_t max\_count, size\_t MAX\_ASTERISK, int BLOCK\_WIDTH)

{

vector<size\_t> result(bins.size());

size\_t i = 0;

for (size\_t bin : bins)

{

result[i] = bin;

if (scaling\_needed)

{

const double scaling\_factor = (double)MAX\_ASTERISK / (max\_count \* BLOCK\_WIDTH);

result[i] = (size\_t)(bin \* scaling\_factor);

}

i++;

}

return result;

}

double

avg(const vector<size\_t>& heights)

{

double avg = 0;

for (size\_t height : heights)

{

avg += height;

}

avg /= heights.size();

return avg;

}

string

color(size\_t height, double heights\_average)

{

if (height > heights\_average)

return "red";

else

return "green";

}

**test.cpp**

#include "histogram.h"

#include "avg\_colour.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\_negative() {

double min = 0;

double max = 0;

find\_minmax({-4, -5, -2}, min, max);

assert(min == -5);

assert(max == -2);

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

assert(min == 2);

assert(max == 2);

}

void

test\_equal() {

double min = 0;

double max = 0;

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

assert(min == 2);

assert(max == 2);

}

void

test\_one() {

double min = 0;

double max = 0;

find\_minmax({5}, min, max);

assert(min == 5);

assert(max == 5);

}

void test\_empty() {

double min = 0;

double max = 0;

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

assert(min == 0);

assert(max == 0);

}

void

test\_colour\_first() {

vector<size\_t> bins = {5, 6, 7};

bool scaling\_needed = false;

size\_t max\_count = 7;

size\_t MAX\_ASTERISK = 330;

int BLOCK\_WIDTH = 10;

auto result11 = bin\_height(bins, scaling\_needed, max\_count, MAX\_ASTERISK, BLOCK\_WIDTH);

vector<size\_t> answ = {5, 6, 7};

assert(result11 == answ);

auto result12 = avg(result11);

assert(result12 == 6);

auto result13 = color(result11[1], result12);

assert(result13 == "green");

}

void

test\_colour\_second() {

vector<size\_t> bins = {10, 12, 20};

bool scaling\_needed = true;

size\_t max\_count = 18;

size\_t MAX\_ASTERISK = 330;

int BLOCK\_WIDTH = 60;

auto result21 = bin\_height(bins, scaling\_needed, max\_count, MAX\_ASTERISK, BLOCK\_WIDTH);

vector<size\_t> answ = {3, 3, 6};

assert(result21 == answ);

auto result22 = avg(result21);

assert(result22 == 4);

auto result23 = color(result21[1], result22);

assert(result23 == "green");

}

int

main() {

test\_positive();

test\_negative();

test\_equal();

test\_one();

test\_empty();

test\_colour\_first();

test\_colour\_second();

}