Отчёт по лабораторной работе №3 по теме «Декомпозиция программ»

по предмету

Разработка Программного Обеспечения Систем Управления

Коваленко Николай, А-03-19

Реализация масштабирования для отрисовки прямоугольников сделана аналогично масштабированию звёздочек в первой лабораторной работе. Для начала находится максимальное количество чисел в корзине и происходит проверка на необходимость масштабирования. В цикле отрисовки проверяется необходимость масштабирования, если да, то bin умножается на коэффициент масштабирования, который отвечает за ширину прямоугольников

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;

//double scaling\_factor = 1;

for (size\_t bin : bins)

{

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

if (scaling\_needed)

{

bin = bin\_count(MAX\_ASTERISK, max\_count, bin);

}

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

const double RECT\_LEFT = OTSTUP - bin\_width;

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

svg\_rect(RECT\_LEFT, top, bin\_width, BIN\_HEIGHT,stroke,fill);

top += BIN\_HEIGHT;

}

svg\_end();

Для реализации создается test.cpp, куда подключается histogram.h, с помощью которого становится возможным использовать функцию find\_minmax для проверки. Также подключается cassert - это модуль для проверки ошибок, в данном примере используется функция assert, которая сравнивает ожидаемые значения с реальными и в случае, если это не так программа выдаёт ошибку. В данном случае 5 тестов для проверки функции find\_minmax, где у нас массив положительных чисел, отрицательных, массив из одинаковых чисел, из одного числа , и без чисел. И 2 теста проверяющих исправную работу функции bin\_count.

#include "histogram.h"

#include "histogram\_new.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\_2()

{

double min =0;

double max =0;

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

assert (min == -3);

assert (max == -1);

}

void test\_3()

{

double min = 0;

double max = 0;

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

assert (min == 1);

assert (max == 1);

}

void test\_4()

{

double min =0;

double max =0;

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

assert (min == 1);

assert (max == 1);

}

void test\_5()

{

double min = 0;

double max = 0;

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

assert (min == 0);

assert (max == 0);

}

void test\_bin\_1 ()

{

size\_t bin = 33;

bin\_count(75, 100, bin);

assert (bin > 0);

}

void test\_bin\_2 ()

{

size\_t bin = 9;

bin\_count(75, 100, bin);

assert (bin <= 100);

}

int main()

{

test\_positive();

test\_2();

test\_3();

test\_4();

test\_5();

test\_bin\_1();

test\_bin\_2();

В качестве индивидуального задания мне было нужно выводить гистограмму на экран зеркально оригинальному виду. Для этого добавляются новые константы, называемые OTSTUP и RECT\_LEFT, которые отвечают за расчет расстояния по оси X после которого нужно начать отрисовку блоков, чтобы создать вид зеркальной отрисовки гистограммы.

#include "histogram\_new.h"

#include <iostream>

#include <math.h>

using namespace std;

size\_t bin\_count (size\_t MAX\_A, size\_t max\_c, size\_t bin)

{

double param = 0;

double scaling\_factor =1;

scaling\_factor = (double)MAX\_A / max\_c;

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

return param;

}

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\_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 svg\_text(double left, double baseline, string text,size\_t bin)

{

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

}

void show\_histogram\_svg(const vector<size\_t> bins,size\_t number\_count)

{

const auto IMAGE\_WIDTH = 400;

const auto OTSTUP = IMAGE\_WIDTH - 40;

const auto IMAGE\_HEIGHT = 300;

const auto TEXT\_LEFT = 400 - 30;

const auto TEXT\_BASELINE = 20;

const auto BIN\_HEIGHT = 30;

const auto BLOCK\_WIDTH = 5;

const size\_t SCREEN\_WIDTH = 80;

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

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

double top = 0;

string stroke="black";

string fill="red";

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)

{

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

if (scaling\_needed)

{

bin = bin\_count(MAX\_ASTERISK, max\_count, bin);

}

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

const double RECT\_LEFT = OTSTUP - bin\_width;

svg\_rect(RECT\_LEFT, top, bin\_width, BIN\_HEIGHT,stroke,fill);

top += BIN\_HEIGHT;

}

svg\_end();

}

Код всех модулей

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

Main.cpp

#include <iostream>

#include <vector>

#include "histogram.h"

#include "histogram\_new.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(vector <double> numbers,size\_t bin\_count,double min,double max)

{

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

for (double number : numbers)

{

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

if (bin == bin\_count)

{

bin--;

}

bins[bin]++;

}

return(bins);

}

void show\_histogram\_text(vector<size\_t>bins,size\_t number\_count)

{

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<<' ';

cout << '\n';

}

}

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;

double max;

find\_minmax(numbers,min,max);

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

show\_histogram\_svg(bins,number\_count);

return 0;

}

Histogram.cpp

**#include "histogram.h"**

**#include <iostream>**

**void find\_minmax(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;**

**}**

**}**

**}**

**}**

**Histogram\_new.cpp**

#include "histogram\_new.h"

#include <iostream>

#include <math.h>

using namespace std;

size\_t bin\_count (size\_t MAX\_A, size\_t max\_c, size\_t bin)

{

double param = 0;

double scaling\_factor =1;

scaling\_factor = (double)MAX\_A / max\_c;

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

return param;

}

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";

}

//svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT,stroke,fill);

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

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 svg\_text(double left, double baseline, string text,size\_t bin)

{

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

}

void show\_histogram\_svg(const vector<size\_t> bins,size\_t number\_count)

{

const auto IMAGE\_WIDTH = 400;

const auto OTSTUP = IMAGE\_WIDTH - 40;

const auto IMAGE\_HEIGHT = 300;

const auto TEXT\_LEFT = 400 - 30;

const auto TEXT\_BASELINE = 20;

const auto BIN\_HEIGHT = 30;

const auto BLOCK\_WIDTH = 5;

const size\_t SCREEN\_WIDTH = 80;

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

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

double top = 0;

string stroke="black";

string fill="red";

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;

//double scaling\_factor = 1;

for (size\_t bin : bins)

{

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

if (scaling\_needed)

{

bin = bin\_count(MAX\_ASTERISK, max\_count, bin);

}

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

const double RECT\_LEFT = OTSTUP - bin\_width;

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

svg\_rect(RECT\_LEFT, top, bin\_width, BIN\_HEIGHT,stroke,fill);

top += BIN\_HEIGHT;

}

svg\_end();

}

**Histogram\_new.h**

**#ifndef HISTOGRAM\_NEW\_H\_INCLUDED**

**#define HISTOGRAM\_NEW\_H\_INCLUDED**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**void show\_histogram\_svg(const vector<size\_t> bins, size\_t number\_count);**

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

**void svg\_end();**

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

**void svg\_text(double left, double baseline, string text,size\_t bin);**

**size\_t bin\_count (size\_t MAX\_A, size\_t max\_c, size\_t bin);**

**#endif // HISTOGRAM\_NEW\_H\_INCLUDED**

**Test.cpp**

**#include "histogram.h"**

**#include "histogram\_new.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\_2()**

**{**

**double min =0;**

**double max =0;**

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

**assert (min == -3);**

**assert (max == -1);**

**}**

**void test\_3()**

**{**

**double min = 0;**

**double max = 0;**

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

**assert (min == 1);**

**assert (max == 1);**

**}**

**void test\_4()**

**{**

**double min =0;**

**double max =0;**

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

**assert (min == 1);**

**assert (max == 1);**

**}**

**void test\_5()**

**{**

**double min = 0;**

**double max = 0;**

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

**assert (min == 0);**

**assert (max == 0);**

**}**

**void test\_bin\_1 ()**

**{**

**size\_t bin = 33;**

**bin\_count(75, 100, bin);**

**assert (bin > 0);**

**}**

**void test\_bin\_2 ()**

**{**

**size\_t bin = 9;**

**bin\_count(75, 100, bin);**

**assert (bin <= 100);**

**}**

**int main()**

**{**

**test\_positive();**

**test\_2();**

**test\_3();**

**test\_4();**

**test\_5();**

**test\_bin\_1();**

**test\_bin\_2();**

**} assert(min == -3);**

**assert(max == -1);**

**}**