Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Поиск минимального остовного дерева в неориентированном взвешенном графе

Выполнил:

студент ИИТММ гр. 0823-1

Конаков А.В.

Проверил:

ассистент каф. МО ЭВМ, ВМК

Сиднев А.А.

Нижний Новгород

2017 г.

Содержание

[Введение 3](#_Toc482967582)

[Постановка задачи 4](#_Toc482967583)

[Руководство пользователя 5](#_Toc482967584)

[Руководство программиста 6](#_Toc482967585)

[Описание структуры программы 6](#_Toc482967586)

[Описание структур данных 7](#_Toc482967587)

[Литература 11](#_Toc482967588)

[Приложения 12](#_Toc482967589)

[main.cpp 12](#_Toc482967590)

[graph.h 18](#_Toc482967591)

[graph.cpp 19](#_Toc482967592)

[graph\_mat\_ident.h 20](#_Toc482967593)

[graph\_mat\_ident.cpp 21](#_Toc482967594)

[heap.h 22](#_Toc482967595)

[heap\_declaration.h 25](#_Toc482967596)

[jarnik.h 26](#_Toc482967597)

[jarnik.cpp 27](#_Toc482967598)

[kruskal.h 28](#_Toc482967599)

[kruskal.cpp 28](#_Toc482967600)

[unionfind.h 29](#_Toc482967601)

[unionfind.cpp 30](#_Toc482967602)

# Введение

Данная работа предназначена для получения практического опыта с использованием языка C++, а также инструментов разработки: CMake, Travis CI, Google Test, Git и GitHub. Также с учётом постоянно растущих потребностей в поиске и работе с информацией очевидной становится потребность в отработке практических навыков работы с популярными алгоритмами обработки графов, таких как например алгоритмы поиска минимальных остовных деревьев и поиска кратчайших путей, так как в наше время графы представляют одно из наиболее удобных представлений информации.

# Постановка задачи

Лабораторная работа ставит задачу в реализации алгоритмов поиска минимального остовного дерева, таких как алгоритм Крускала использующего для представления графа матрицу смежности и алгоритма Ярника получающего на вход граф в виде матрицы инцидентности. Для работы алгоритмов ставится задача реализации структур разделённого множества в виде древовидной структуры с использованием рангов и приоритетной очереди, в основании которой находится 4-куча. И последним, но не менее значимым является задача реализация графической составляющей проекта.

# Руководство пользователя

При начале работы с программой пользователь получает начальное окно, заполненное чёрным цветом. Ввод осуществляется с помощью мыши:

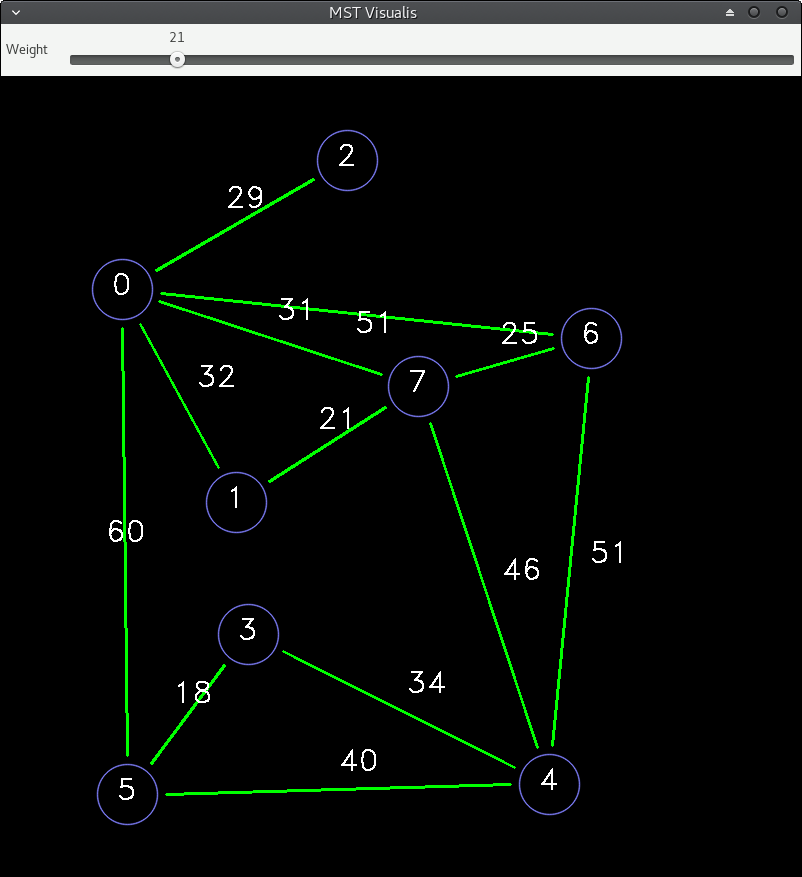
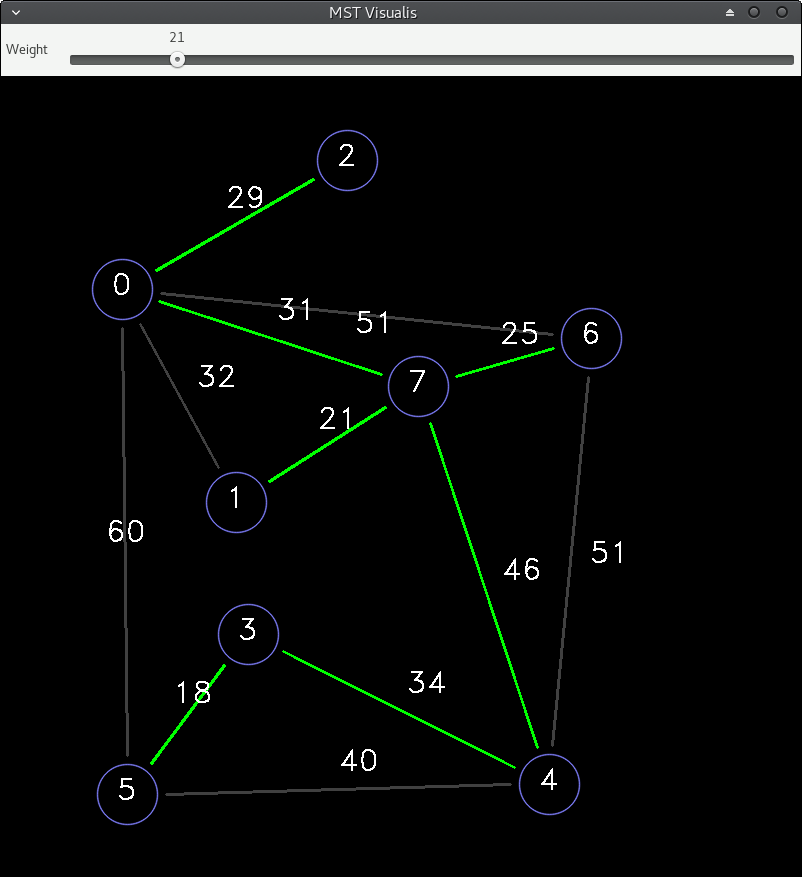
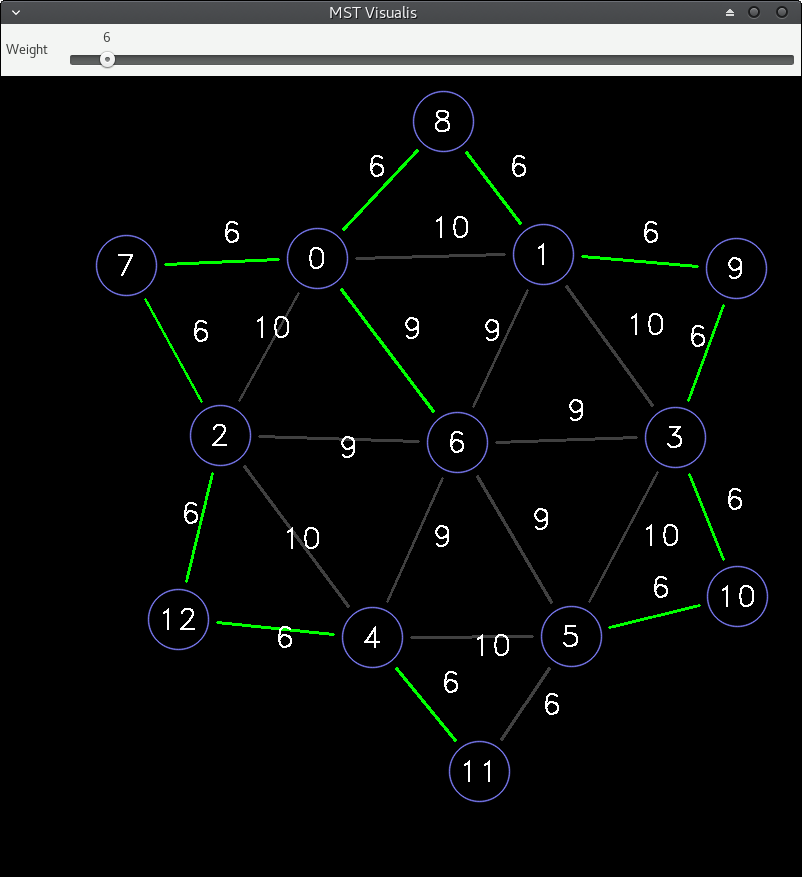
1. Одиночное нажатие ПКМ установить вершину, в случае зажатия ПКМ пользователь сможет выбрать наиболее удобное на его усмотрение положение устанавливаемой вершины.
2. Зажимая ЛКМ(начиная от положения вершины) пользователь может провести ребро между вершинами, вес вершины будет установлен в соответствии с выбранным на расположенном в верхней части окна подвижным ползунком.
3. Двойной щелчок ПКМ сбросит текущее состояние(будут стёрты все вершины и рёбра) и поместит начальную вершину в место щелчка.
4. Нажатие СКМ затемнит все рёбра не входящие в минимальное остовное дерево «собранного» пользователем графа

Пример работы:

1. Собираем граф, не забывая менять 2) По нажатию СКМ получаем минимальное

своевременно веса рёбер остовное дерево (не участвующие в МОД

рёбра сменят цвет на тёмно-серый).

Возможный результат работы для

«большого» графа.

# Руководство программиста

## Описание структуры программы

Описываемая программа app состоит из исходного файла main.cpp.

По мимо этого программа состоит из файлов:

* graph.h и vertex.h – соответственно файлы с интерфейсом и реализацией шаблона Vertex.
* graph\_mat\_ident.h и graph\_mat\_ident.cpp – соответственно файлы с интерфейсом и реализацией шаблона Graph.
* kruskal.h, kruskal.cpp и jarnik.h, jarnik.cpp – соответственно с интерфейсами и реализациями алгоритмов Крускала и Ярвика.
* heap\_declaration.h и heap.h, а также priority\_queue.h – соответственно с интерфейсом и реализацией шаблона Heap и их обёрткой PriorityQueue конкретизирующей приоритетную очередь как 4-кучу.
* unionfind.h и unionfind.cpp – соответственно c интерфейсом и реализацией разделённых множеств.

## Описание структур данных

* ***Graph*** – класс для хранения и основных операций для работы с графом хранящимся в виде матрицы смежности, имеет конструктор с параметром позволяющим задать количество вершин в графе,

Основные методы для работы с ним:

* + *setEdge(from, to, w)* – метод устанавливающий соответствующее ребро веса w, от вершины с номером from, до вершины с номером to
  + *unsetEdge(from, to)* – удаляет ребро связывающее from и to
  + *getEdgesWeight(from, to)* – метод возвращающий вес ребра между вершинами from и to или ноль если ребра нет.
  + *getSize()* – метод для получения числа вершин в графе
* ***GraphMatIdent*** – класс для хранения и основных операций для работы с графом хранящимся в виде матрицы смежности, имеет конструктор с параметром позволяющим задать количество вершин и рёбер в графе.

Основные методы для работы с ним:

* + *setEdge(from, to, n, w)* – метод устанавливающий соответствующее ребро *n* веса w, от вершины с номером from, до вершины с номером to
  + *getEdgesWeight(v, e)* – метод возвращающий вес ребра *e* к вершине *v* или ноль если ребро *e* не связано с *v*.
  + *getVerticesSize()* – метод для получения числа вершин в графе
  + *getEdgesSize()* – метод для получения числа рёбер в графе
* ***Heap<DataType, int n>*** – шаблон для хранения и основных операций для работы с n-кучами, имеет конструктор позволяющий задать размер предварительно выделяемой памяти для элементов, также в файле priority\_queue.h содержит подстановку ***PriorityQueue<DataType>*** соответствующую 4-кучам
  + *Insert(const DataType& value)* – выполняет вставку элемента со значением value в приоритетную очередь
  + *Top()* – метод позволяющий получить значение минимального элемента в очереди
  + *Resize()* – метод позволяющий вручную регулировать размер выделенного для кучи пространства памяти
  + *getSize()* и *getHeapSize()* – методы позволяющие узнать соответственно количество размещённых в куче элементов в данный момент и общее количество элементов для которых предварительно выделена память
  + *isEmpty()* – метод возвращающий истину, если очередь пуста
* ***UnionFind*** – класс для работы с разделёнными множествами
  + *find(e1, e2)* – метод позволяющий узнать находятся ли элементы e1 и e2 в одном подмножестве унивёрса.
  + *unite(e1, e2)* – метод объединяющий множества в которых находятся элементы e1 и e2.
  + *getSize()* – метод возвращает мощность унивёрса множества вершин относительно которых производятся вычисления.
* методы ***findMinSpanTreeKruskal(graph)*** и ***findMinSpanTreeJarnik(graph)*** – возвращают список пар рёбер между которыми нужно оставить рёбра чтобы получить минимальное остовное дерево.
  1. **Описание алгоритмов**

*Алгоритм Крускала.*

Будем работать с рёбрами входного графа и т.к. мы собираемся брать рёбра минимального веса, поместим рёбра в промежуточный контейнер например в приоритетную очередь, далее выполняем шаги:

1. Извлекаем из промежуточного контейнера ребро минимального веса, переходим к 2-ому шагу
2. Проверяем вершины, которые соединяют ребро, если они уже находятся в остовном дереве(или в одном и том же разделённом множестве), то переходим к следующему шагу, иначе добавляем это ребро к остовному дереву.
3. Убираем из рассмотрения ребро, взятое на первом шаге, если промежуточный контейнер пуст, то алгоритм завершается, иначе повторяем, начиная с 1-ого шага.

В результате получим список (набор) ребер, из которых состоит минимальное остовное дерево графа, исходя из алгоритма, можно также утверждать, что список упорядочен по весу содержащихся в нём рёбер.

*Алгоритм Ярника-Прима****.***

Изначально возьмём случайную вершину графа(она будет образовывать начальное остовное дерево), поместим её в промежуточный контейнер и проделаем следующие шаги:

1. Ищем минимальное по весу ребро по всем вершинам из промежуточного контейнера, которое соединяет текущее остовное дерево с вершиной, которая пока не входит в дерево, переходим к шагу 2.
2. Найденное на первом шаге ребро отправляем в остовное ребро, вершину же отправляем в промежуточный контейнер и переходим к 3-ему шагу.
3. Если промежуточный контейнер содержит все вершины графа, возвращаем результат, иначе возвращаемся к 1-ому шагу.

В результате получим список (набор) рёбер из которых состоит минимальное остовное дерево графа.

**Заключение**

Выполненная лабораторная работа позволила получить неоценимый опыт разработки и реализации алгоритмов поиска минимальных остовных деревьев в графе. Плюсом является то, что в алгоритм Крускала подкреплён также графической частью, позволяющей пользователю наглядно и быстро выполнить поиск минимального остовного дерева в неориентированном взвешенном графе, который он сам же и соберёт.

# Литература

1. Справка по классам исключений std::logic\_error, std::range\_error из стандартной библиотеки – [[http://www.cplusplus.com/reference/stdexcept](http://www.cplusplus.com/reference/stdexcept/logic_error/)].
2. Алгоритм Крускала – [<https://ru.wikipedia.org/wiki/Алгоритм_Крускала> ].
3. Алгоритм Ярника-Прима – [[https://ru.wikipedia.org/wiki/Алгоритм\_Прима](https://ru.wikipedia.org/wiki/Алгоритм_Примакала) ].
4. Д-кучи – [<https://en.wikipedia.org/wiki/Binary_heap>].

# Приложения

## main.cpp

#include <opencv2/opencv.hpp>

#include <stdio.h>

#include <cmath>

#include <cstdint>

#include <list>

#include <utility>

#include "graph.h"

#include "kruskal.h"

#define WIDTH 800

#define MAX\_VERTEX\_COUNT 60

#define CIRCLE\_RADIUS 20

#define WINDOW\_NAME "MST Visualis"

#define TRACKBAR\_NAME "Weight"

using std::list;

using std::pair;

using std::make\_pair;

using cv::Point;

using cv::line;

using cv::circle;

using cv::Mat;

using cv::putText;

using cv::Scalar;

using cv::waitKey;

using cv::namedWindow;

using cv::WINDOW\_AUTOSIZE;

using cv::Size;

using cv::getTextSize;

struct Edge {

uint32\_t e\_weight;

uint32\_t v\_from;

uint32\_t v\_to;

Point coord\_from;

Point coord\_to;

};

Edge make\_edge(uint32\_t e\_weight, uint32\_t v\_from, uint32\_t v\_to,

Point coord\_from, Point coord\_to) {

return {e\_weight, v\_from, v\_to, coord\_from, coord\_to};

}

int currentMaxVert = 0;

int coords[2];

char TestStr[100];

int slider;

int verticesCount;

Mat canvas;

list<pair<int, Point> > vertices;

list<Edge> edges;

Graph \*g;

list<pair<uint32\_t, uint32\_t> > points;

void redrawMem(int mode = 0) {

canvas = Mat::zeros(WIDTH, WIDTH, CV\_8UC3);

if (mode > 0) {

if (g != nullptr) delete g;

g = new Graph(vertices.size());

for (list<Edge>::iterator e = edges.begin(); e != edges.end(); ++e)

g->setEdge(e->v\_from, e->v\_to, e->e\_weight);

points = findMinSpanTreeKruskal(\*g);

}

bool flagToHighlight = false;

for (list<Edge>::iterator e = edges.begin(); e != edges.end(); ++e) {

int x[2], y[2];

x[0] = e->coord\_from.x;

y[0] = e->coord\_from.y;

x[1] = e->coord\_to.x;

y[1] = e->coord\_to.y;

flagToHighlight = false;

for (list<pair<uint32\_t, uint32\_t> >::iterator i = points.begin();

i != points.end(); ++i) {

if (((e->v\_from == i->first) && (e->v\_to == i->second)) ||

((e->v\_to == i->first) && (e->v\_from == i->second))) {

flagToHighlight = true;

break;

}

}

int line\_len\_x = x[1] - x[0], line\_len\_y = y[1] - y[0];

int linelen = sqrt(line\_len\_x \* line\_len\_x + line\_len\_y \* line\_len\_y);

line(canvas, Point(x[0] + 2 \* CIRCLE\_RADIUS \* line\_len\_x / (1 + linelen),

y[0] + 2 \* CIRCLE\_RADIUS \* line\_len\_y / (1 + linelen)),

Point(x[1] - 2 \* CIRCLE\_RADIUS \* line\_len\_x / (1 + linelen),

y[1] - 2 \* CIRCLE\_RADIUS \* line\_len\_y / (1 + linelen)),

((mode == 0) || (flagToHighlight)) ? Scalar(0, 255, 0)

: Scalar(64, 64, 64),

2, 1);

snprintf(TestStr, sizeof(TestStr), "%d", e->e\_weight);

putText(canvas, TestStr,

Point((x[0] + x[1]) / 2 + (20 \* line\_len\_y / (1 + linelen)),

(y[0] + y[1]) / 2 - (20 \* line\_len\_x / (1 + linelen))),

CV\_FONT\_NORMAL, 1, Scalar(255, 255, 255), 1, 1);

}

for (list<pair<int, Point> >::iterator i = vertices.begin();

i != vertices.end(); ++i) {

snprintf(TestStr, sizeof(TestStr), "%d", i->first);

Size textSize = getTextSize(TestStr, 1, 1, 1, NULL);

putText(canvas, TestStr, Point(i->second.x - textSize.width,

i->second.y + textSize.height / 2),

CV\_FONT\_NORMAL, 1, Scalar(255, 255, 255), 1, 1);

circle(canvas, Point(i->second.x, i->second.y), 30, Scalar(255, 128, 128),

1, CIRCLE\_RADIUS, 0);

}

imshow(WINDOW\_NAME, canvas);

}

static void onMouse(int event, int x, int y, int a, void \*vvvvv) {

static bool flag1 = false;

static bool flag2 = false;

static bool vertexStarted = false;

static bool edgeStarted = false;

if (edgeStarted) {

if (flag2) {

redrawMem();

int cords[2];

cords[0] = coords[0];

cords[1] = coords[1];

int line\_len\_x = x - cords[0], line\_len\_y = y - cords[1];

int linelen = hypot(line\_len\_x, line\_len\_y);

line(canvas,

Point(cords[0] + 2 \* CIRCLE\_RADIUS \* line\_len\_x / (1 + linelen),

cords[1] + 2 \* CIRCLE\_RADIUS \* line\_len\_y / (1 + linelen)),

Point(x, y), Scalar(0, 255, 0), 2, 1);

snprintf(TestStr, sizeof(TestStr), "%d", slider);

putText(canvas, TestStr,

Point((cords[0] + x) / 2 + (20 \* line\_len\_y / (1 + linelen)),

(cords[1] + y) / 2 - (20 \* line\_len\_x / (1 + linelen))),

CV\_FONT\_NORMAL, 1, Scalar(255, 255, 255), 1, 1);

imshow(WINDOW\_NAME, canvas);

}

}

if (vertexStarted) {

if (flag1) {

redrawMem();

snprintf(TestStr, sizeof(TestStr), "%d",

static\_cast<int>(vertices.size()));

Size textSize = getTextSize(TestStr, 1, 1, 1, NULL);

putText(canvas, TestStr,

Point(x - textSize.width, y + textSize.height / 2),

CV\_FONT\_NORMAL, 1, Scalar(255, 255, 255), 1, 1);

circle(canvas, Point(x, y), 30, Scalar(255, 128, 128), 1, CIRCLE\_RADIUS,

0);

imshow(WINDOW\_NAME, canvas);

}

}

static int v\_to = 0;

static int v\_from = 0;

switch (event) {

case CV\_EVENT\_LBUTTONDOWN: {

if (vertices.size() >= MAX\_VERTEX\_COUNT) break;

edgeStarted = false;

vertexStarted = true;

coords[0] = x;

coords[1] = y;

flag1 = true;

imshow(WINDOW\_NAME, canvas);

break;

}

case CV\_EVENT\_LBUTTONUP: {

if (vertices.size() >= MAX\_VERTEX\_COUNT) break;

flag1 = false;

if (!vertexStarted) break;

vertices.push\_back(make\_pair(vertices.size(), Point(x + 10, y - 5)));

redrawMem();

vertexStarted = false;

break;

}

case CV\_EVENT\_RBUTTONDOWN: {

if (slider == 0) break;

vertexStarted = false;

for (list<pair<int, Point> >::iterator i = vertices.begin();

i != vertices.end(); ++i) {

if (hypot(x - i->second.x, y - i->second.y) <= 2 \* CIRCLE\_RADIUS) {

v\_from = i->first;

coords[0] = i->second.x;

coords[1] = i->second.y;

edgeStarted = true;

flag2 = true;

}

}

break;

}

case CV\_EVENT\_RBUTTONUP: {

if (slider == 0) break;

snprintf(TestStr, sizeof(TestStr), "%d", slider);

for (list<pair<int, Point> >::iterator i = vertices.begin();

i != vertices.end(); ++i) {

if (hypot(x - i->second.x, y - i->second.y) <= 2 \* CIRCLE\_RADIUS) {

v\_to = i->first;

x = i->second.x;

y = i->second.y;

flag2 = false;

}

}

if (edgeStarted && !flag2 && !((coords[0] == x) && (coords[1] == y)))

edges.push\_back(make\_edge(slider, v\_from, v\_to,

Point(coords[0], coords[1]), Point(x, y)));

else

flag2 = false;

edgeStarted = false;

redrawMem();

break;

}

case CV\_EVENT\_LBUTTONDBLCLK: {

vertices.clear();

edges.clear();

points.clear();

redrawMem();

}

case CV\_EVENT\_MBUTTONDOWN:

redrawMem(1);

break;

}

}

static void onTrackBar(int value) { slider = value; }

int main() {

verticesCount = 0;

CvFont \*font = new CvFont;

cvInitFont(font, CV\_FONT\_VECTOR0, 0.5f, 0.8f, 0, 1, 8);

slider = 0;

canvas = Mat::zeros(WIDTH, WIDTH, CV\_8UC3);

snprintf(TestStr, sizeof(TestStr), "Hapucyu");

putText(canvas, TestStr,

Point(400 - getTextSize(TestStr, 1, 1, 1, NULL).width, 400),

CV\_FONT\_NORMAL, 1, Scalar(255, 255, 255), 1, 1);

circle(canvas, Point(400, 400), 200, Scalar(255, 0, 255), 1, 8, 0);

namedWindow(WINDOW\_NAME, WINDOW\_AUTOSIZE);

cvCreateTrackbar(TRACKBAR\_NAME, WINDOW\_NAME, &slider, 150, onTrackBar);

cvSetMouseCallback(WINDOW\_NAME, onMouse, NULL);

imshow(WINDOW\_NAME, canvas);

waitKey(0);

return 0;

}

## graph.h

#ifndef INCLUDE\_GRAPH\_H\_

#define INCLUDE\_GRAPH\_H\_

#include <cstdint>

class Graph {

private:

uint32\_t size;

uint32\_t\*\* connMat;

public:

explicit Graph(uint32\_t size);

explicit Graph(const Graph& rhs);

~Graph();

void setEdge(uint32\_t vFirst, uint32\_t vSecond, uint32\_t eWeight);

void unsetEdge(uint32\_t vFirst, uint32\_t vSecond);

uint32\_t getEdgeWeight(uint32\_t vFirst, uint32\_t vSecond) const;

uint32\_t getSize() const;

};

#endif // INCLUDE\_GRAPH\_H\_

## graph.cpp

#include "graph.h"

Graph::Graph(uint32\_t size) {

this->size = size;

connMat = new uint32\_t \*[size];

for (uint32\_t i = 0; i < size; ++i) connMat[i] = new uint32\_t[size];

for (uint32\_t i = 0; i < size; ++i)

for (uint32\_t j = 0; j < size; ++j) connMat[i][j] = 0;

}

Graph::Graph(const Graph &rhs) {

size = rhs.size;

connMat = new uint32\_t \*[size];

for (uint32\_t i = 0; i < size; ++i) connMat[i] = new uint32\_t[size];

for (uint32\_t i = 0; i < size; ++i)

for (uint32\_t j = 0; j < size; ++j) connMat[i][j] = rhs.connMat[i][j];

}

Graph::~Graph() {

for (uint32\_t i = 0; i < size; ++i) delete[] connMat[i];

delete[] connMat;

}

void Graph::setEdge(uint32\_t vFirst, uint32\_t vSecond, uint32\_t eWeight) {

connMat[vFirst][vSecond] = connMat[vSecond][vFirst] = eWeight;

}

void Graph::unsetEdge(uint32\_t vFirst, uint32\_t vSecond) {

connMat[vFirst][vSecond] = connMat[vSecond][vFirst] = 0;

}

uint32\_t Graph::getEdgeWeight(uint32\_t vFirst, uint32\_t vSecond) const {

return connMat[vFirst][vSecond];

}

uint32\_t Graph::getSize() const { return size; }

## graph\_mat\_ident.h

#ifndef INCLUDE\_GRAPH\_MAT\_IDENT\_H\_

#define INCLUDE\_GRAPH\_MAT\_IDENT\_H\_

#include <cstdint>

class GraphMatIdent {

uint32\_t countVertices;

uint32\_t countEdges;

uint32\_t \*\*idenMat;

public:

GraphMatIdent(uint32\_t countVertices, uint32\_t countEdges);

GraphMatIdent(const GraphMatIdent &rhs);

~GraphMatIdent();

void setEdge(uint32\_t vFirst, uint32\_t vSecond, uint32\_t eNumber,

uint32\_t eWeight);

uint32\_t getEdgeWeight(uint32\_t vFirst, uint32\_t eNumber) const;

uint32\_t getEdgesSize() const;

uint32\_t getVerticesSize() const;

};

#endif // INCLUDE\_GRAPH\_MAT\_IDENT\_H\_

## graph\_mat\_ident.cpp

#include "graph\_mat\_ident.h"

GraphMatIdent::GraphMatIdent(uint32\_t countVertices, uint32\_t countEdges) {

this->countEdges = countEdges;

this->countVertices = countVertices;

idenMat = new uint32\_t \*[countVertices];

for (uint32\_t i = 0; i < countVertices; ++i)

idenMat[i] = new uint32\_t[countEdges];

for (uint32\_t i = 0; i < countVertices; ++i)

for (uint32\_t j = 0; j < countEdges; ++j) idenMat[i][j] = 0;

}

GraphMatIdent::GraphMatIdent(const GraphMatIdent &rhs) {

countEdges = rhs.countEdges;

countVertices = rhs.countVertices;

idenMat = new uint32\_t \*[countVertices];

for (uint32\_t i = 0; i < countVertices; ++i)

idenMat[i] = new uint32\_t[countEdges];

for (uint32\_t i = 0; i < countVertices; ++i)

for (uint32\_t j = 0; j < countEdges; ++j) idenMat[i][j] = rhs.idenMat[i][j];

}

GraphMatIdent::~GraphMatIdent() {

for (uint32\_t i = 0; i < countVertices; ++i) delete[] idenMat[i];

delete[] idenMat;

}

void GraphMatIdent::setEdge(uint32\_t vFirst, uint32\_t vSecond, uint32\_t eNumber,

uint32\_t eWeight) {

idenMat[vFirst][eNumber] = idenMat[vSecond][eNumber] = eWeight;

}

uint32\_t GraphMatIdent::getEdgeWeight(uint32\_t vFirst, uint32\_t eNumber) const {

vFirst %= countVertices;

eNumber %= countEdges;

return idenMat[vFirst][eNumber];

}

uint32\_t GraphMatIdent::getEdgesSize() const { return countEdges; }

uint32\_t GraphMatIdent::getVerticesSize() const { return countVertices; }

## heap.h

#ifndef INCLUDE\_HEAP\_H\_

#define INCLUDE\_HEAP\_H\_

#include <exception>

#include <new>

#include <stdexcept>

#include <algorithm>

#include "heap\_declaration.h"

namespace akon {

template <typename DataType, uint8\_t base>

Heap<DataType, base>::Heap(uint32\_t size) {

data = new DataType[size];

if (data == nullptr)

throw std::runtime\_error(

"Heap::Heap(uint32\_t): Can't allocate memory for an array");

currSize = -1;

heapSize = size;

}

template <typename DataType, uint8\_t base>

Heap<DataType, base>::Heap(const Heap &rhs) {

heapSize = rhs.heapSize;

currSize = rhs.currSize;

data = new DataType[rhs.heapSize];

if (data == nullptr)

throw std::runtime\_error(

"Heap::Heap(const Heap&): Can't allocate memory for an array");

for (int i = 0; i < currSize; ++i) data[i] = rhs.data[i];

}

template <typename DataType, uint8\_t base>

void Heap<DataType, base>::insert(const DataType &rhs) {

if ((currSize + 1) == heapSize) resize(heapSize \* 2);

data[++currSize] = rhs;

int indx = currSize, buf;

while (data[buf = ((indx - 1) / 3)] > rhs) {

std::swap(data[buf], data[indx]);

indx = buf;

}

}

template <typename DataType, uint8\_t base>

const DataType &Heap<DataType, base>::top() const {

return data[0];

}

template <typename DataType, uint8\_t base>

const DataType Heap<DataType, base>::pop() {

if (currSize < 0)

throw std::runtime\_error("Heap::pop(): popping element from empty Heap");

DataType temp = data[0];

data[0] = data[currSize--];

int indx = 0;

bool childLess = true;

while (childLess) {

int minChildIndx = (indx \* base) + 1;

if (minChildIndx > currSize) break;

childLess = false;

for (int i = minChildIndx, j = 0; (j < base), (i <= currSize); i++, j++) {

if (data[i] < data[minChildIndx]) {

minChildIndx = i;

childLess = true;

}

}

if (data[indx] > data[minChildIndx]) {

std::swap(data[indx], data[minChildIndx]);

indx = minChildIndx;

} else {

childLess = false;

}

}

return temp;

}

template <typename DataType, uint8\_t base>

const Heap<DataType, base> &Heap<DataType, base>::operator=(const Heap &rhs) {

if (data != nullptr) delete[] data;

heapSize = rhs.heapSize;

currSize = rhs.currSize;

data = new DataType[rhs.heapSize];

if (data == nullptr)

throw std::runtime\_error(

"Heap::operator=(const Heap&): Can't allocate memory for an array");

for (int i = 0; i < currSize; ++i) data[i] = rhs.data[i];

}

template <typename DataType, uint8\_t base>

void Heap<DataType, base>::resize(int32\_t size) {

if (size <= currSize)

throw std::length\_error(

"Heap::resize(uint32\_t): Bad new value of size(it can't be less than "

"the old one)");

DataType \*updDataArray = new DataType[size];

if (updDataArray == nullptr)

throw std::runtime\_error(

"Heap::resize(uint32\_t): Can't allocate memory for an array");

for (int i = 0; i < currSize; ++i) updDataArray[i] = data[i];

delete[] data;

data = updDataArray;

heapSize = size;

}

template <typename DataType, uint8\_t base>

constexpr uint32\_t Heap<DataType, base>::getSize() const {

return currSize + 1;

}

template <typename DataType, uint8\_t base>

constexpr uint32\_t Heap<DataType, base>::getHeapSize() const {

return heapSize;

}

template <typename DataType, uint8\_t base>

constexpr bool Heap<DataType, base>::isEmpty() const {

return currSize <= -1;

}

} // namespace akon

#endif // INCLUDE\_HEAP\_H\_

## heap\_declaration.h

#ifndef INCLUDE\_HEAP\_DECLARATION\_H\_

#define INCLUDE\_HEAP\_DECLARATION\_H\_

#if (\_\_cplusplus < 201103L)

#error "C++11 required"

#endif

#include <cstdint>

namespace akon {

template <typename DataType, uint8\_t base = 3>

class Heap {

private:

DataType \*data;

int32\_t heapSize;

int32\_t currSize;

public:

explicit Heap(uint32\_t size = 256);

explicit Heap(const Heap &rhs);

void insert(const DataType &rhs);

const DataType &top() const;

const DataType pop();

const Heap &operator=(const Heap &rhs);

void resize(int32\_t size);

constexpr uint32\_t getSize() const;

constexpr uint32\_t getHeapSize() const;

constexpr bool isEmpty() const;

};

} // namespace akon

#endif // INCLUDE\_HEAP\_DECLARATION\_H\_

## jarnik.h

#ifndef INCLUDE\_JARNIK\_H\_

#define INCLUDE\_JARNIK\_H\_

#include <cstdint>

#include <list>

#include <utility>

#include "graph\_mat\_ident.h"

using std::list;

using std::pair;

list<pair<uint32\_t, uint32\_t> > findMinSpanTreeJarnik(const GraphMatIdent &gid);

#endif // INCLUDE\_JARNIK\_H\_

## jarnik.cpp

#include <utility>

#include <list>

using std::list;

using std::pair;

using std::make\_pair;

#include "graph.h"

#include "graph\_mat\_ident.h"

#include "jarnik.h"

#include "priority\_queue.h"

#include "unionfind.h"

list<pair<uint32\_t, uint32\_t> > findMinSpanTreeJarnik(

const GraphMatIdent& gid) {

Graph g(gid.getVerticesSize());

for (uint32\_t i = 0; i < gid.getEdgesSize(); ++i)

for (uint32\_t j = 0; j < gid.getVerticesSize(); ++j)

if (gid.getEdgeWeight(j, i) > 0) {

for (uint32\_t k = j + 1; k < gid.getVerticesSize(); ++k)

if (gid.getEdgeWeight(k, i) > 0) {

g.setEdge(j, k, gid.getEdgeWeight(j, i));

break;

}

break;

}

list<pair<uint32\_t, uint32\_t> > res;

UnionFind uf(g.getSize());

akon::PriorityQueue<pair<int, pair<int, int> > > pq;

for (uint32\_t i = 0; i < g.getSize(); ++i)

if (g.getEdgeWeight(0, i) != 0)

pq.insert(make\_pair(g.getEdgeWeight(0, i), make\_pair(0, i)));

while (!pq.isEmpty()) {

uint32\_t vFrom = pq.top().second.first;

uint32\_t vTo = pq.top().second.second;

pq.pop();

if (!uf.find(vFrom, vTo)) {

for (uint32\_t i = 0; i < g.getSize(); ++i)

if (g.getEdgeWeight(vTo, i) != 0 && !uf.find(vFrom, i))

pq.insert(make\_pair(g.getEdgeWeight(vTo, i), make\_pair(vTo, i)));

uf.unite(vFrom, vTo);

res.push\_back(make\_pair(vFrom, vTo));

}

}

return res;

}

## kruskal.h

#ifndef INCLUDE\_KRUSKAL\_H\_

#define INCLUDE\_KRUSKAL\_H\_

#include <cstdint>

#include <list>

#include <utility>

#include "graph.h"

using std::list;

using std::pair;

list<pair<uint32\_t, uint32\_t> > findMinSpanTreeKruskal(const Graph& g);

#endif // INCLUDE\_KRUSKAL\_H\_

## kruskal.cpp

#include <utility>

#include <list>

using std::list;

using std::pair;

using std::make\_pair;

#include "kruskal.h"

#include "priority\_queue.h"

#include "unionfind.h"

list<pair<uint32\_t, uint32\_t> > findMinSpanTreeKruskal(const Graph &g) {

list<pair<uint32\_t, uint32\_t> > res;

akon::PriorityQueue<pair<uint32\_t, pair<uint32\_t, uint32\_t> > >

pq;

for (uint32\_t i = 0; i < g.getSize(); ++i)

for (uint32\_t j = i + 1; j < g.getSize(); ++j)

if (g.getEdgeWeight(i, j) != 0)

pq.insert(make\_pair(g.getEdgeWeight(i, j), make\_pair(i, j)));

UnionFind uf(g.getSize());

while (!pq.isEmpty()) {

int vFrom = pq.top().second.first, vTo = pq.top().second.second;

if (!uf.find(vFrom, vTo)) {

uf.unite(vFrom, vTo);

res.push\_back(make\_pair(vFrom, vTo));

}

pq.pop();

}

return res;

}

## unionfind.h

#ifndef INCLUDE\_UNIONFIND\_H\_

#define INCLUDE\_UNIONFIND\_H\_

#include <cstdint>

class UnionFind {

private:

uint32\_t size;

uint32\_t \*unionParents;

uint32\_t \*unionSizes;

public:

explicit UnionFind(uint32\_t quantity);

~UnionFind();

bool find(uint32\_t first, uint32\_t second);

bool unite(uint32\_t first, uint32\_t second);

uint32\_t getSize();

};

#endif // INCLUDE\_UNIONFIND\_H\_

## unionfind.cpp

#include "unionfind.h"

UnionFind::UnionFind(uint32\_t quantity) {

size = quantity;

unionParents = new uint32\_t[size];

unionSizes = new uint32\_t[size];

for (uint32\_t i = 0; i < size; ++i) {

unionParents[i] = i;

unionSizes[i] = 1;

}

}

UnionFind::~UnionFind() {

delete[] unionParents;

delete[] unionSizes;

unionParents = unionSizes = nullptr;

}

bool UnionFind::find(uint32\_t first, uint32\_t second) {

if ((first >= size) || (second >= size)) return false; // throw

while (first != unionParents[first]) first = unionParents[first];

while (second != unionParents[second]) second = unionParents[second];

return first == second;

}

bool UnionFind::unite(uint32\_t first, uint32\_t second) {

if ((first >= size) || (second >= size)) return false; // throw

while (first != unionParents[first]) first = unionParents[first];

while (second != unionParents[second]) second = unionParents[second];

if (first == second) return false;

if (unionSizes[first] > unionSizes[second]) {

unionSizes[first] += unionSizes[second];

unionParents[second] = unionParents[first];

} else {

unionSizes[second] += unionSizes[first];

unionParents[first] = unionParents[second];

}

return true;

}

uint32\_t UnionFind::getSize() { return size; }