Пенза 2020

Выполнил:

студент группы 19ВВ3

Ланцов А.С.

Принял:

профессор д.т.н.

Митрохин М.А.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задача»

на тему «Алгоритм раскрашивания графа»

Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Содержание

[**Реферат** 2](#_Toc59751413)

[Введение 3](#_Toc59751414)

[1 Теоретическая часть 4](#_Toc59751415)

[2 Постановка задачи 6](#_Toc59751416)

[3 Описание алгоритма программы 7](#_Toc59751417)

[4 Описание работы программы 8](#_Toc59751418)

[**Сторона клиента** 8](#_Toc59751419)

[**Canvas** 8](#_Toc59751420)

[**Описание классов** 9](#_Toc59751421)

[**Вывод графа** 11](#_Toc59751422)

[**Изменение графа** 11](#_Toc59751423)

[**Связь стороны клиента и сервера** 12](#_Toc59751424)

[**CGI** 12](#_Toc59751425)

[**Ajax** 13](#_Toc59751426)

[**Сторона сервера** 13](#_Toc59751428)

[Руководство пользователя 22](#_Toc59751429)

[Тестирование 31](#_Toc59751430)

[**Ручной просчет алгоритма** 31](#_Toc59751431)

[**Проверка на типовые ошибки** 49](#_Toc59751432)

[Заключение 51](#_Toc59751433)

[Список литературы 52](#_Toc59751434)

[**Приложение А** 53](#_Toc59751435)

[**Листинг** 53](#_Toc59751436)

# **Реферат**

Отчет 86 стр, 55 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, НЕЗВИСИСИМЫЕ МНОЖЕСТВА, РАСКРАШИВАНИЕ ГРАФА, ХРОМАТИЧЕСКОЕ ЧИСЛО

Цель исследования – разработка программы, осуществляющая правильную минимальную раскраску графа используя алгоритм поиска максимальных независимых множеств.

В работе рассмотрены правила поиска всех независимых множеств. Установлено, что с помощью данного алгоритма возможно правильно раскрасить граф наименьшим количеством цветов.

Введение

Раскраской вершин графа называется назначение цветов (в общем случае - меток) его вершинам. Ставится задача раскраски в наименьшее число цветов так, чтобы любые две смежные вершины имели разные цвета.

Наименьшее число цветов раскраски называется *хроматическим числом графа* и обозначается χ(*G*).

Правильная раскраска с минимальным количеством цветов в общем случае - сложная задача. Фактически, до сих пор не было найдено простого способа характеризации κ-хроматического графа. (Метод грубой силы с использованием всех возможных комбинаций, конечно, может применяться всегда, как и в любой комбинаторной задаче, но он крайне неэффективен на больших графах.

Применения раскраски графа:

1. Составление расписаний. Например, лекции – вершины, которые смежны тогда, когда они не могут проходить одновременно. Необходимо найти наименьшее число красок – пар.
2. Распределение регистров, в которой наиболее часто используемые переменные компилируемой программы хранятся в быстродействующих регистрах процессора. В идеальном случае переменные хранятся в регистрах так, что они все находятся в регистрах во время их использования.

Компилятор строит граф, где вершины соответствуют переменным, а грань соединяет две из них, если они нужны в один и тот же момент времени. Если этот граф *k*-хроматический, то переменные могут храниться в *k*-регистрах.

1. Решение головоломки Судоку можно рассматривать как завершение раскраски 9 цветами заданного графа из 81 вершины.

1 Теоретическая часть

Правильная раскраска графа естественным образом индуцирует разбиение вершин на различные подмножества. Например, раскраска на рис. производит разбиение {*v*1, *v*4}, {*v*2}, и {*v*3, *v*5}. Такое подмножество вершин называется независимым множеством.

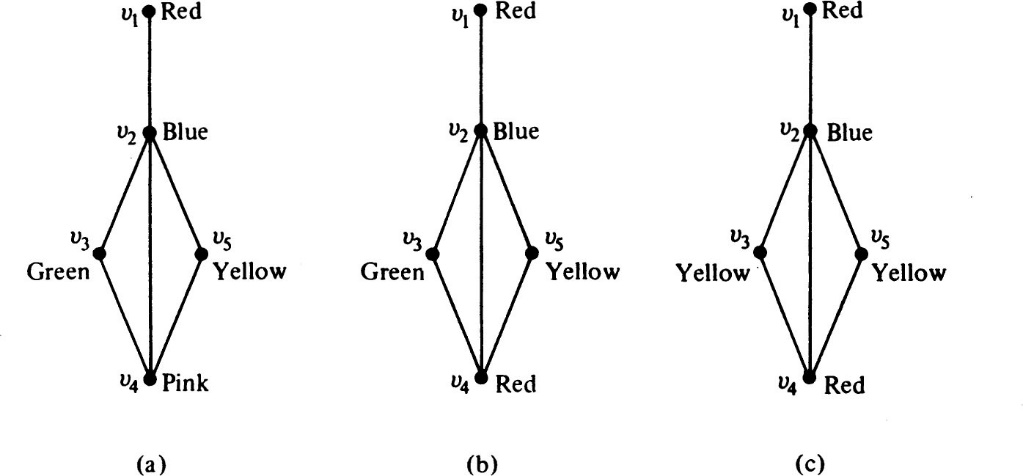


Рисунок 1 Примеры раскраски

Независимое множество вершин графа – такое множество, в котором любая пара вершин не смежная.

Максимальное независимое множество – такое независимое множество, что в него нельзя добавить другую вершину графа, не нарушив его независимость.

Разумным методом поиска максимального независимого множества в графе G будет начать с любой вершины v графа G в множестве. Далее добавлять еще вершины к набору, выбирая на каждом этапе вершину, не смежную ни с одной из уже выбранных. Эта процедура в конечном итоге даст максимальный независимый набор. Однако это множество не обязательно является максимальным независимым множеством с наибольшим числом вершин.

Хроматическое разбиение. (дословно с анг. *Chromatic Partitioning*) - нахождение всех возможных независимых множества графа и выделение из них наименьшего количества множеств, которые при объединении включают все вершины графа. Так, для графа на рис. имеем следующие максимальные независимые множества: *acdf*, *acdg*, *bg*, *bf*, *ae*.

и их комбинации:

{(*a*, *c*, *d*, *f*), (*b*, *g*), (*e*)},  
{(*a*, *c*, *d*, *g*), (*b*, *f*), (*e*)},  
{(*c*, *d*, *f*), (*b*, *g*), (*a*, *e*)},  
{(*c*, *d*, *g*), (*b*, *f*), (*a*, *e*)}.

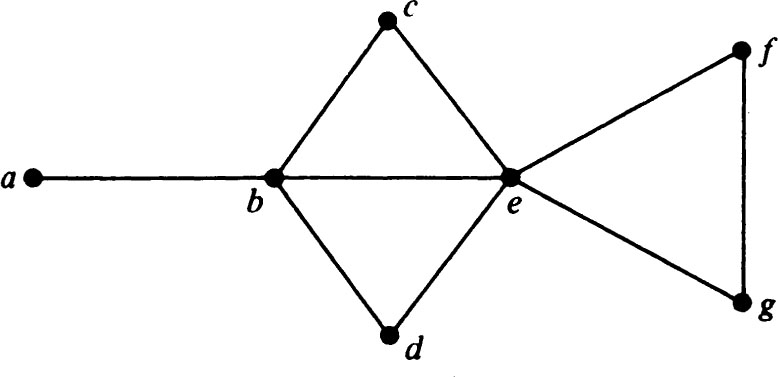


Рисунок 2 Граф G

Метод хроматического разбиения (требующий перечисления всех максимальных независимых множеств) неэффективен и требует чрезмерно больших объемов компьютерной памяти.

Для правильной раскраски графа, достаточно найти лишь одну из таких комбинаций максимальных независимых множеств и раскрасить граф в минимальное количество цветов. Имеем следующий алгоритм:

1. Найти в графе максимальное независимое множество вершин.
2. Раскрасить найденное множество в один цвет.
3. Удалить найденные вершины из графа.
4. Если остались не раскрашенные вершины, то повторить пункты 1..3.

Этот алгоритм рассмотрен в данной работе.

2 Постановка задачи

Требуется разработать алгоритм поиска максимальных независимых множеств для раскрашивания графа. Реализовать графический интерфейс для интуитивного взаимодействия с ней пользователя, а также визуализацию результата работы программы.

Исходный граф должен задаваться матрицей смежности. При генерации данных должны быть предусмотрены граничные условия. Пользователь должен иметь выбор ввода графа – вручную, или с рандомно.

Устройства ввода – клавиатура и мышь.

Непосредственно алгоритм будет реализован на языке Си, так он компилируем, предоставляет прямой доступ к потокам вывода и прозрачную работу с памятью. Графический интерфейс будет написан с использованием веб технологий HTML, CSS, Javascript. Данный выбор обусловлен простой реализацией кроссплатформенности приложения, а также уже имеющимся опытом работы с данными технологиями ранее. Связь бэкенда и фронтенда будет достигнута по средствам технологии CGI (*Common Gateway Interface*).

В качестве среды программирования был выбран программный продукт JetBrains CLion.

3 Описание алгоритма программы

Псевдокод алгоритма:

**Пока** (размер графа > 0){

set\_size = 0;

**Пока** (v\_index != -1){

visited[v\_index]=1; отмечаем вершину под индексом v\_index, как посещенную;

current\_set[set\_size] = название вершины под индексом v\_index;

**Для** (каждой вершины **i** в графе){

**Если** ((вершина **a[v\_index][i]** не ноль или **a[i][v\_index]** не ноль) и не отмечена, как посещенная), **то**{

visited[i]=1; отмечаем вершину под индексом **i**, как посещенную;

}

}

увеличиваем set\_size на один;

v\_index = индекс первой не посещенной вершины, или -1 (см. функция fst\_false);

}

обнуляем массив visited;

**Для** (set\_size элементов **i** массива current\_set){

вывести через запятую следующий current\_set[i]

удалить вершину current\_set[i] из графа

}

вывести перевод на новую строку

v\_index = 0;

}

4 Описание работы программы

## **Сторона клиента**

Клиентская часть программы представляет собой HTML разметку страницы и Javascript код работающий с главным элементом - canvas. Меню реализовано посредствам взаимодействия HTML-Javascript.

Для упрощения разметки и стилизации страницы использовалась библиотека стилей Twitter Bootstrap.

### **Canvas**

Основной элемент страницы - Сanvas.

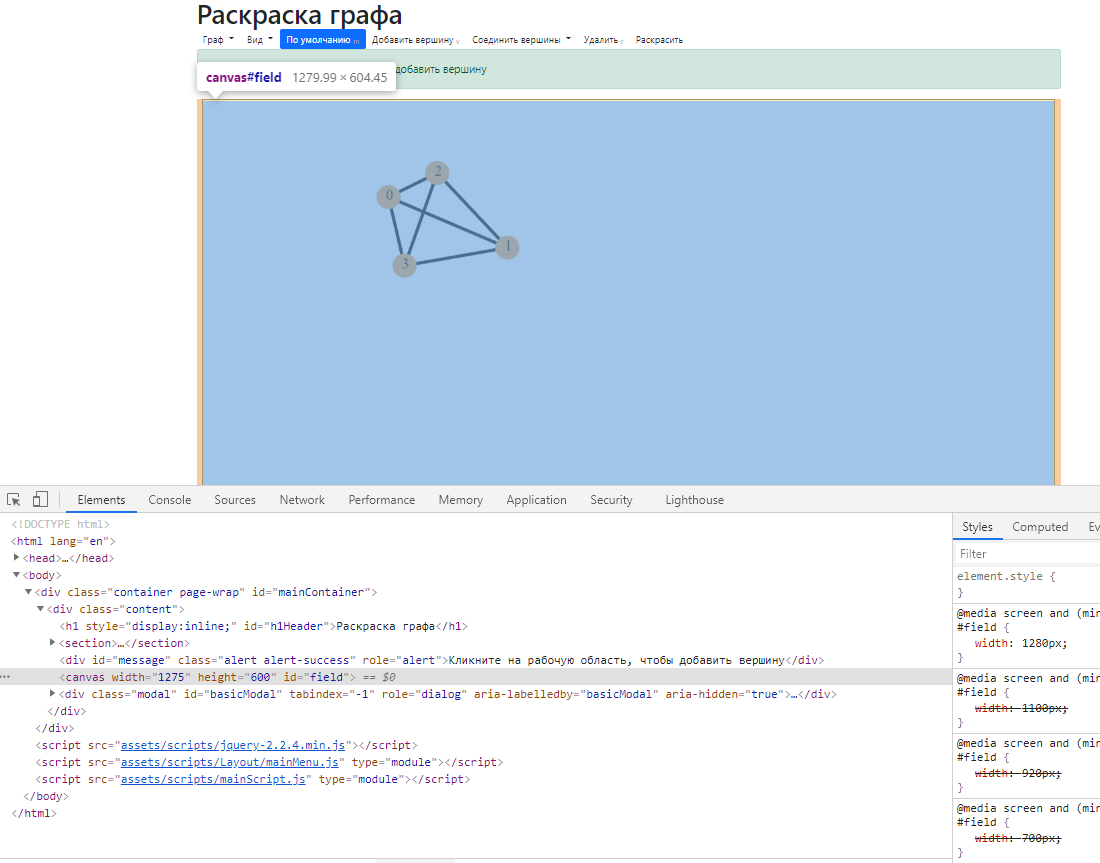


Рисунок 3 Элемент Canvas

Canvas — это HTML элемент, использующийся для рисования графики средствами языков программирования. Он может, к примеру, использоваться для рисования графов, создания коллажей или анимации.

Впервые использовался компанией Apple для создания Mac OS X Dashboard, а затем был реализован в Web-браузерах. На сегодняшний день все основные браузеры поддерживают работу с canvas.

В данной программе с помощью canvas происходит рисование вершин и ребер графа.

Элемент canvas имеет [метод](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement#Method) getContext(), используется для получения контекста визуализации и ее функции рисования. Он принимает один параметр, тип контекста. Для 2D графики, которая используется в данной работе, использовалась метка "2d".

### **Описание классов**

Данные об объектах для рисования объединены в классы и имеют следующую иерархию:

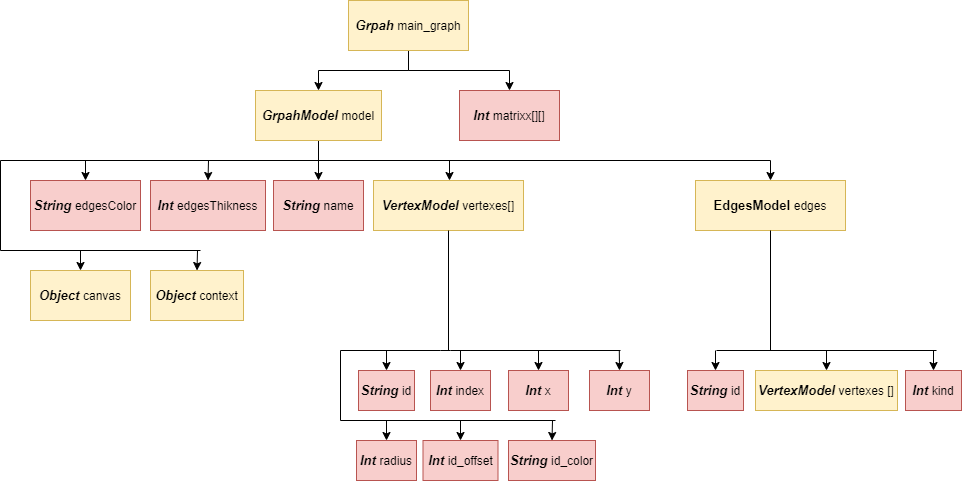


Таблица 1 Иерархия объектов клиентской стороны. (Бежевый цвет - объекты, красный - базовые типы)

Graph

Глобальная переменная *main\_graph* класса *Graph с*одержит всю информацию о графе.

Поля:

1. *model* – модель графа. Хранит информацию для рисования.
2. *matirx –* двумерный массив. Хранит матрицу смежности графа, для раскраски.

#### GraphModel

Содержит модели вершин и ребер графа, и параметры их вывода.

Поля:

1. *name* – название графа.
2. *vertexes -*  массив моделей вершин для рисования
3. *edges -* массив моделей ребер для рисования
4. *edgeThickness –* толщина моделей ребер
5. *canvas –* Объект canvas отвечающий за вывод изображения
6. *context -* Объект context, в котором происходит рисование

#### VertexModel

Содержит информацию о модели вершины

Поля:

1. *id* – название вершины, которое будет видно после его вывода
2. *index –* индекс в массиве вершин
3. *context –* Объект context
4. *x –* положение по оси x
5. *y – положение по оси y*
6. *radius –* радиус круга модели графа
7. *main\_color* – цвет круга
8. *id\_color –* цвет названия модели
9. *id\_offset –* отступ для рисования названия

#### EdgeModel

Содержит информацию о модели ребра

Поля:

1. *id* – названия вершины. И
2. *vertexes –* массив из двух вершин которые соединяет ребро
3. *kind -* вид ребра (ориентированное или нет)

### **Вывод графа**

Отрисовка графа происходит обращением к методу draw() класса Graph.

Производится полная очистка холста. Затем рисуются ребра по обращению к методу drawEdges класса ModelGraph. После рисуются вершины по обращению к методу drawVertexes класса ModelGraph. Классы моделей вершин и ребер также имеют метод draw(). Таким образом происходит последовательный вывод вначале ребер, а потом вершин.

Отрисовка графа происходит:

1. При добавлении/удалении вершины
2. При добавлении ребра
3. При перемещении вершины
4. При задании графа, с помощью матрицы смежности
5. После раскраски

### **Изменение графа**

Глобальная переменная fieldState задает режим холста. Режимы меняются соответственно выбору действия в главном меню.

Режимы:

1. *“move” –* перемещение вершины
2. *“addV” –* добавление вершины
3. *“addEO” –* добавление ориентированного ребра
4. *“addENO”* – добавление неориентированного ребра
5. *“remove”* – удаление ребра

Почти любые действия пользователя на странице можно считывать и обрабатывать. Для этого у каждого html элемента есть метод addEventlistner(). Он принимает 3 значения: метка процесса для прослушивания, и функция обработчик.

Так, элемент canvas “реагирует” на нажатие кнопки мыши, перемещение, и отпускание кнопки, и в зависимости от режима холста производятся соответствующие действия над моделями, и матрицей графа, подробно рассматривать которые, я считаю не обязательно, в рамках данной работы.

## **Связь стороны клиента и сервера**

### **CGI**

CGI (от англ. Common Gateway Interface — «общий интерфейс шлюза») — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято называть шлюзом, хотя многие предпочитают названия «скрипт» (сценарий) или «CGI-программа».

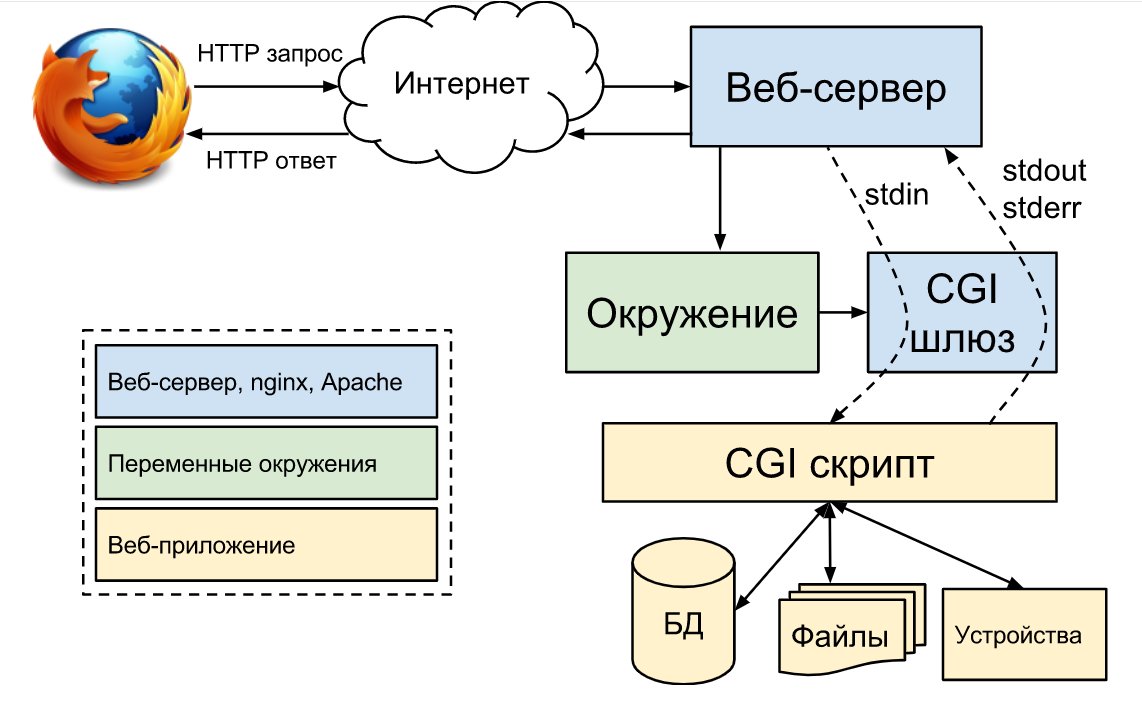


Рисунок 4 Схема работы CGI

Обобщенный алгоритм работы через CGI можно представить в следующем виде:

1. Клиент запрашивает CGI-приложение по его URI.
2. Веб-сервер принимает запрос и устанавливает переменные окружения, через них приложению передаются данные и служебная информация.
3. Веб-сервер перенаправляет запросы через стандартный поток ввода (stdin) на вход вызываемой программы.
4. CGI-приложение выполняет все необходимые операции и формирует результаты в виде HTML.
5. Сформированный гипертекст возвращается веб-серверу через стандартный поток вывода (stdout). Сообщения об ошибках передаются через stderr.
6. Веб-сервер передает результаты запроса клиенту.

### **Ajax**

### Ajax (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Asynchronous Javascript and XML* —«асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), заключающийся в «фоновом» обмене данными браузера с веб-сервером.

По нажатии кнопки раскрасить на стороне клиента срабатывает eventlistener, который отдает управление функции color. В ней матрица заданного графа конвертируется в строку и создается post запрос для отправки на сервер по технологии ajax. Функция обработчик ответа с сервера принимает строку с независимыми множествами графа. Строка конвертируется в массивы (множеста) согласно которым моделям вершин присваиваются цвета.

## **Сторона сервера**

Входная точка CGI скрипта функция int main:

Main

**int** main(**int** argc,**char**\*\* argv) {  
 matrix\_graph\* graph = init\_graph\_from\_cgi\_client();  
 **cnt\_type**(**"text/plain"**);  
 indp\_sets\_out(graph);  
 matrixGraph\_free(graph);  
 **return** 0;  
}

Первая строчка инициализирует структуру matrix\_graph

typedef struct matrix\_graph\_s{  
 int\*\* matrix;  
 int\* vertexes;  
 size\_t size;  
}matrix\_graph;

init\_graph\_from\_cgi\_client()

В функции init\_graph\_from\_cgi\_client() определяем размер переданной клиентом строки, с помощью обращения к переменной среды “CONTENT\_LENGTH”, с помощью функции getenv. Т.к. все переменные окружения имеют строковый тип, переводим строку в число, с помощью функции atoi().

matrix\_graph\* init\_graph\_from\_cgi\_client(){  
 int data\_length = **CONTENT\_LENGTH**;  
 char\* data = post\_data(data\_length);  
 if (!data\_length){printf("error,%d\n",data\_length);exit(1);}  
 matrix\_graph\* graph = init\_graph\_from\_string(data);  
 **row**(graph->vertexes,graph->size,0,i)  
 free(data);  
 return graph;  
}

#define **CONTENT\_LENGTH** atoi(getenv("CONTENT\_LENGTH"))

Функция post\_data() считывает данные из стандартного потока ввода:

#define **REQ\_MTD** getenv("REQUEST\_METHOD")

char\* post\_data(int content\_len){  
 if (!strcmp(**REQ\_MTD**,"POST")){//проверяем метод запроса  
 char\* post\_data = **NULL**;  
 if (content\_len != 0) {  
 post\_data = (char\*)malloc(content\_len);  
  
 char next\_char = **NULL**;  
 int count = 0;  
 while (next\_char!=**EOF** && count<content\_len){  
 next\_char = fgetc(**stdin**);  
 post\_data[count] = next\_char;  
 count ++;  
 }  
 post\_data[content\_len-1]=0;  
 }  
 return post\_data;  
 }  
 return **NULL**;  
}

Далее, если данные считались неверно, то завершаем выполнения программы с кодом 1.

Если данный были считаны верно, выделяем память под структуру, и матрицу графа, переводим переданную матрицу из строкового типа в двумерный массив типа int. Это происходит в функции init\_graph\_from\_string(data):

matrix\_graph\* init\_graph\_from\_string(char\* str){  
 int order=0;  
 char next\_char=' ';  
  
 int i=0;//определяем размер матрицы  
 do{  
 if (next\_char!=' '){  
 order++;  
 }  
 next\_char = str[i];  
 i++;  
 }while (next\_char!='\n');  
  
 matrix\_graph\* graph = matrixGraph\_create(order);// выделяем память под граф  
  
 int str\_len = strchr(str,'\n') - str;  
 for (int i = 0; i < order; ++i) {//переводим строку в матрицу  
 for (int j = 0; j < order; ++j){  
 next\_char = str[2\*j+i\*(str\_len+1)];  
 graph->matrix[i][j] = (int)(next\_char-'0');  
 }  
 }  
  
 return graph;  
}

Выделение памяти:

Некоторые макросы:

#define **new**(type) malloc(sizeof(type))

#define **new\_block**(type,how\_much) malloc(sizeof(type)\*how\_much)

int\*\* matrix\_create(int size){  
 int \*\*matrix = (int \*\*) malloc(size \* sizeof(int \*));  
 for (int i = 0; i < size; i++){

matrix[i] = (int \*) malloc(size \* sizeof(int));

}  
 return matrix;  
}

matrix\_graph\* matrix\_graph\_create(size\_t size){  
 matrix\_graph\* graph = **new**(matrix\_graph);  
 graph->matrix = matrix\_create(size);  
 graph->vertexes = **new\_block**(int,size);  
 graph->size = size;  
 return graph;  
}

Макрос row() заполняет массив graph->vertexes числами от 0 до size

#define **fill**(block,block\_size,value,i) if (block && block\_size){for(size\_t i=0; i<block\_size; i++){block[i] = value;}}

#define **row**(dst,size,begin,index) **fill**(dst,size, index +begin, index)

Массив, vertexes нужен для того, чтобы при удалении вершин во время выполнения алгоритма выводить индексы вершин исходного графа, а не модифицированного.

После того, как граф был инициализирован выполняем следующую инструкцию:

printf("Content-type: text/plain\n\n");

Строка ”Content-type: text/plain\n\n” — http-заголовок, задающий тип содержимого (mime-type). Удвоенный символ разрыва строки (\n\n) — обязателен, он отделяет заголовки от тела сообщения. В нашем случае тип сообщения – обычный текст.

Далее происходит выполнение самого алгоритма в функции indp\_sets\_out(), с последовательным выводом найденных независимых множеств:

void indp\_sets\_out(matrix\_graph\* graph){  
 int\* current\_set = **new\_block**(int,graph->size);  
 int set\_size = 0;  
 **bool**\* visited = (**bool**\*)calloc(graph->size,1);  
 int v\_index = 0;  
  
 while (graph->size>0){  
 set\_size = 0;  
 while (v\_index!=-1){  
 visited[v\_index]=1;  
 current\_set[set\_size] = graph->vertexes[v\_index];  
  
 for (int i = 0; i < graph->size; ++i) {  
 if ((graph->matrix[v\_index][i] **or** graph->matrix[i][v\_index]) **and** !visited[i]){  
 visited[i]=1;  
 }  
 }  
 set\_size++;  
 v\_index = fst\_false(visited,graph->size);  
 }  
 memset(visited,0,graph->size);  
 for (int i = 0; i < set\_size; i++) {  
 printf("%d,",current\_set[i]);  
 remove\_vertex(graph,get\_int\_index\_binary(graph->vertexes,current\_set[i],graph->size));  
 }  
 printf("\n");  
  
 v\_index = 0;  
 }  
  
 free(visited);  
 free(current\_set);  
}

При реализации алгоритма использовался массив current\_set типа “int”, массив visited типа “bool”, функция fst\_false, а также структура matrix\_graph.

Функция int fst\_false возвращает индекс первого ложного значения массива bool, если такого нет, возвращает -1)

int fst\_false(**bool**\* arr,int size){  
 for (int i = 0; i < size; ++i) {  
 if(!arr[i]){  
 return i;  
 }  
 }  
 return -1;  
}

Удаление вершины производи функция remove\_vertexes()

void remove\_vertex(matrix\_graph\* graph, int vrtxindex){  
 int data\_size;  
  
 void\* dst;  
 void\* src;  
  
 *//удаляем строку матрицы* for (int i=0; i<graph->size;i++){  
 for (int j = vrtxindex; j < graph->size-1; j++) {  
 graph->matrix[j][i] = graph->matrix[j+1][i];  
 }  
 }  
  
 *//удаляем столбец матрицы* for (int i=0; i<graph->size;i++){  
 dst = &(graph->matrix[i][vrtxindex]);  
 src = &(graph->matrix[i][vrtxindex+1]);  
  
 data\_size=(graph->size)-vrtxindex;  
  
 memmove(dst,src,data\_size\*sizeof(int));  
 }  
  
 *//удаляем название вершины* dst = &(graph->vertexes[vrtxindex]);  
 src = &(graph->vertexes[vrtxindex+1]);  
  
 memmove(dst,src,(graph->size-vrtxindex)\*sizeof(int));  
  
 graph->size--;  
}

Для удаления вершины графа, нам нужен индекс вершины. Т.к. в массив current\_set сохраняются названия вершин (индексы изначальной матрицы), необходимо найти, какому индексу она соответствует в новой матрице. Т.к. массив всегда будет отсортирован, воспользуемся двоичным поиском.

int get\_int\_index\_binary(int\* arr, size\_t size, int key){  
 size\_t left = 0;  
 size\_t right = size-1;  
 while (left<=right){  
 size\_t mid = (left-right)/2;  
 if (key==arr[mid]){  
 return mid;  
 }  
 if (key< arr[mid]){  
 right = mid - 1;  
 } else{  
 left = mid +1;  
 }  
 }  
 return -1;  
}

Далее происходит высвобождение выделенной памяти, и завершение программы.

Руководство пользователя

При заходе на сайт пользователь видит пустой холст, главное меню, и баннер с вспомогательной информацией:



Рисунок 5Старт программы

Для того, чтобы раскрасить граф, его нужно задать. Предоставляется три способа задания графа:

1. Вручную
2. С помощью матрицы смежности
3. С помощью случайной генерации

Вручную:

Главное меню – Добавить вершину (либо нажать кнопку v на клавиатуре), кликнуть в область экрана. Соединить вершину – Неориентированное/ориентированное ребро, кликнуть на первую и вторую вершину:

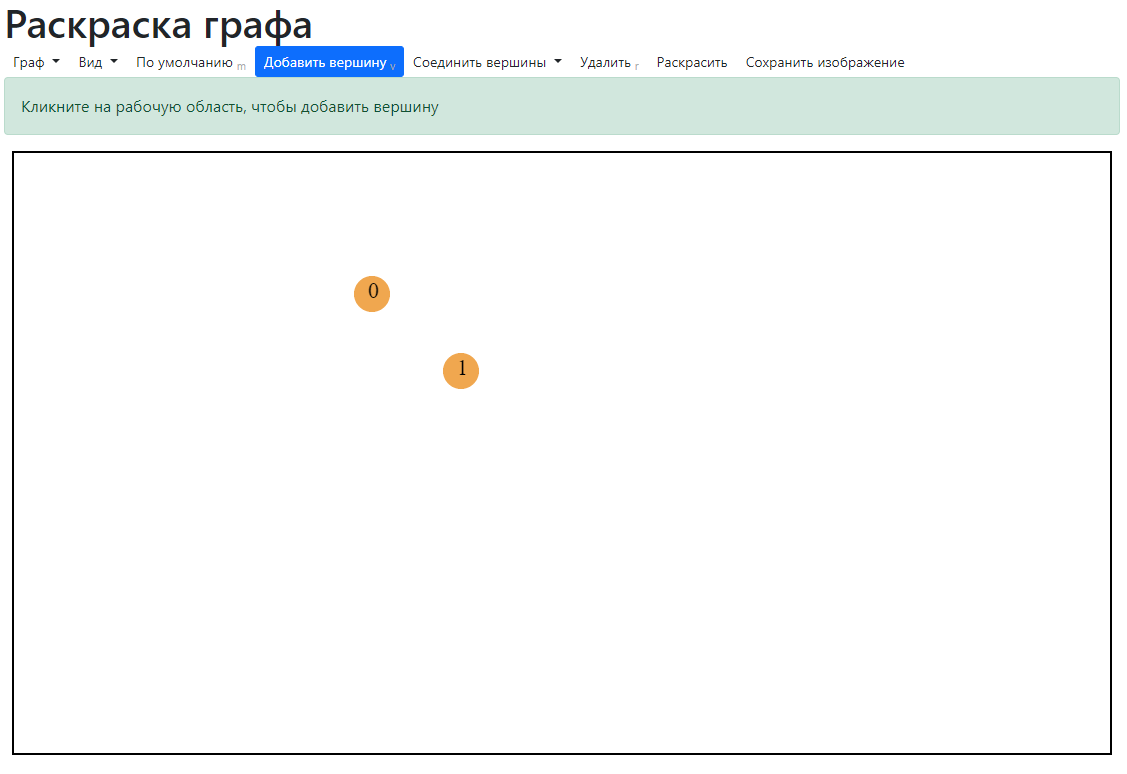


Рисунок 6 Добавление вершин

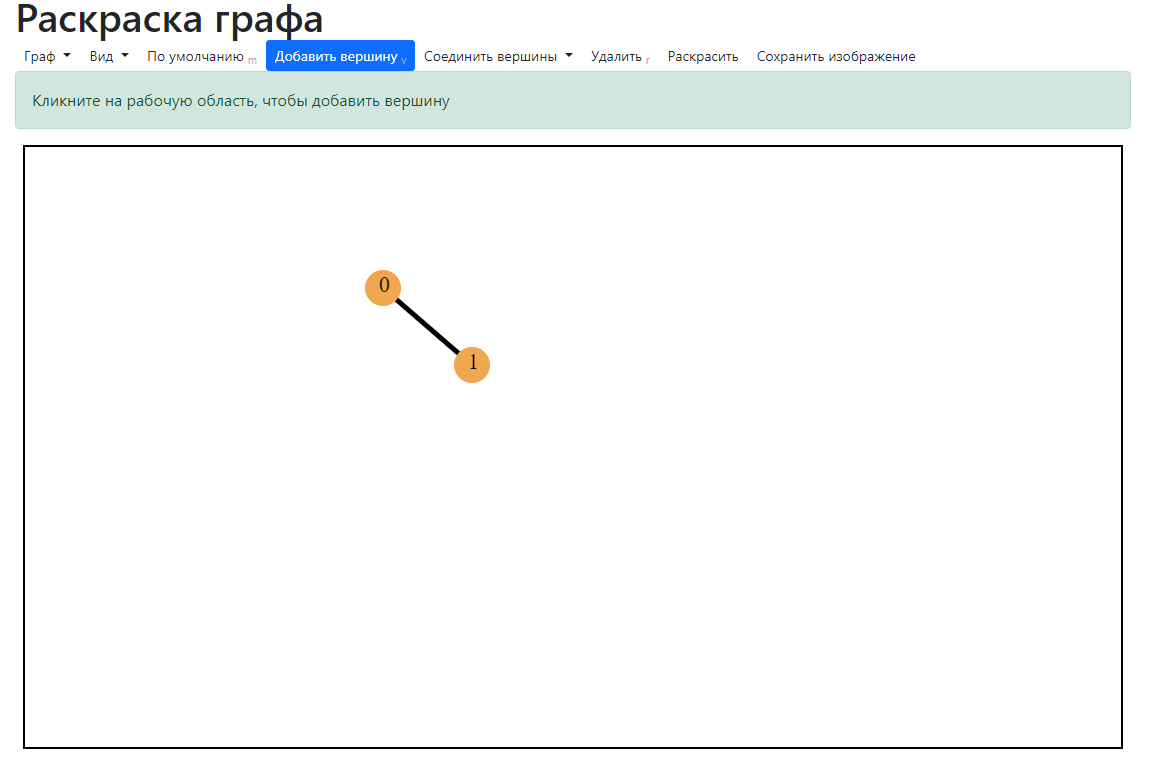


Рисунок 7Добавление неориентированного ребра

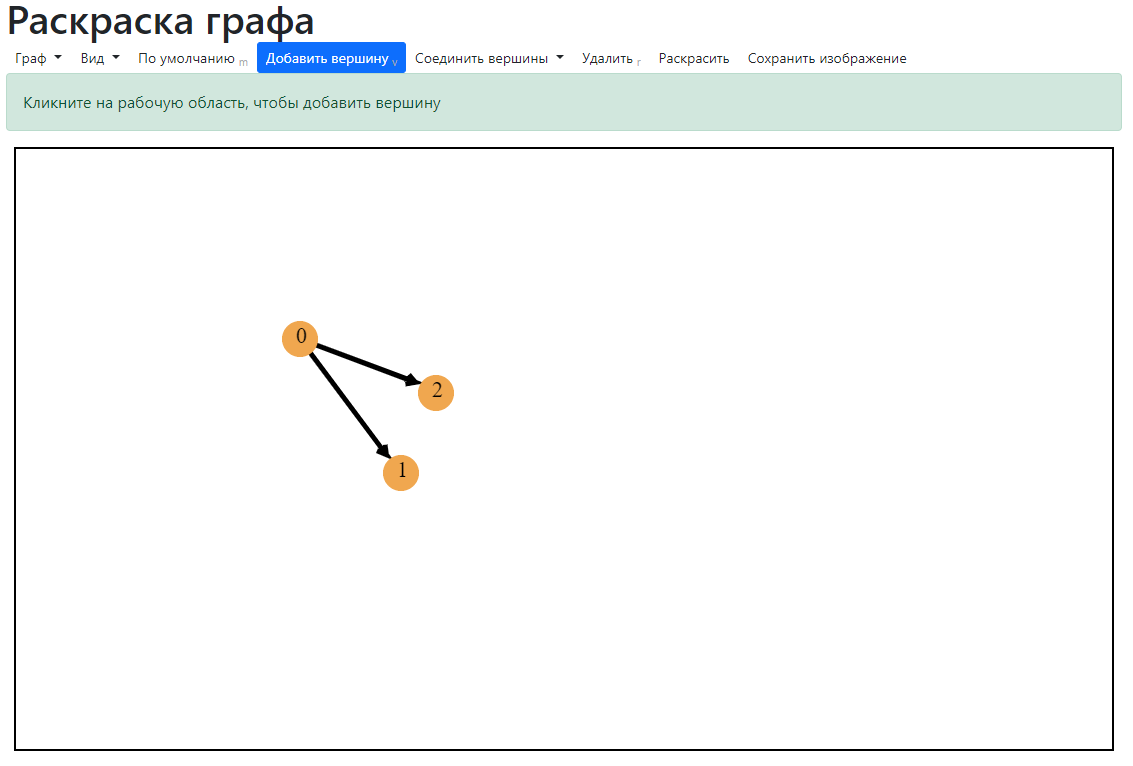


Рисунок 8 Добавление ориентированных ребер

С помощью матрицы смежности (Граф – Матрица смежности – Создать):

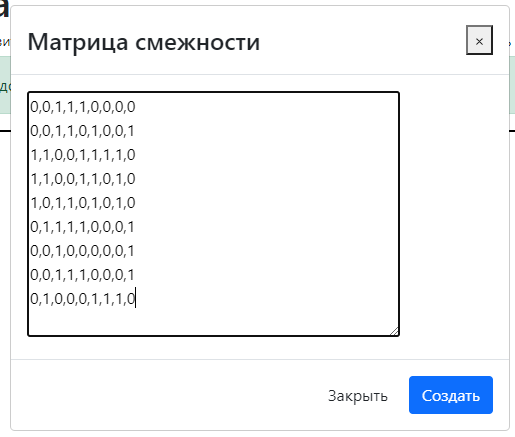


Рисунок 9 Матрицы смежности графа

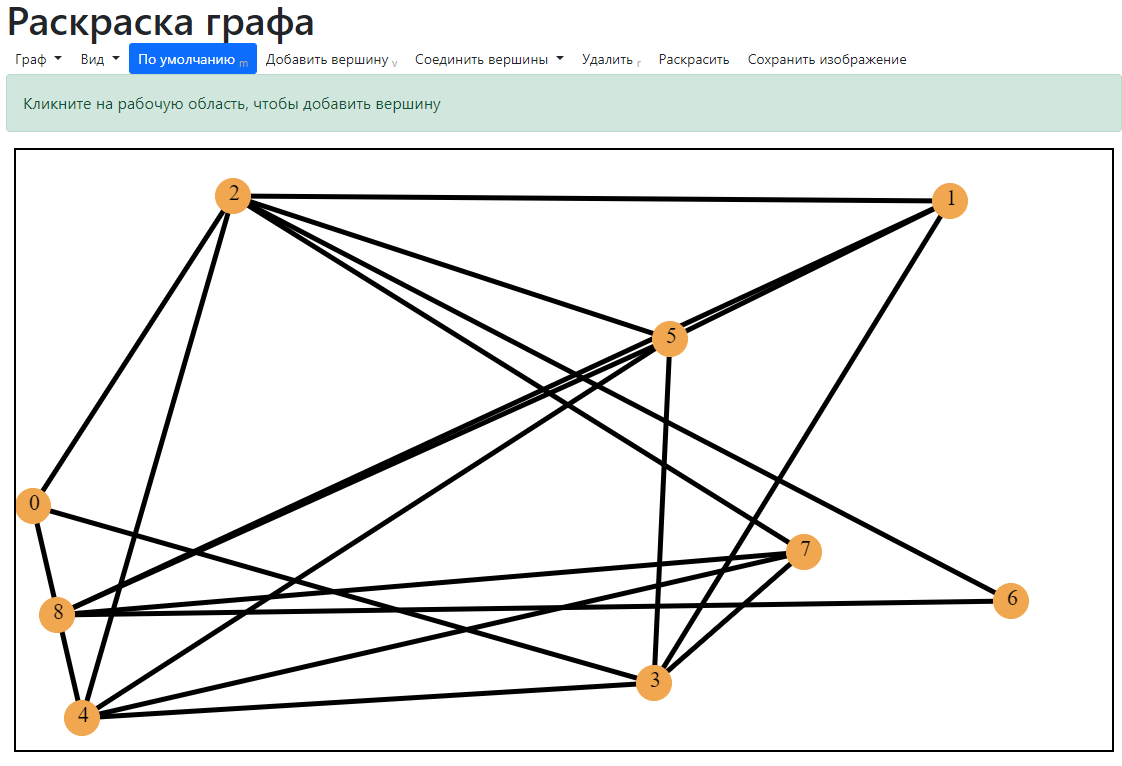


Рисунок 10 Результат создания графа по матрице смежности

Случайная генерация (Граф – Создать рандомный граф – Создать ):

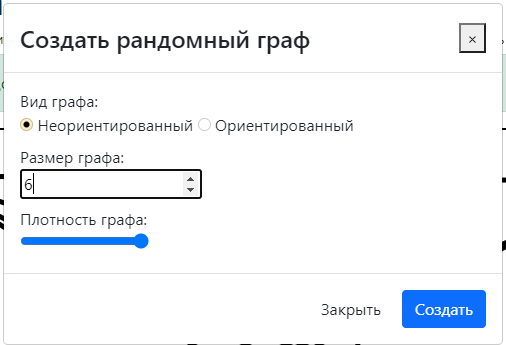


Рисунок 11 Случайная генерация ориентированного графа

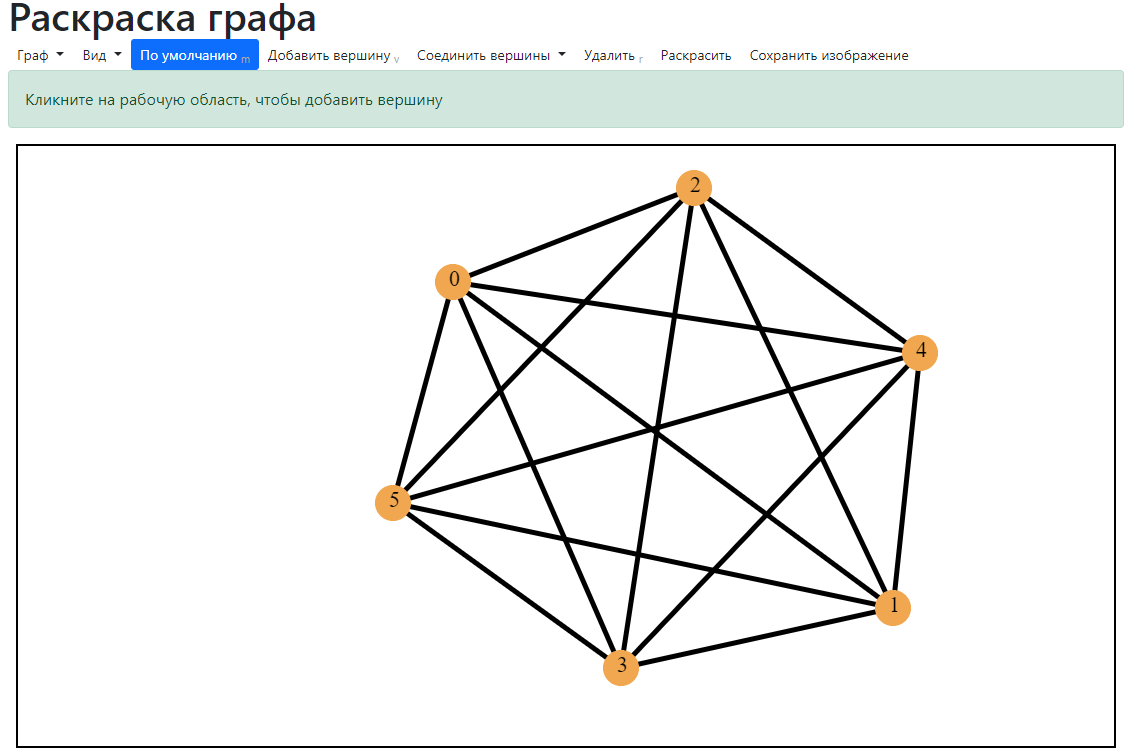


Рисунок 12 Результат случайной генерации ориентированного графа (вершины были перемещены для наглядности)

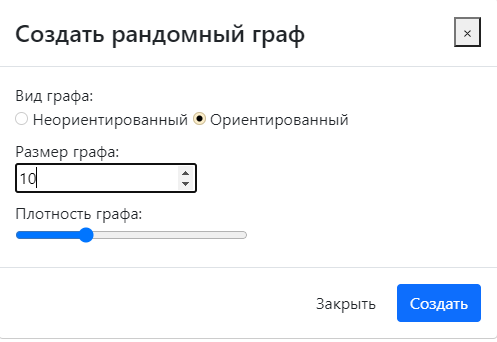


Рисунок 13 Случайная генерация ориентированного графа

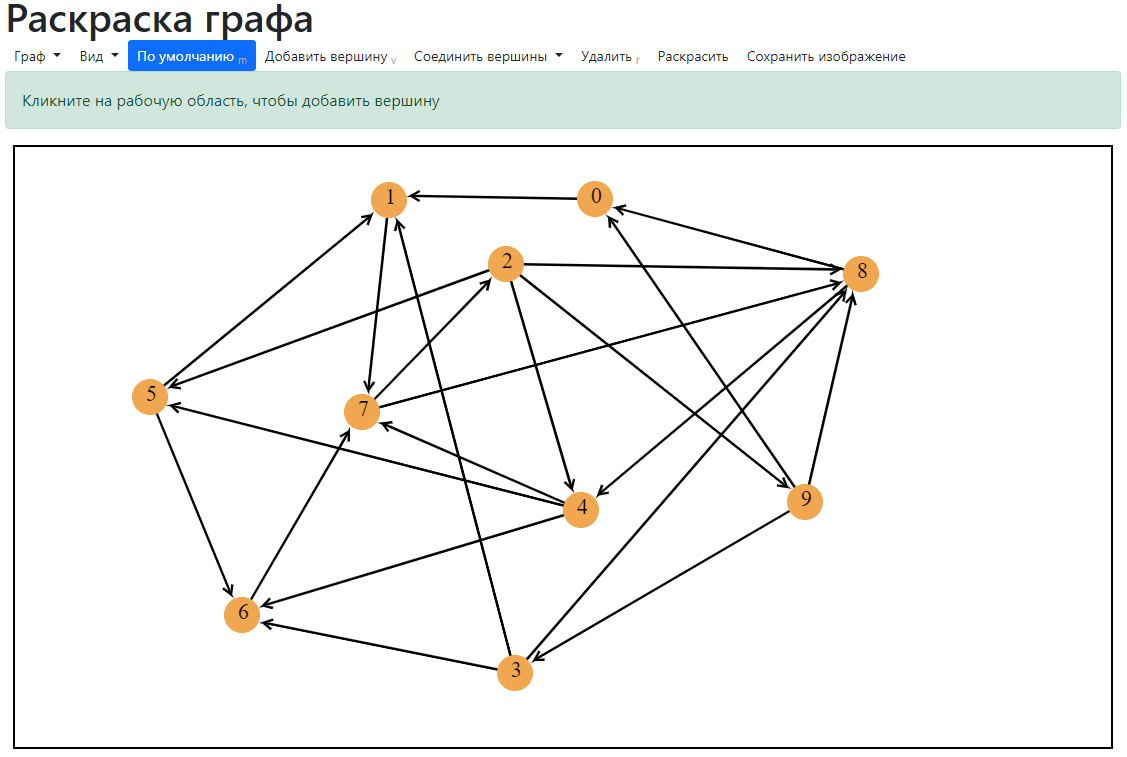


Рисунок 14 Результат генерации случайного ориентированного графа

После того, как граф был задан, пользователь может расставить вершины с помощью мыши в нужном порядке. Для этого в главном меню нужно выбрать режим По умолчанию или нажать кнопку m на клавиатуре.

Можно удалить ненужные вершины (Главное меню – Удалить, кликнуть на вершину или нажать кнопку r на клавиатуре):

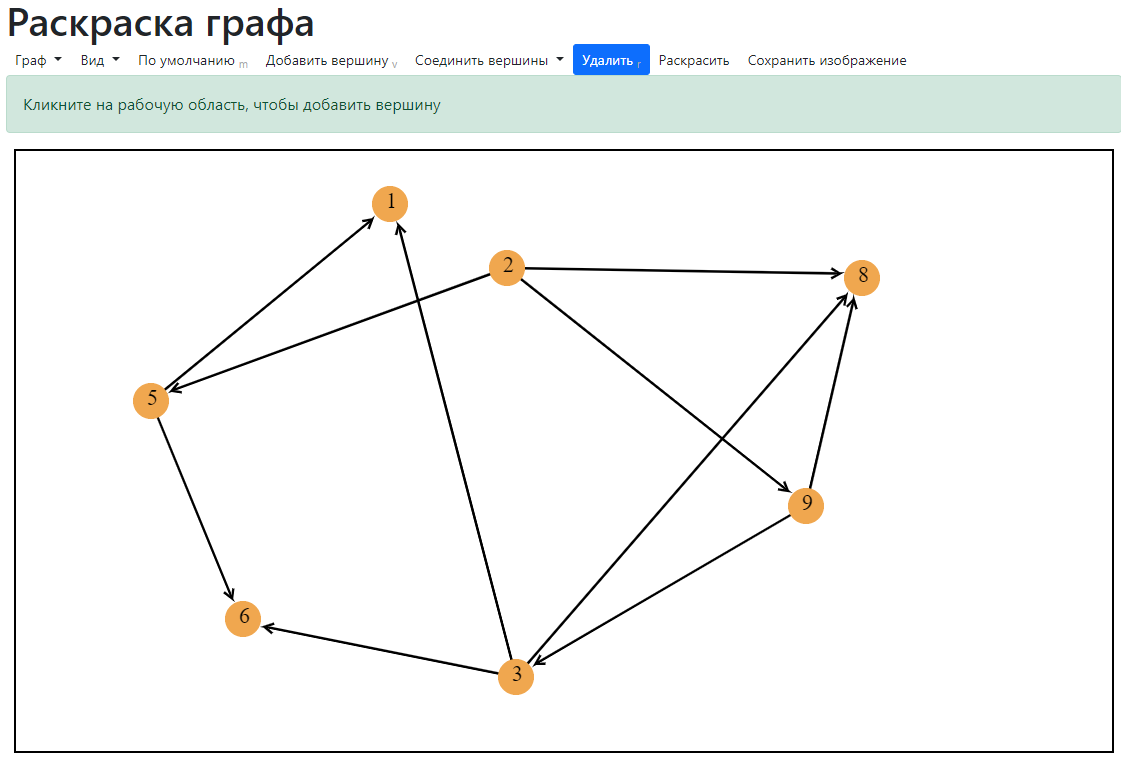


Рисунок 15 Удаление вершин 0,4,7 (см рис. 14)

Также предоставляется возможность регулировать отображение элементов графа - менять радиус моделей вершин, и толщину ребер и выделять ребра выбранной вершины:

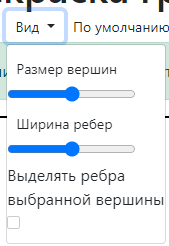


Рисунок 16 Настройки вида графа

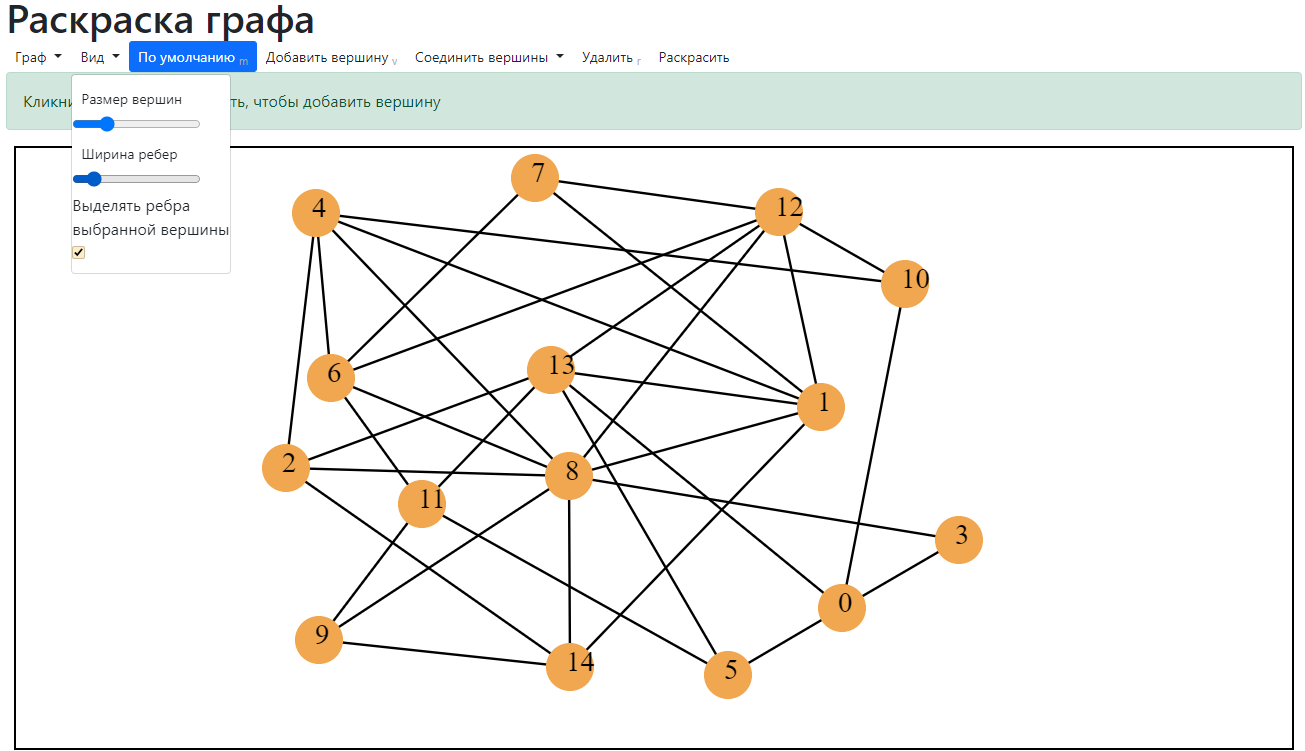


Рисунок 17 Изменён вид графа

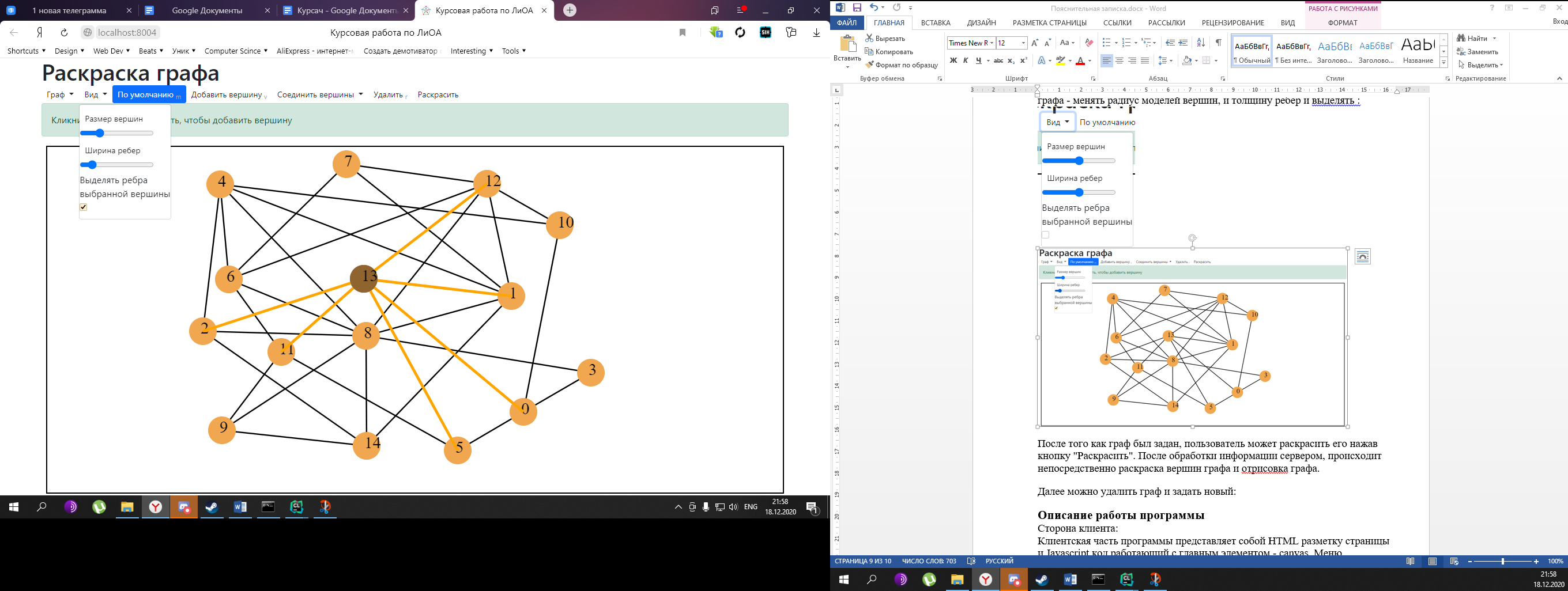


Рисунок 18Функция выделения ребер выбранной вершины

После того как граф был задан, пользователь может раскрасить (Главное меню – Раскрасить ). После обработки информации сервером, происходит непосредственно раскраска вершин графа и вывод результат на холст:

Можно удалить граф и задать новый:

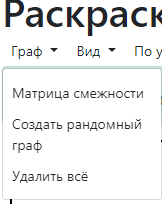


Рисунок 19 Функция Удалить всё

Чтобы сохранить результат работы программы нужно нажать кнопку Сохранить изображение.

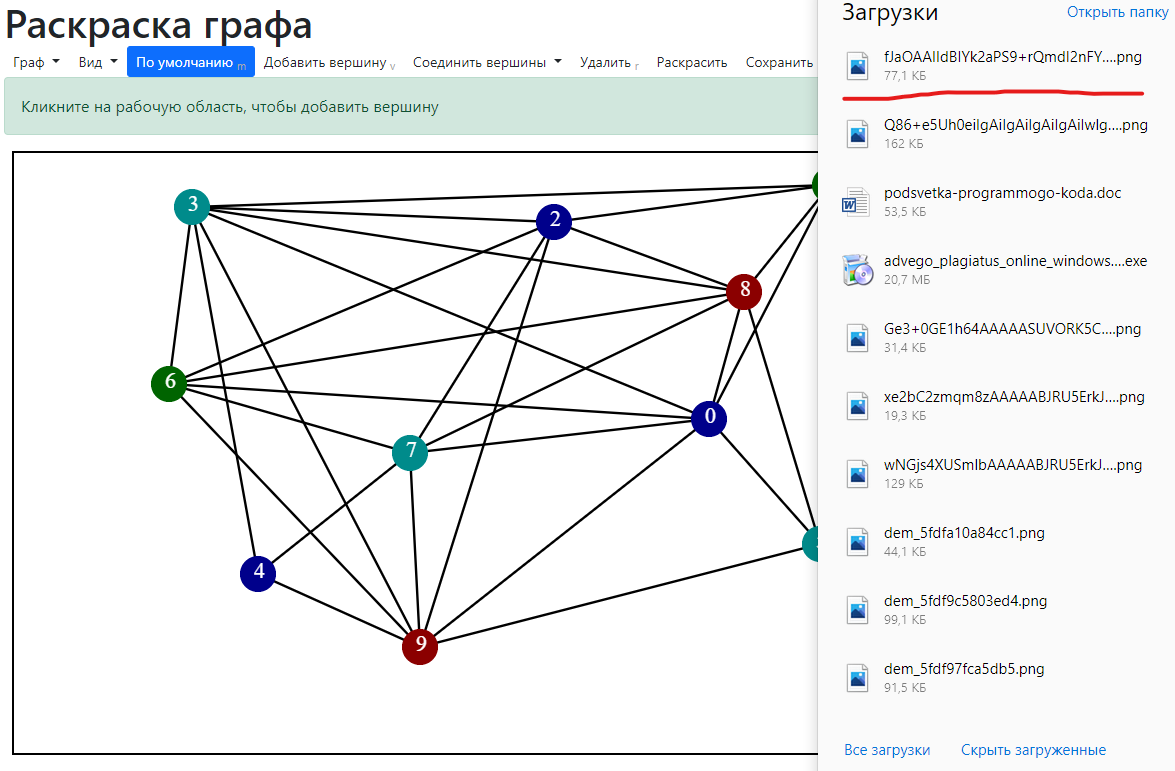


Рисунок 20 Сохранение результата работы работы программы

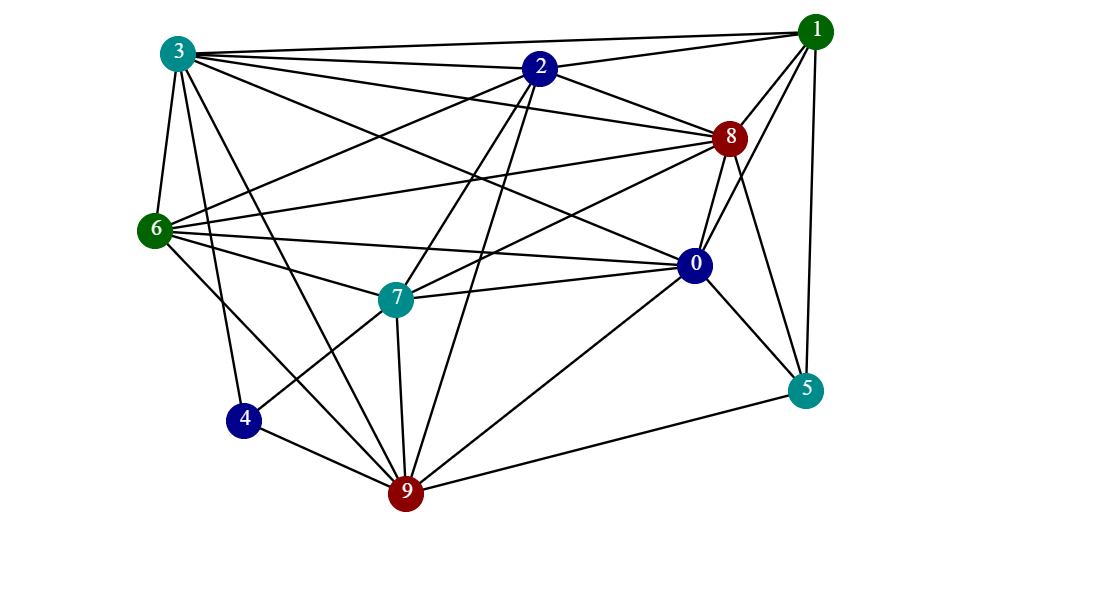


Рисунок 21 Сохраненное изображение

Тестирование

## **Ручной просчет алгоритма**

Для примера возьмем следующий граф:

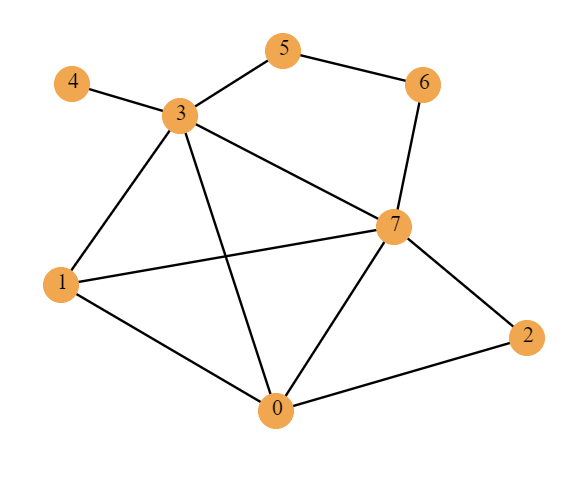


Рисунок 22 Исходный граф для тестирования

Примем за один шаг – нахождение одного независимого множества.

*Примечание: после окончания поиска одного независимого множества массив current\_set не обнуляется. Вершины нового независимого множества, перезаписывают “старые”, и обнуляется переменная set\_size, поэтому на работоспособность программы это не влияет. Для наглядности ненужные элементы будут скрыты при тестировании.*

Шаг 1.

Начинаем с вершины 0

graph->size = 8;

v\_index = 0;

visited: 0

current\_set: 0

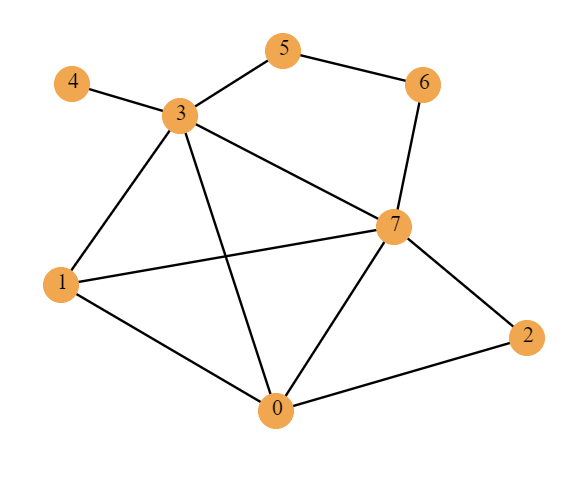


Рисунок 23 Ручной просчет графа. Шаг 1.1

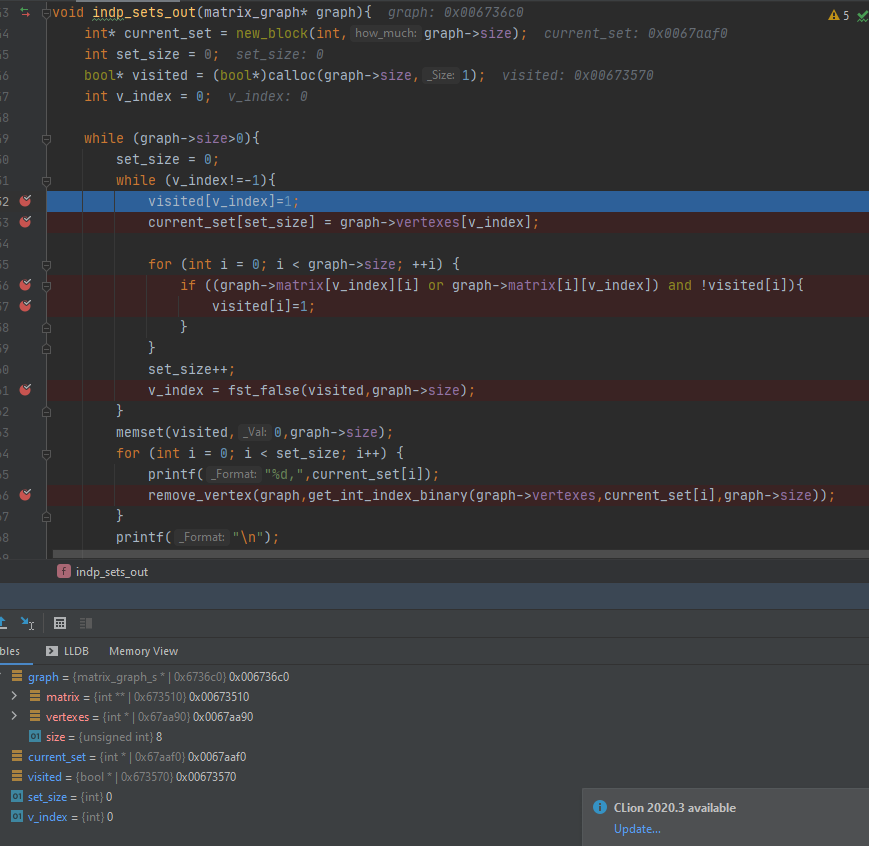


Рисунок 24 Ручной просчет графа. Шаг 1.1 Отладка

Посещенные вершины:

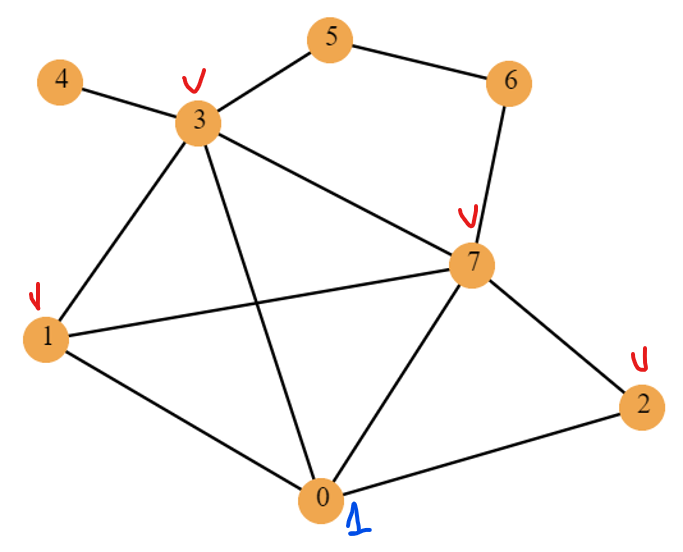


Рисунок 25Ручной просчет графа. Шаг 1.2

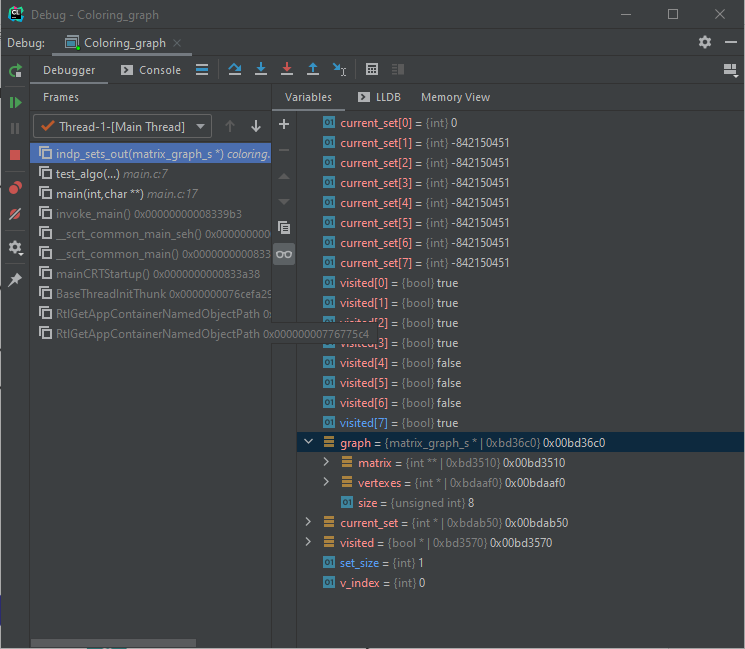


Рисунок 26 Ручной просчет графа. Шаг 1.2 Отладка

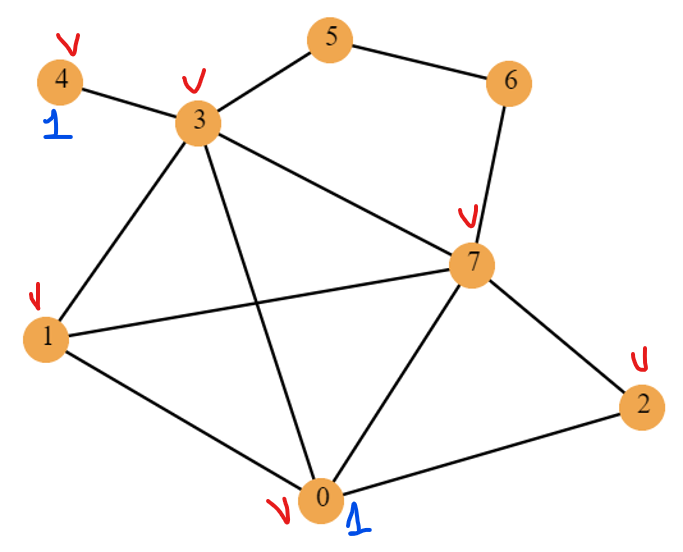


Рисунок 27Ручной просчет графа. Шаг 1.3

graph->size = 8;

v\_index = **4**;

visited: 0,**1,2,3,7,4**

current\_set: 0, **4**

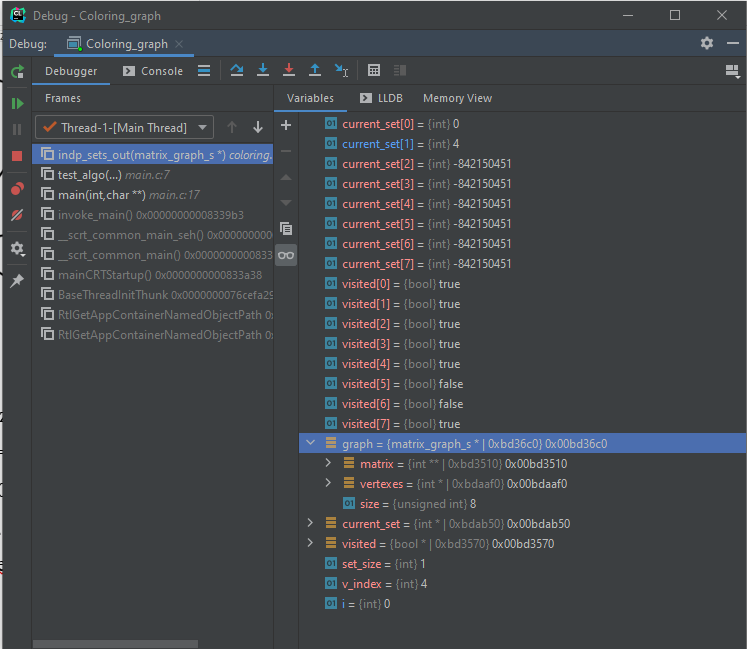


Рисунок 28 Ручной просчет графа. Шаг 1.3 Отладка

graph->size = 8;

v\_index = **5**;

visited: 0,1,2,3,7,4,**5**

current\_set: 0, 4, **5**

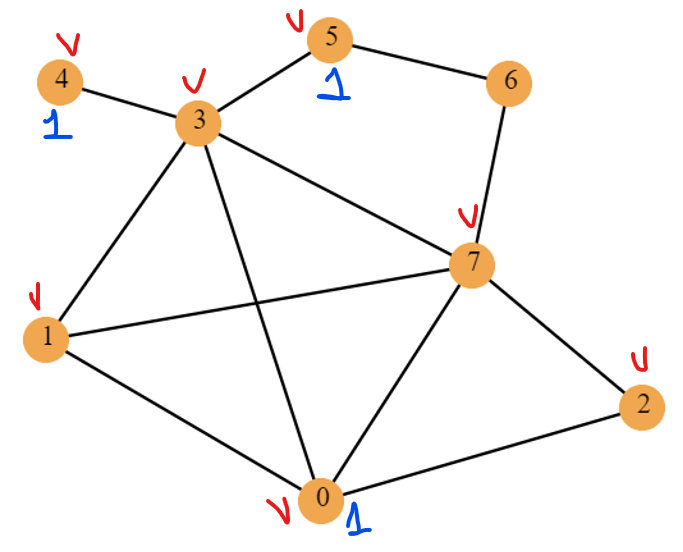


Рисунок 29 Ручной просчет графа. Шаг 1.4

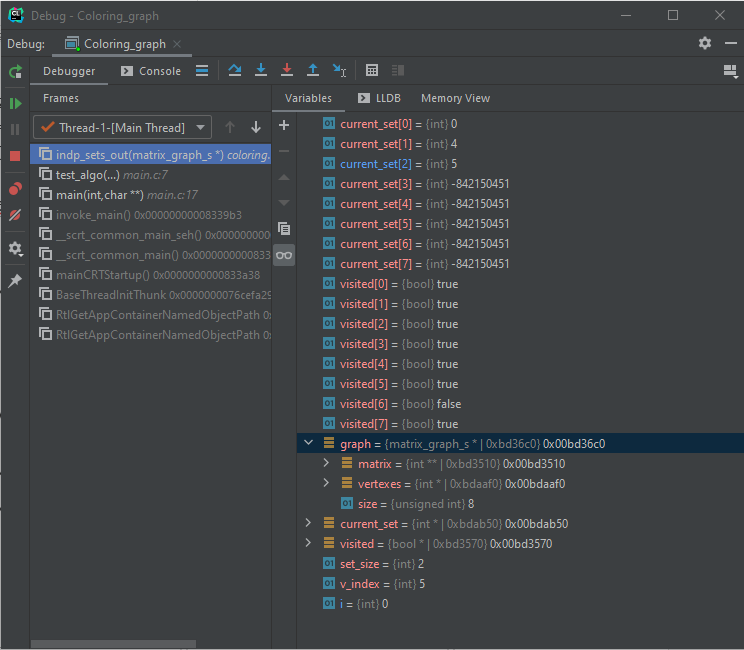


Рисунок 30 Ручной просчет графа. Шаг 1.4 Отладка

graph->size = 8;

v\_index = 5;

visited: 0,1,2,3,7,4,5, **6**

current\_set: 0, 4, 5

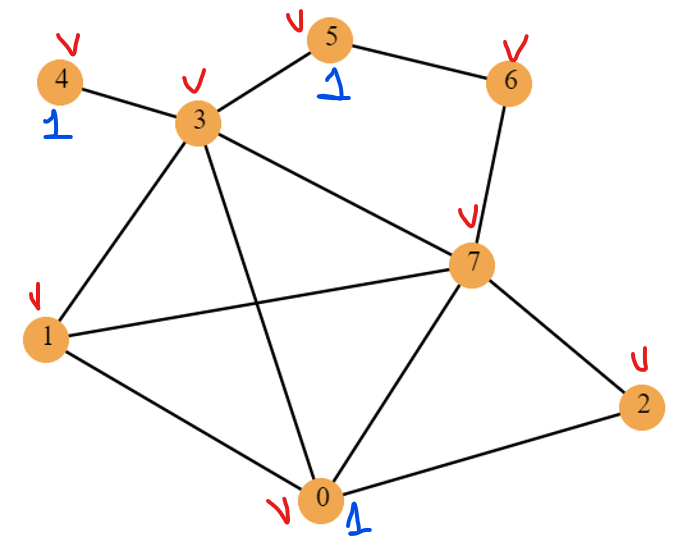


Рисунок 31 Ручной просчет графа. Шаг 1.5

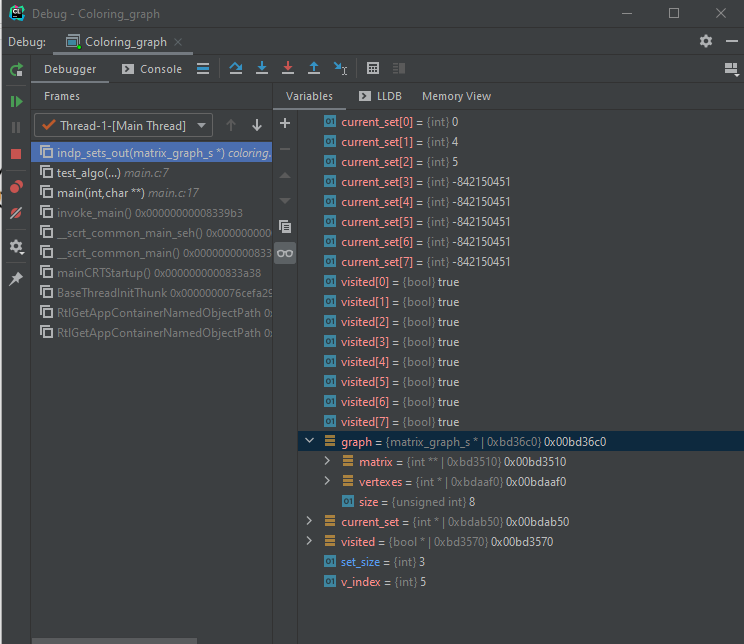


Рисунок 32 Ручной просчет графа. Шаг 1.5 Отладка

graph->size = 8;

v\_index = **-1**;

visited: 0,1,2,3,7,4,5, 6

current\_set: 0, 4, 5

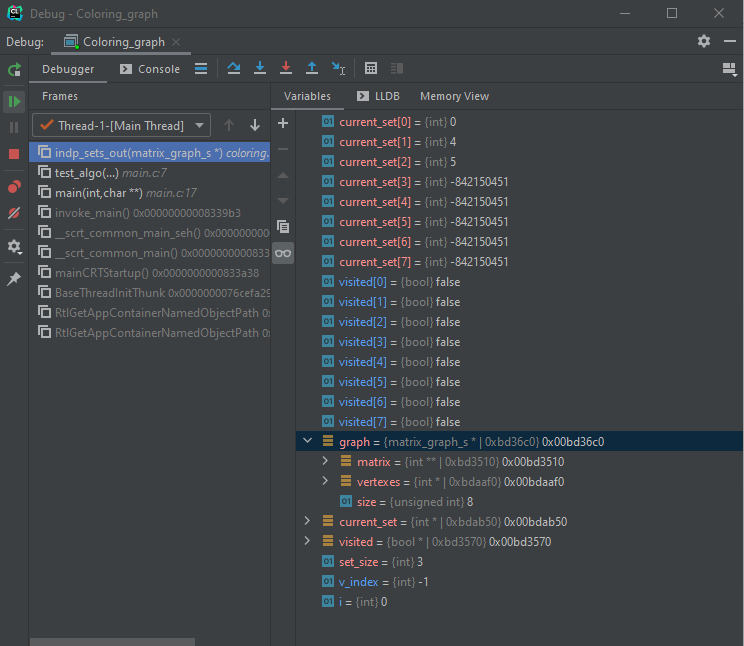


Рисунок 33Ручной просчёт графа. Шаг 1.5 Отладка

Шаг 2.

graph->size = 5;

v\_index = 0; (вершина 1)

visited:

current\_set:

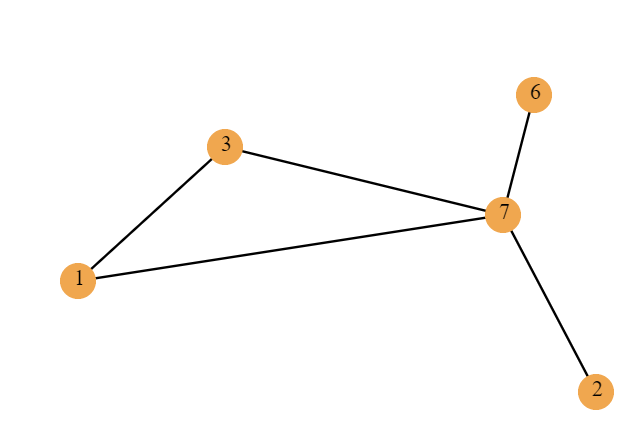


Рисунок 34 Ручной просчет графа. Шаг 2

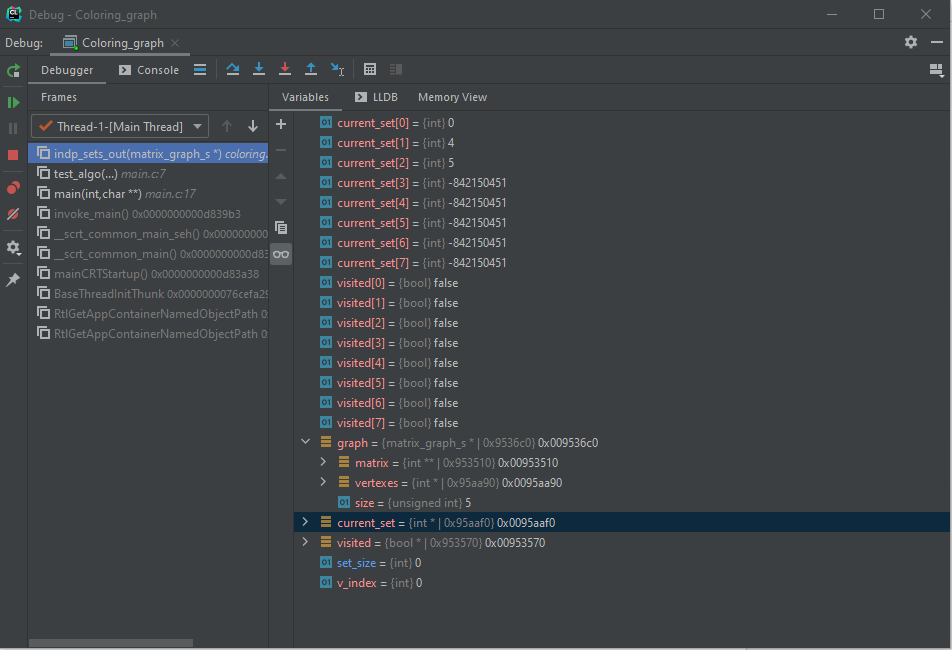


Рисунок 35 Ручной просчет графа. Шаг 2 Отладка

graph->size = 5;

v\_index = 0; (вершина 1)

visited: 0,2,4 (вершины **1,3,7)**

current\_set: **1**,

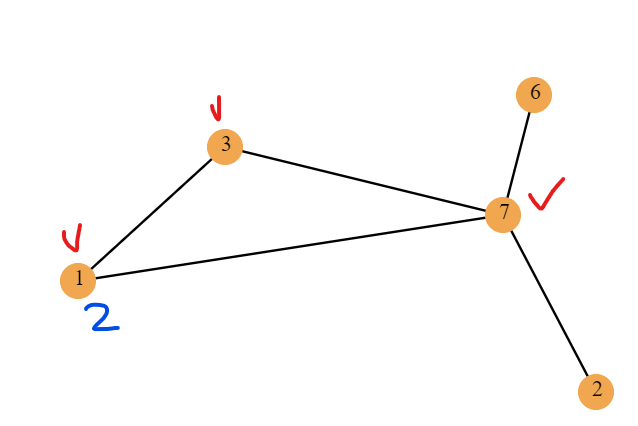


Рисунок 36 Ручной просчет графа. Шаг 2.1

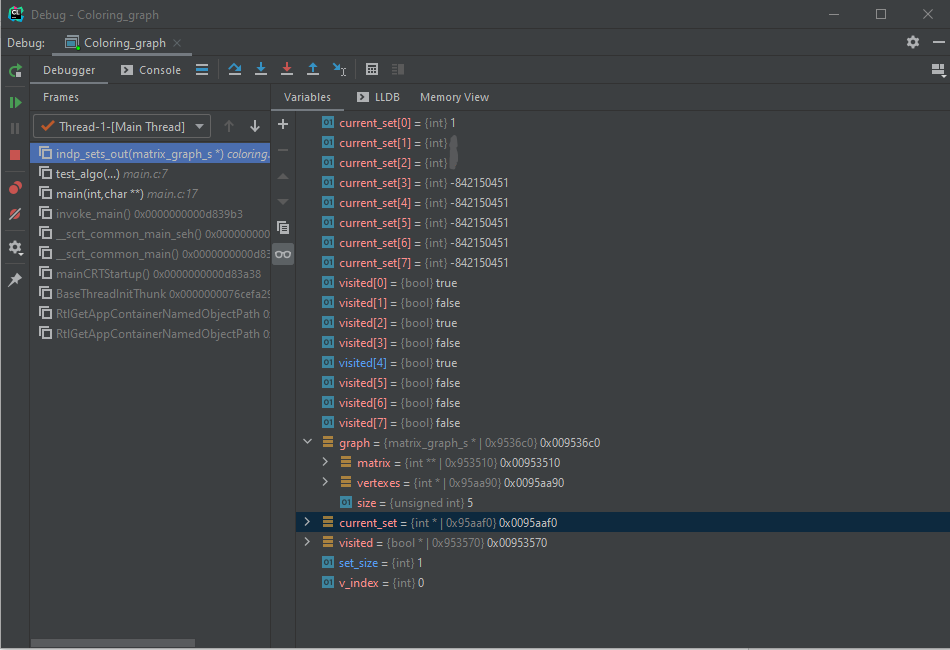


Рисунок 37 Ручной просчет графа. Шаг 2.1 Отладка

graph->size = 5;

v\_index = **1**; (вершина 2)

visited: 1,2,4,**1**(вершины 1,3,7,**2)**

current\_set: 1,**2**

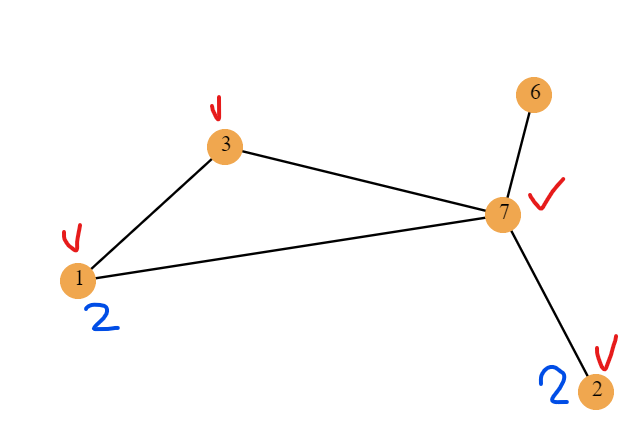


Рисунок 38 Ручной просчет графа. Шаг 2.2

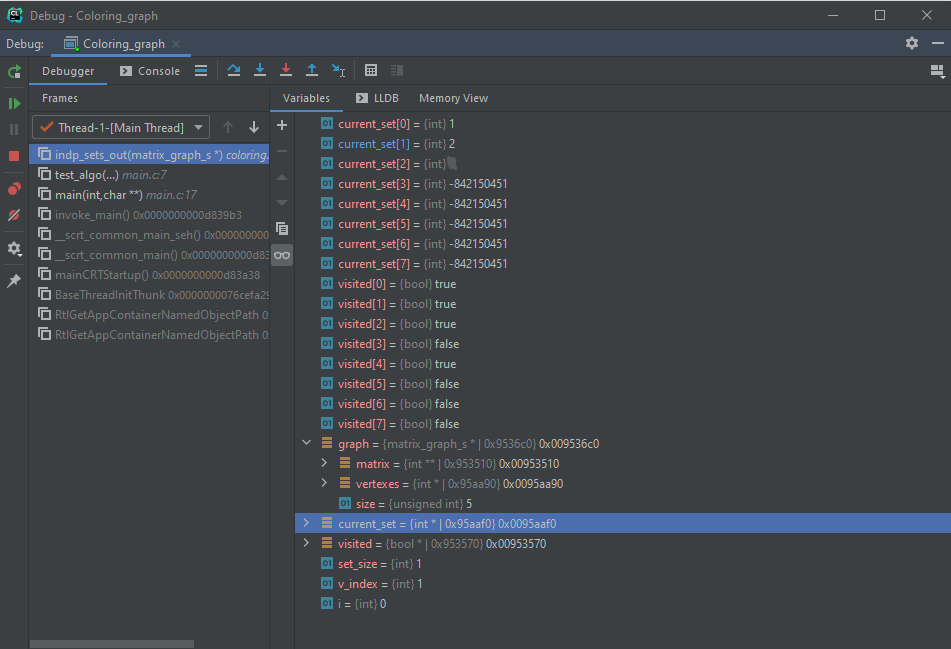


Рисунок 39 Ручной просчет графа. Шаг 2.2 Отладка

graph->size = 5;

v\_index = 3; (вершина 6)

visited: 1,2,4,1**,3**(вершины 1,3,7,2**,6)**

current\_set: 1,2,6

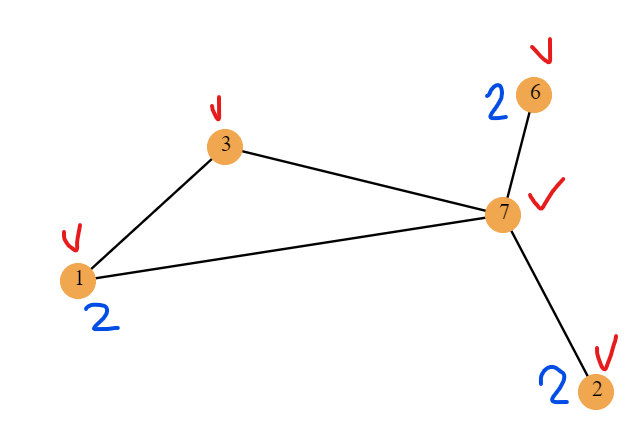


Рисунок 40 Ручной просчет графа. Шаг 2.3

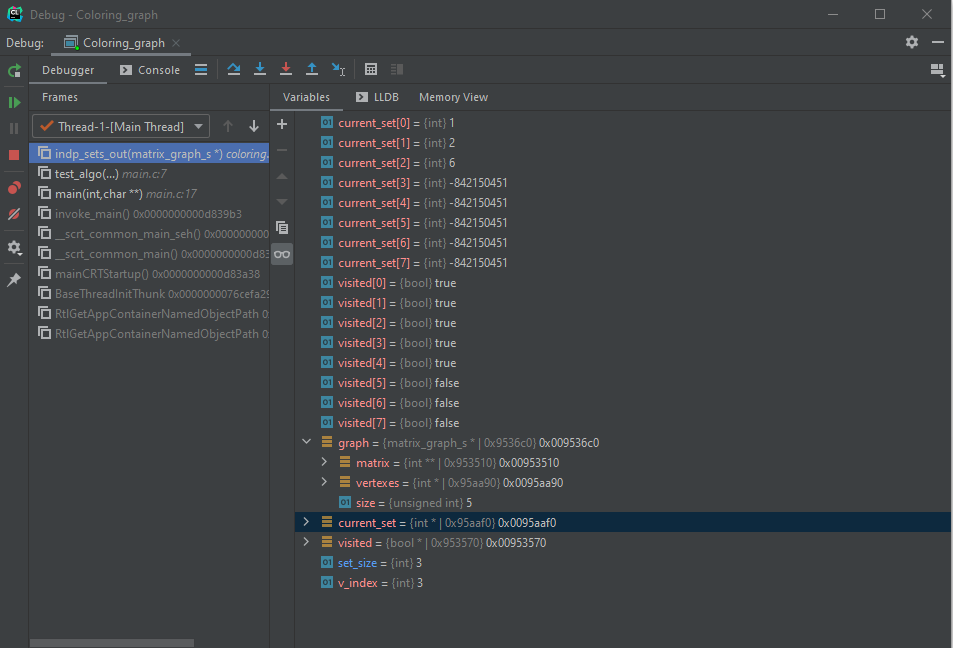


Рисунок 41Ручной просчет графа. Шаг 2.3 Отладка

graph->size = 5;

v\_index = -1; (вершина 6)

visited:

current\_set:

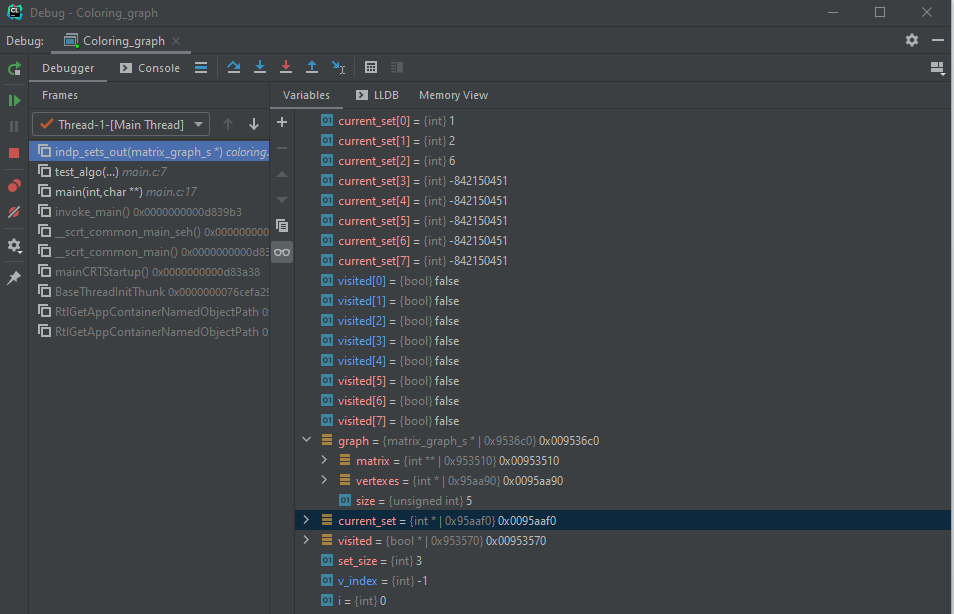


Рисунок 42Ручной просчет графа. Шаг 2.4 Отладка

Шаг 3.

graph->size = 2;

v\_index = 0; (вершина 3)

visited: 0 ; (вершина 3)

current\_set: 3

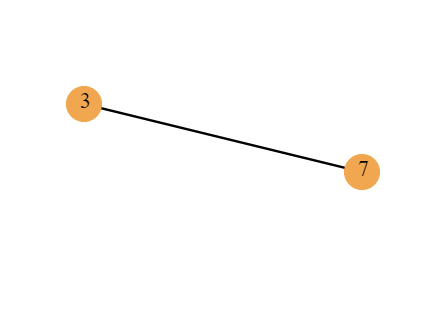


Рисунок 43 Ручной просчет графа. Шаг 3

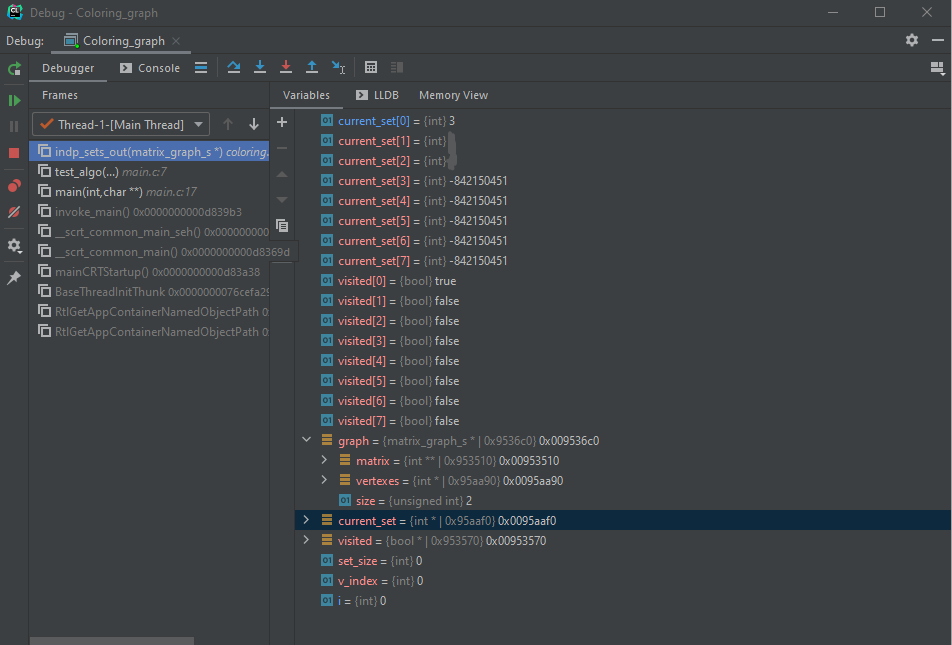


Рисунок 44 Ручной просчет графа. Шаг 3 Отладка

graph->size = 2;

v\_index = 0; (вершина 3)

visited: 0,1; (вершина 3,7)

current\_set: 3

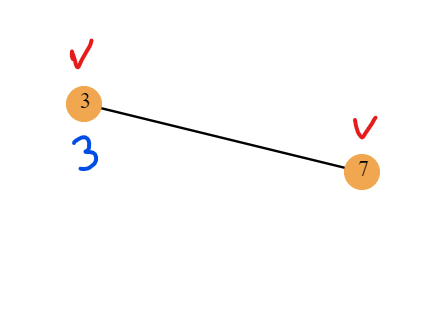


Рисунок 45Ручной просчет графа. Шаг 3.1

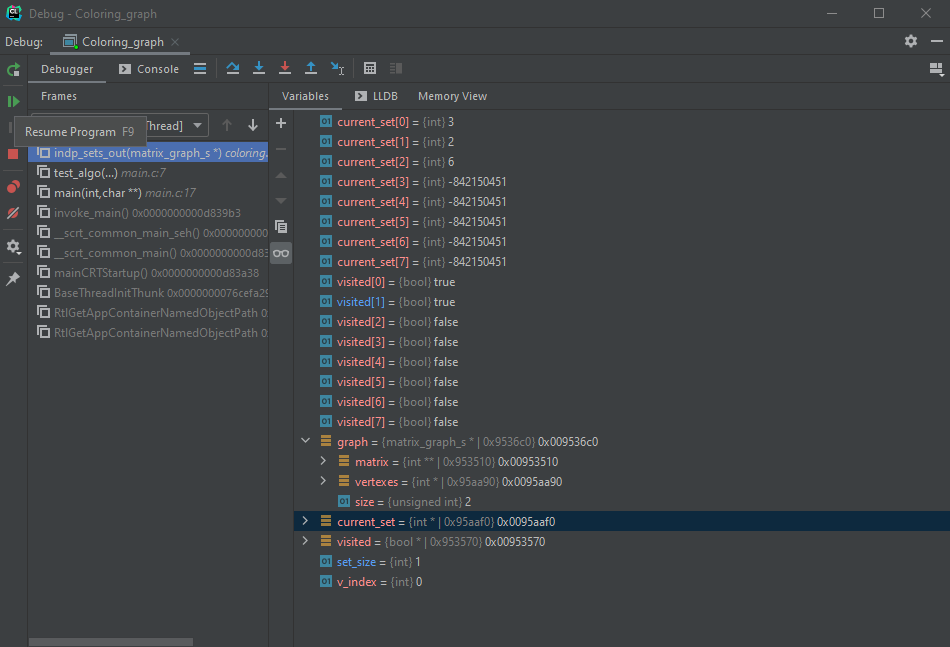


Рисунок 46 Ручной просчет графа. Шаг 3.1 Отладка

graph->size = 2;

v\_index = -1;

visited:

current\_set:

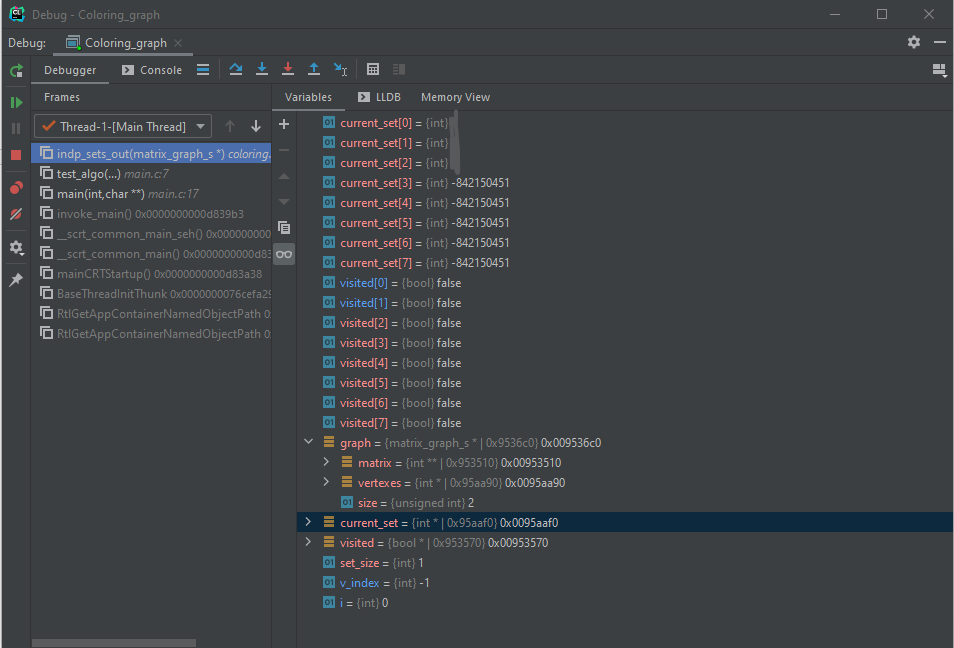


Рисунок 47 Ручной просчет графа. Шаг 3.2 Отладка

Шаг 4.

graph->size = 1;

v\_index = 0; (вершина 7)

visited: 0; (вершина 7)

current\_set: 7

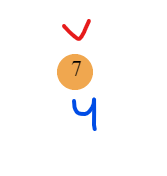


Рисунок 48 Ручной просчет графа. Шаг 4

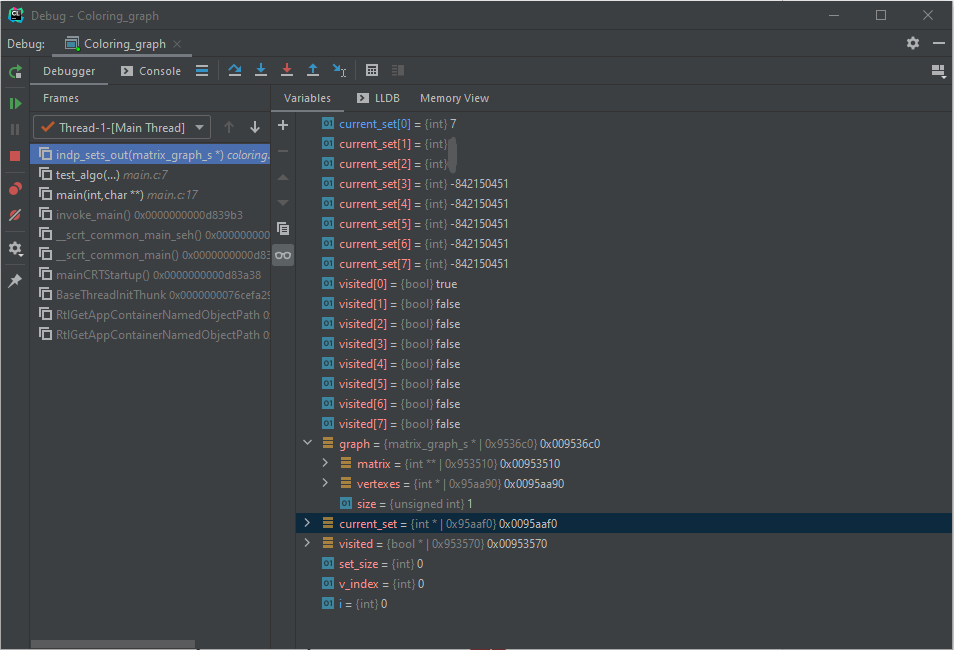


Рисунок 49Рисунок 48 Ручной просчет графа. Шаг 4 Отладка

graph->size = 0;

v\_index = **-1**;

visited:

current\_set:

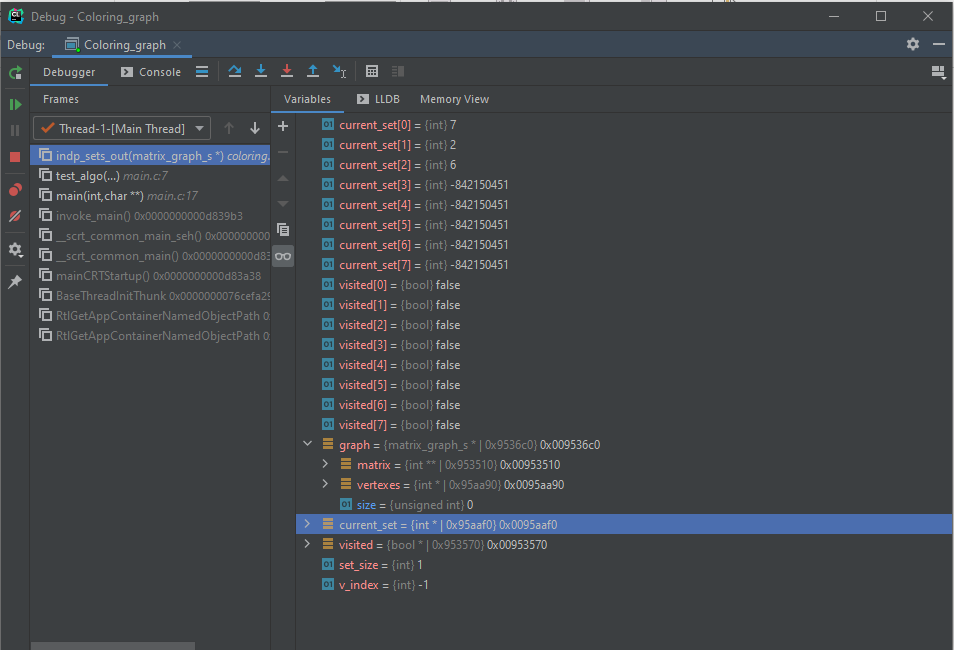


Рисунок 50 Ручной просчет графа. Шаг 4.1 Отладка

Результат ручного расчета:

Найдено независимых множеств: 4:

1. 0,4,5
2. 1,2,6
3. 3
4. 7

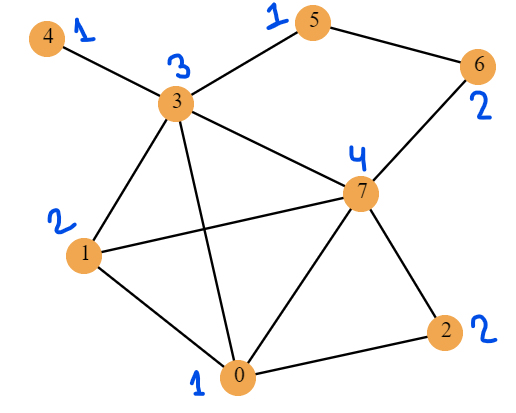


Рисунок 51 Результат ручного просчета

Результат работы программы:

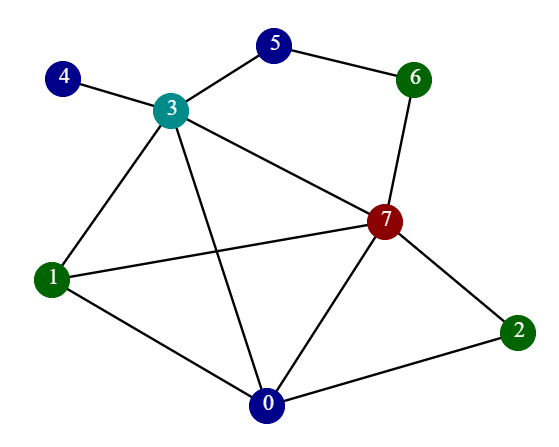


Рисунок 52 Результат работы программы

## **Проверка на типовые ошибки**

Раскраска незаданного графа:

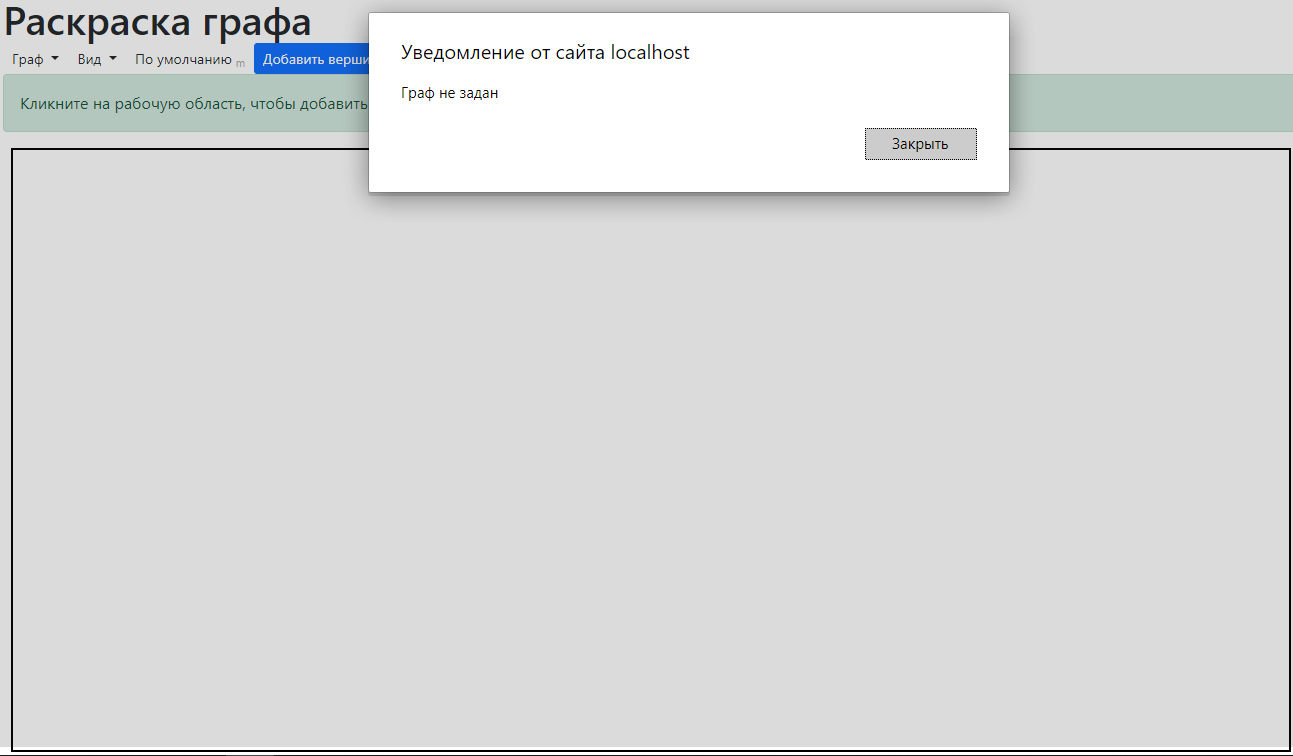


Рисунок 53 Попытка раскрасить нулевой граф

Ограничение на генерацию графа

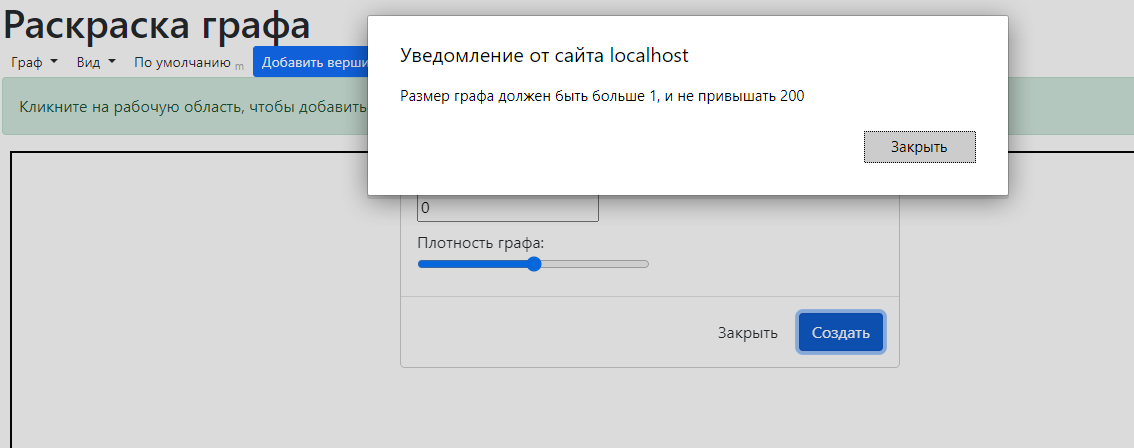


Рисунок 54 Попытка задать граф вне диапазона допустимого размера

Обработка ввода матрицы смежности

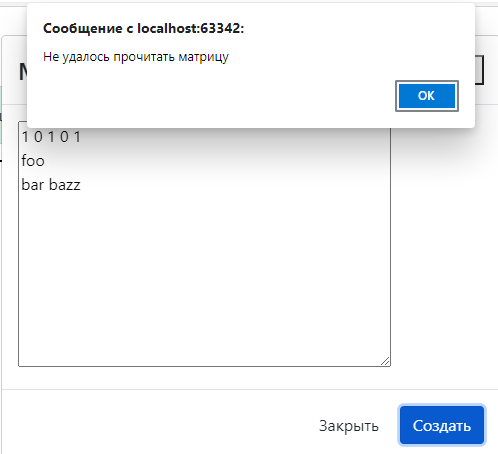


Рисунок 55 Обработка ввода матрицы смежности

Заключение

В процессе создании этого проекта была разработана программа, реализующая алгоритма раскрашивания графа.

При выполнении данной курсовой работы были получены навыки разработки программ, и освоены приемы программирования с использованием теории графов. Были отточены навыки программирования на языке Си и веб разработки.

Функционал программы можно расширить, добавив другие алгоритмы на графах. Возможны некоторые доработки, связанные с визуализацией графов, например, удаление ребер, масштабирование, выделение нескольких элементов графа, назначение названий вершин.

Программа имеет достаточный для использования функционал.

Список литературы

1. Narsingh Deo Graph Theory with Applications to Engineering & Computer Science. - перепечатанное изд. Prentice-Hall of India, 2004, ISBN 8120301455, 9788120301450
2. Вазирани Умеш, Дасгупта Санджой, Пападимитриу Христос Graph Theory with Applications to Engineering & Computer Science. - перепечатанное изд. МЦНМО, 2019, ISBN 978-5-44392-893-7
3. David Flanagan JavaScript: The Definitive Guide. - 7-e изд. O'Reilly Media, Inc., 2020, ISBN: 9781491952023
4. John Wiley & Sons - 1-e изд. 2011, ISBN-10 : 1118008189 ISBN-13 : 978-1118008188.
5. Resources for developers, by developers. // developer.mozilla.org URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas\_API/Tutorial
6. http://www.4stud.info/web-programming/cgi.html

**Приложение А**

**Листинг**

**main.c**

#include "coloring.h"  
  
void test\_algo(){  
 FILE\* test\_matrix = fopen(**HOME\_DIR**"test.txt","r");  
 matrix\_graph\* graph = init\_graph\_from\_stream(test\_matrix);  
 row(graph->vertexes,graph->size,0,i)  
 indp\_sets\_out(graph);  
 matrixGraph\_free(graph);  
}  
  
int main(int argc,char\*\* argv) {  
 matrix\_graph\* graph = init\_graph\_from\_cgi\_client();  
 printf("Content-type: text/plain\n\n");  
 indp\_sets\_out(graph);  
 matrixGraph\_free(graph);  
 return 0;  
}

**coloring.h**

#ifndef **GRAPHMATRIX\_COLORING\_H**#define **GRAPHMATRIX\_COLORING\_H**#endif *//GRAPHMATRIX\_COLORING\_H*#ifndef **GRAPHMATRIX\_LISTSGRAPH\_H**#include "matrixgraph.h"  
#endif  
  
void indp\_sets\_out(matrix\_graph\* graph);

**coloring.c**

#include "coloring.h"  
  
void indp\_sets\_out(matrix\_graph\* graph){  
 int\* current\_set = **new\_block**(int,graph->size);  
 int set\_size = 0;  
 **bool**\* visited = (**bool**\*)calloc(graph->size,1);  
 int v\_index = 0;  
  
 while (graph->size>0){  
 set\_size = 0;  
 while (v\_index!=-1){  
 visited[v\_index]=1;  
 current\_set[set\_size] = graph->vertexes[v\_index];  
  
 for (int i = 0; i < graph->size; ++i) {  
 if ((graph->matrix[v\_index][i] **or** graph->matrix[i][v\_index]) **and** !visited[i]){  
 visited[i]=1;  
 }  
 }  
 set\_size++;  
 v\_index = fst\_false(visited,graph->size);  
 }  
 memset(visited,0,graph->size);  
 for (int i = 0; i < set\_size; i++) {  
 printf("%d,",current\_set[i]);  
 remove\_vertex(graph,get\_int\_index\_binary(graph->vertexes,graph->size,current\_set[i]));  
 }  
 printf("\n");  
  
 v\_index = 0;  
 }  
  
 free(visited);  
 free(current\_set);  
}

**matrix\_graph.h**

#ifndef **GRAPHMATRIX\_MTRXGRPH\_H**#define **GRAPHMATRIX\_MTRXGRPH\_H**#endif *//GRAPHMATRIX\_MTRXGRPH\_H*#include <stdio.h>  
#include <string.h>  
  
  
#ifndef **GRAPHMATRIX\_CGI\_H**#include "cgi.h"  
#endif  
  
#include "matrix.h"  
  
typedef struct matrix\_graph\_s{  
 int\*\* matrix;  
 int\* vertexes;  
 size\_t size;  
}matrix\_graph;  
  
matrix\_graph\* matrix\_graph\_create(size\_t size);  
  
matrix\_graph\* init\_graph\_from\_string(char\* str);  
  
matrix\_graph\* init\_graph\_from\_stream(FILE\* stream);  
  
matrix\_graph\* init\_graph\_from\_stream(FILE\* stream);  
  
matrix\_graph\* init\_graph\_from\_cgi\_client();  
  
int edge\_count(matrix\_graph\* grpah, int kind);  
  
void remove\_vertex(matrix\_graph\* graph, int vrtxindex);  
  
void remove\_vrtx\_sectors(matrix\_graph\* graph, int vrtxindex);  
  
void add\_vrtx(matrix\_graph\* graph, int new\_vrtx);  
  
void print\_graph(matrix\_graph\* graph);  
  
void matrixGraph\_free(matrix\_graph\* graph);

**matrix\_graph.c**

#include "matrixgraph.h"  
  
void print\_graph(matrix\_graph\* graph){  
 printf("|v\t|");  
  
 for (int i=0; i<graph->size; i++){  
 printf("%d\t|",graph->vertexes[i]);  
 }  
 printf("\n|");  
  
 for (int i=0; i<graph->size; i++){  
  
 printf("\t|");  
 }  
 printf("\t|\n");  
  
 for (int i=0; i<graph->size; i++){  
 printf("|%d\t",graph->vertexes[i]);  
 for (int j=0; j<graph->size; j++){  
 printf("|%d\t",graph->matrix[i][j]);  
 }  
 printf("|\n");  
 }  
 printf("\n");  
}  
  
void matrixGraph\_free(matrix\_graph\* graph){  
 matrix\_free(graph->matrix,graph->size);  
 free(graph);  
}  
  
matrix\_graph\* matrix\_graph\_create(size\_t size){  
 matrix\_graph\* graph = **new**(matrix\_graph);  
 graph->matrix = matrix\_create(size);  
 graph->vertexes = **new\_block**(int,size);  
 graph->size = size;  
 return graph;  
}  
  
matrix\_graph\* init\_graph\_from\_stream(FILE\* stream){  
 int order=0;  
 char next\_char=' ';  
  
 int position = ftell(stream);  
 do{  
 if (next\_char!=' '){  
 order++;  
 }  
 next\_char = fgetc(stream);  
 }while (next\_char!='\n');  
 fseek(stream, position, **SEEK\_SET**);  
  
 matrix\_graph\* graph = matrix\_graph\_create(order);  
  
 int check;  
  
 for (int i = 0; i < order; ++i) {  
 for (int j = 0; j < order; ++j) {  
 fscanf(stream,"%d",&graph->matrix[i][j]);  
 }  
 }  
  
 return graph;  
  
}  
  
matrix\_graph\* init\_graph\_from\_string(char\* str){  
 int order=0;  
 char next\_char=' ';  
  
 int i=0;  
 do{  
 if (next\_char!=' '){  
 order++;  
 }  
 next\_char = str[i];  
 i++;  
 }while (next\_char!='\n');  
  
 matrix\_graph\* graph = matrix\_graph\_create(order);  
  
 int str\_len = strchr(str,'\n') - str;  
 for (int i = 0; i < order; ++i) {  
 for (int j = 0; j < order; ++j){  
 next\_char = str[2\*j+i\*(str\_len+1)];  
 graph->matrix[i][j] = (int)(next\_char-'0');  
 }  
 }  
  
 return graph;  
}  
  
matrix\_graph\* init\_graph\_from\_cgi\_client(){  
 int data\_length = **CONTENT\_LENGTH**;  
 char\* data = post\_data(data\_length);  
 if (!data\_length){printf("error,%d\n",data\_length);exit(1);}  
 matrix\_graph\* graph = init\_graph\_from\_string(data);  
 **row**(graph->vertexes,graph->size,0,i)  
 free(data);  
 return graph;  
}  
  
int edge\_count(matrix\_graph\* graph, int kind){  
 int count = 0;  
 for (int i = 0; i < graph->size; ++i) {  
 for (int j = 0; j < graph->size; ++j) {  
 if (graph->matrix[i][j]){  
 count ++;  
 }  
 }  
 }  
  
 if (kind>=2){  
 count /= 2;  
 }  
  
 return count;  
}  
  
  
void remove\_vertex(matrix\_graph\* graph, int vrtxindex){  
 int data\_size;  
  
 void\* dst;  
 void\* src;  
  
 *//удаляем строку* for (int i=0; i<graph->size;i++){  
 for (int j = vrtxindex; j < graph->size-1; j++) {  
 graph->matrix[j][i] = graph->matrix[j+1][i];  
 }  
 }  
  
 *//удаляем столбец* for (int i=0; i<graph->size;i++){  
 dst = &(graph->matrix[i][vrtxindex]);  
 src = &(graph->matrix[i][vrtxindex+1]);  
  
 data\_size=(graph->size)-vrtxindex;  
  
 memmove(dst,src,data\_size\*sizeof(int));  
 }  
  
 *//удаляем название вершины* dst = &(graph->vertexes[vrtxindex]);  
 src = &(graph->vertexes[vrtxindex+1]);  
  
 memmove(dst,src,(graph->size-vrtxindex)\*sizeof(int));  
  
 graph->size--;  
}  
  
void add\_vrtx(matrix\_graph\* graph, int new\_vrtx){  
 int old\_size=graph->size;  
  
 graph->matrix = (int\*\*)realloc(graph->matrix, (old\_size+1) \* sizeof(int\*));  
 graph->matrix[old\_size] = (int\*)malloc((old\_size+1)\* sizeof(int));  
  
 for (int i = 0; i < old\_size; i++){  
 graph->matrix[i] = (int\*)realloc(graph->matrix[i], (old\_size+1)\* sizeof(int));  
 }  
  
 graph->size++;  
 int new\_size=graph->size;  
  
 for (int i=0; i<new\_size; i++){  
 graph->matrix[new\_size-1][i]=0;  
 graph->matrix[i][new\_size-1]=0;  
 }  
  
  
 graph->vertexes = realloc(graph->vertexes,sizeof(int)\*new\_size);  
 graph->vertexes[new\_size-1] = new\_vrtx;  
}

**cgi.h**

#ifndef **GRAPHMATRIX\_CGI\_H**#define **GRAPHMATRIX\_CGI\_H**#endif *//GRAPHMATRIX\_CGI\_H*#include "stdlib.h"  
#include "stdio.h"  
#include "string.h"  
  
#define **REQ\_MTD** getenv("REQUEST\_METHOD")  
#define **CONTENT\_TYPE** getenv("CONTENT\_TYPE")  
#define **CONTENT\_LENGTH** atoi(getenv("CONTENT\_LENGTH"))  
  
char\* post\_data(int content\_len);

**cgi.c**

#include "cgi.h"  
  
char\* post\_data(int content\_len){  
 if (!strcmp(**REQ\_MTD**,"POST")){  
 char\* post\_data = **NULL**;  
 if (content\_len != 0) {  
 post\_data = (char\*)malloc(content\_len);  
  
 char next\_char = **NULL**;  
 int count = 0;  
 while (next\_char!=**EOF** && count<content\_len){  
 next\_char = fgetc(**stdin**);  
 post\_data[count] = next\_char;  
 count ++;  
 }  
 post\_data[content\_len-1]=0;  
 }  
 return post\_data;  
 }  
 return **NULL**;  
}

**tools.h**

#ifndef GRAPHMATRIX\_TOOLS\_H  
#define GRAPHMATRIX\_TOOLS\_H  
  
#endif //GRAPHMATRIX\_TOOLS\_H  
  
#include <stdbool.h>  
#include <stdlib.h>  
  
#define HOME\_DIR "C:\\code\\Coloring\_graph\\"  
  
#define fill(block,block\_size,value,i) if (block && block\_size){for(size\_t i=0; i<block\_size; i++){block[i] = value;}}  
#define random\_int(min,max) min + rand()%(max - min +1)  
#define row(dst,size,begin,index) fill(dst,size,index+begin,index)  
  
#define \_VALUE\_OF(type,pointer) \*((type\*)pointer)  
#define new(type) malloc(sizeof(type))  
#define new\_const(type,value) &((type){value})  
#define new\_block(type,how\_much) malloc(sizeof(type)\*how\_much)  
#define del(ptr) free(ptr)  
  
#define and &&  
#define or ||  
  
int get\_int\_index\_binary(int\* arr, size\_t size, int num);  
  
int one\_or\_zero(float chance);  
  
int fst\_false(bool\* arr,int size);  
  
int positive\_or\_zero(int max, double chance);

**tools.c**

#include "tools.h"  
  
int one\_or\_zero(float chance){  
 int a = rand()%100;  
 if ((float)a<(100\*chance)){  
 return 1;  
 }  
 else{  
 return 0;  
 }  
}  
  
int positive\_or\_zero(int max, double chance){  
 int a = rand()%100;  
 if ((float)a<(100\*chance)){  
 return 1+rand()%max;  
 }  
 else{  
 return 0;  
 }  
}  
  
int get\_int\_index\_binary(int\* arr, size\_t size, int key){  
 size\_t left = 0;  
 size\_t right = size-1;  
 while (left<=right){  
 size\_t mid = (left+right)/2;  
 if (key==arr[mid]){  
 return mid;  
 }  
 if (key< arr[mid]){  
 right = mid - 1;  
 } else{  
 left = mid +1;  
 }  
 }  
 return -1;  
}  
  
int fst\_false(**bool**\* arr,int size){  
 for (int i = 0; i < size; ++i) {  
 if(!arr[i]){  
 return i;  
 }  
 }  
 return -1;  
}

**index.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Курсовая работа по ЛиОА</title>  
  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <link rel="stylesheet/less" type="text/css" href="assets/styles/main.less" />  
 <script src="//cdn.jsdelivr.net/npm/less" ></script>  
 <link rel="stylesheet" href= "assets/bootstrap/css/bootstrap.min.css">  
 <link rel="icon" href="assets/images/favicon-32x32.png" sizes="32x32">  
</head>  
<body>  
<div class="container page-wrap" id="mainContainer">  
 <div class="content">  
 <h1 style="display:inline;" id="h1Header">Раскраска графа</h1>  
 <section>  
 <ul class="nav nav-pills">  
 <div class="btn-group" role="group">  
 <button type="button" class="btn btn-default btn-sm dropdown-toggle" data-toggle="dropdown" aria-expanded="false">  
 <span class="glyphicon glyphicon-cog fa-fw"></span>  
 <span class="hidden-phone"> Граф </span>  
 <span class="caret"></span></button>  
 <ul class="dropdown-menu" role="menu">  
 <li>  
 <button type="button" class="btn btn-default btn-sm btn-submenu" id="ShowAdjacencyMatrix">  
 <span class="glyphicon glyphicon-th fa-fw"></span> Матрица смежности</button>  
 </li>  
 <li>  
 <button style="text-align: left;" type="button" class="btn btn-default btn-sm btn-submenu" id="RandomGraph">  
 <span class="glyphicon glyphicon-pencil fa-fw"></span> Создать рандомный граф </button>  
 </li>  
 <li class="divider"></li>  
 <li>  
 <button type="button" class="btn btn-default btn-sm btn-submenu" id="DeleteAll">  
 <span class="glyphicon glyphicon-remove fa-fw"></span> Удалить всё</button>  
 </li>  
 </ul>  
 </div>  
  
 <div class="btn-group" role="group">  
 <button type="button" class="btn btn-default btn-sm dropdown-toggle" data-toggle="dropdown" aria-expanded="false">  
 <span class="glyphicon glyphicon-cog fa-fw"></span>  
 <span class="hidden-phone"> Вид </span>  
 <span class="caret"></span></button>  
 <ul class="dropdown-menu" role="menu">  
 <li>  
 <button type="button" class="btn btn-default btn-sm btn-submenu" id="vertex-size">  
 <span class="glyphicon glyphicon-plus fa-fw"></span>Размер вершин</button>  
 <input id="range-vertex-radius" name="range" type="range" value="20">  
 </li>  
 <li>  
 <button type="button" class="btn btn-default btn-sm btn-submenu" id="edge-thickness">  
 <span class="glyphicon glyphicon-plus fa-fw"></span>Ширина ребер</button>  
 <input id="range-edge-thickness" name="range" type="range" value="21">  
 </li>  
 <li>  
 <span class="glyphicon glyphicon-plus fa-fw"></span>Выделять ребра выбранной вершины</button>  
 <input id="edge-color" type="checkbox" name="standout" value="true">  
 </li>  
 </ul>  
 </div>  
  
 <button type="button" class="btn btn-default btn-sm" id="Default"><span class="glyphicon glyphicon-fullscreen fa-fw">  
 </span><span class="hidden-phone"> По умолчанию <sub style="color:#AAAAAA">m</sub></span>  
 </button>  
  
 <button type="button" class="btn btn-primary btn-sm" id="AddGraph">  
 <span class="glyphicon glyphicon-plus fa-fw"></span>  
 <span class="hidden-phone"> Добавить вершину <sub style="color:#AAAAAA">v</sub></span>  
 </button>  
  
 <div class="btn-group" role="group">  
 <button type="button" class="btn btn-default btn-sm dropdown-toggle" data-toggle="dropdown" aria-expanded="false">  
 <span class="glyphicon glyphicon-cog fa-fw"></span>  
 <span class="hidden-phone"> Соединить вершины </span>  
 <span class="caret"></span></button>  
 <ul class="dropdown-menu" role="menu">  
 <li>  
 <button style="text-align: left" type="button" class="btn btn-default btn-sm btn-submenu" id="oriented">  
 <span class="glyphicon glyphicon-plus fa-fw"></span> Ориенированное ребро </button>  
 </li>  
 <li>  
 <button style="text-align: left" type="button" class="btn btn-default btn-sm btn-submenu" id="not-oriented">  
 <span class="glyphicon glyphicon-plus fa-fw"></span> Неориенированное ребро </button>  
 </li>  
  
 </ul>  
 </div>  
  
 <button type="button" class="btn btn-default btn-sm" id="DeleteObject"><span class="glyphicon glyphicon-remove fa-fw"></span>  
 <span class="hidden-phone"> Удалить <sub style="color:#AAAAAA">r</sub></span></button>  
  
 <button type="button" class="btn btn-default btn-sm" data-toggle="dropdown" aria-expanded="false" id="color">  
 <span class="glyphicon glyphicon-cog fa-fw "></span><span class="hidden-phone">Раскрасить </span><span class="caret"></span>  
 </button>  
  
 <button type="button" class="btn btn-default btn-sm" data-toggle="dropdown" aria-expanded="false" id="save-image">  
 <span class="glyphicon glyphicon-cog fa-fw "></span><span class="hidden-phone">Сохранить изображение </span><span class="caret"></span>  
 </button>  
  
 </ul>  
 </section>  
 <div id="message" class="alert alert-success" role="alert">Кликните на рабочую область, чтобы добавить вершину</div>  
 <canvas width="1100" height="600" id="field" ></canvas>  
 <div class="modal" id="basicModal" tabindex="-1" role="dialog" aria-labelledby="basicModal" aria-hidden="true">  
 <div class="modal-dialog">  
 <div class="modal-content">  
 <div class="modal-header">  
 <h4 class="modal-title" >Матрица смежности</h4>  
 <button type="button" class="close" data-dismiss="modal" aria-label="Close">  
 <span aria-hidden="true">×</span>  
 </button>  
 </div>  
 <div class="modal-body">  
 <textarea id="matrix-input" name="matrix-input" rows="10" cols="50"></textarea>  
 </div>  
 <div class="modal-footer">  
 <button type="button" class="btn btn-default close" data-dismiss="modal">Закрыть</button>  
 <button type="button" class="btn btn-primary close confirm-matrix">Создать</button>  
 </div>  
 </div>  
 </div>  
 </div>  
 <div class="modal" id="basicModal2" tabindex="-1" role="dialog" aria-labelledby="basicModal" aria-hidden="true">  
 <div class="modal-dialog">  
 <div class="modal-content">  
 <div class="modal-header">  
 <h4 class="modal-title" >Создать рандомный граф</h4>  
 <button type="button" class="close" data-dismiss="modal" aria-label="Close">  
 <span aria-hidden="true">×</span>  
 </button>  
 </div>  
 <div class="modal-body">  
 <form class="graph-kind" style="margin-bottom:8px">  
 <p style="margin:0">Вид графа:</p>  
 <input type="radio" name="kind" value="Nor" checked="checked">  
 Неориентированный  
 <input type="radio" name="kind" value="Or" >  
 Ориентированный  
 </form>  
 <p style="margin:0">Размер графа:</p>  
 <input id="gr-size" type="number" value="5" style="margin-bottom:8px">  
 <p style="margin:0">Плотность графа:</p>  
 <input id="gr-density" name="range" type="range" style="margin-bottom:8px;width:50%;">  
  
 </div>  
 <div class="modal-footer">  
 <button type="button" class="btn btn-default close" data-dismiss="modal">Закрыть</button>  
 <button type="button" class="btn btn-primary close create-graph">Создать</button>  
 </div>  
 </div>  
 </div>  
 </div>  
  
 </div>  
  
  
  
</div>  
  
  
<script src="assets/scripts/jquery-2.2.4.min.js" ></script>  
<script src="assets/scripts/Layout/mainMenu.js" type="module"></script>  
<script src="assets/scripts/mainScript.js" type="module"></script>  
  
</body>  
</html>

**main.less**

body{  
 #field{  
 position: inherit;  
 display: inherit;  
 margin-right: auto;  
 margin-left: auto;  
 border: black 2px solid;  
  
 }  
  
 @media screen and (min-width: 576px) {#field{width: 500px;}}  
 @media screen and (min-width: 768px) {#field{width: 700px;}}  
 @media screen and (min-width: 992px) {#field{width: 920px;}}  
 @media screen and (min-width: 1200px) {#field{width: 1100px;}}  
 @media screen and (min-width: 1400px) {#field{width: 1280px;}}  
  
 .modal {  
 position: fixed; */\* фиксированное положение \*/* top: 0;  
 right: 0;  
 bottom: 0;  
 left: 0;  
 background: rgba(0,0,0,0.5); */\* цвет фона \*/* z-index: 10050;  
 display: none;  
 opacity: 0; */\* по умолчанию модальное окно прозрачно \*/* -webkit-transition: opacity 200ms ease-in;  
 -moz-transition: opacity 200ms ease-in;  
 transition: opacity 200ms ease-in; */\* анимация перехода \*/* pointer-events: none; */\* элемент невидим для событий мыши \*/* margin: 0;  
 padding: 0;  
 }  
 .modal-active {  
 position: fixed;  
 top: 0;  
 left: 0;  
 z-index: 1050;  
 display: block;  
 width: 100%;  
 height: 100%;  
 overflow: hidden;  
 outline: 0;  
 }  
  
 .dropdown-menu-active{  
 position: absolute;  
 top: 100%;  
 left: 0;  
 z-index: 1000;  
 min-width: 10rem;  
 padding: .5rem 0;  
 margin: .125rem 0 0;  
 font-size: 1rem;  
 color: #212529;  
 text-align: left;  
 list-style: none;  
 background-color: #fff;  
 background-clip: padding-box;  
 border: 1px solid rgba(0,0,0,.15);  
 border-radius: .25rem;  
 }  
}

**mainScript.js**

import {*addEdge*, *addVertex*, *remVertex*} from "./graph/manually.js";  
import {*VertexModel*,*GraphModel*} from './graph/models.js'  
import {*canvas\_arrow*, *canvas\_line*, *Dot*,*clear*} from "./canvasTools.js";  
import {*coordinates\_on\_line*,*vector\_length*} from "./math.js";  
import {*generateEdges*, *generateVertexes*} from "./graph/VertexesGeneration.js";  
import {*colorGraph*, *Graph*, *readIndpSets*, *matrixToString*, *readMatrix*, *inti\_randomMatrix*} from "./graph/graph.js"  
import {*HideAll*} from "./Layout/mainMenu.js";  
  
*//macros*function el(elementName){let el = document.getElementById(elementName);if (el){return el;} else{ return document.getElementsByClassName(elementName)[0];}}  
  
let canvas = document.getElementById('field');  
let fieldOffset = new *Dot*(canvas.offsetLeft,canvas.offsetTop);  
let current\_primary\_button = "AddGraph";  
  
let set\_primary\_button = function (new\_btn){  
 el(current\_primary\_button).className = el(current\_primary\_button).className.replace("primary","default");  
 el(new\_btn).className = el(new\_btn).className.replace("default","primary");  
 current\_primary\_button = new\_btn;  
}  
  
let ctx = canvas.getContext('2d');  
let main\_colorToSave = '#f0a74f';  
let id\_colorToSave = '#000000';  
  
let main\_graph\_model = new *GraphModel*('main',canvas,ctx);  
let main\_graph = new *Graph*(main\_graph\_model,[[]]);  
  
let resizeField = function (){  
 canvas.width = canvas.clientWidth;  
 fieldOffset = new *Dot*(canvas.offsetLeft,canvas.offsetTop);  
  
 console.log(fieldOffset);  
 main\_graph.draw();  
}  
  
resizeField();  
  
  
console.log(fieldOffset);  
  
let fieldState = 'addV';  
let running = false;  
let standoutEdges = false;  
let vertex\_to\_move = null;  
  
let mouse\_is\_in\_vertex = function (vertex,clientX,clientY){  
 let len = *vector\_length*(vertex.x,vertex.y,clientX,clientY);  
 return len <= vertex.radius;  
}  
  
let determineVertex = function (graph,clientX,clientY){  
 let vertex;  
 for (let i = 0; i<graph.vertexes.length; i++){  
 vertex = graph.vertexes[i];  
 if (mouse\_is\_in\_vertex(vertex,clientX- fieldOffset.x,clientY-fieldOffset.y)){  
 return vertex;  
 }  
 }  
 return null;  
}  
  
let moveState = function (e){  
 fieldOffset.x = canvas.offsetLeft;  
 fieldOffset.y = canvas.offsetTop;  
  
 vertex\_to\_move = determineVertex(main\_graph\_model,e.clientX,e.clientY);  
 if (vertex\_to\_move) {  
 main\_colorToSave = vertex\_to\_move.main\_color;  
 id\_colorToSave = vertex\_to\_move.id\_color;  
 vertex\_to\_move.main\_color = '#90642F';  
 vertex\_to\_move.id\_color = '#000000';  
 main\_graph.draw();  
  
 if (standoutEdges){  
 let buf = main\_graph.model.edgeThickness;  
 main\_graph.model.edgeThickness = main\_graph.model.edgeThickness\*2;  
  
 for (let i = 0; i < main\_graph.model.edges.length; i++) {  
 if (main\_graph.model.edges[i].vertexes[0].id === vertex\_to\_move.id ||  
 main\_graph.model.edges[i].vertexes[1].id === vertex\_to\_move.id){  
 main\_graph.model.edges[i].draw(main\_graph.model.context,main\_graph.model.edgeThickness,'orange');  
 }  
 }  
  
 vertex\_to\_move.draw(main\_graph.model.context);  
 main\_graph.model.edgeThickness = buf;  
 }  
  
 running = true;  
 }  
}  
  
let addVState = function (e){  
 let nextIndex = main\_graph.model.vertexes.length;  
 let vertex = new *VertexModel*(  
 nextIndex,  
 nextIndex,  
 e.clientX - fieldOffset.x,  
 e.clientY - fieldOffset.y,  
 '#f0a74f',  
 '#000000',  
 main\_graph.model.context);  
 *addVertex*(main\_graph,vertex);  
 main\_graph.draw();  
}  
  
let vertexes\_to\_connect = [null,null];  
let vertex\_to\_reset;  
let addEState = function (e,kind){  
 if (vertexes\_to\_connect[0] && vertexes\_to\_connect[1]){  
 vertexes\_to\_connect = [null,null];  
 }  
  
 if (vertexes\_to\_connect[0]){  
 vertexes\_to\_connect[1] = determineVertex(  
 main\_graph.model,  
 e.clientX,  
 e.clientY)  
  
 if (vertexes\_to\_connect[1]) {  
 main\_colorToSave = vertexes\_to\_connect[1].main\_color;  
 id\_colorToSave = vertexes\_to\_connect[1].id\_color;  
 vertexes\_to\_connect[1].main\_color = '#90642F';  
 vertexes\_to\_connect[1].id\_color = '#000000';  
 vertex\_to\_reset = vertexes\_to\_connect[1];  
 main\_graph.draw();  
 }  
 }  
 else{  
 vertexes\_to\_connect[0] = determineVertex(  
 main\_graph.model,  
 e.clientX,  
 e.clientY)  
  
 if (vertexes\_to\_connect[0]){  
 main\_colorToSave = vertexes\_to\_connect[0].main\_color;  
 id\_colorToSave = vertexes\_to\_connect[0].id\_color;  
 vertexes\_to\_connect[0].main\_color = '#90642F';  
 vertexes\_to\_connect[0].id\_color = '#000000';  
 vertex\_to\_reset = vertexes\_to\_connect[0];  
 main\_graph.draw();  
 }  
 }  
  
 if (vertexes\_to\_connect[0] && vertexes\_to\_connect[1]){  
 *addEdge*(main\_graph,vertexes\_to\_connect[0],vertexes\_to\_connect[1],kind);  
 }  
}  
  
let vertex\_to\_delete;  
let edge\_to\_delete;  
let removeState = function (e){  
 vertex\_to\_delete = determineVertex(main\_graph\_model,e.clientX,e.clientY);  
 if (vertex\_to\_delete){  
 *remVertex*(main\_graph,vertex\_to\_delete.index);  
 main\_graph.draw();  
 }  
}  
  
  
  
canvas.onmousemove = function (e){  
 if (vertex\_to\_move && running && fieldState === 'move'){  
 vertex\_to\_move.x = e.clientX - fieldOffset.x;  
 vertex\_to\_move.y = e.clientY - fieldOffset.y;  
 main\_graph.draw();  
 }  
 else{return null;}  
};  
  
canvas.onmousedown = function (e){  
 switch (fieldState){  
 case 'move':  
 moveState(e);  
 break;  
 case 'addV':  
 addVState(e);  
 break;  
 case 'addEO':  
 addEState(e,0);  
 break;  
 case 'addENO':  
 addEState(e,1);  
 break;  
 case 'remove':  
 removeState(e);  
 break;  
  
 }  
};  
  
function resetVertex(vertex){  
 vertex.main\_color = main\_colorToSave;  
 vertex.id\_color = id\_colorToSave;  
}  
canvas.onmouseup = function (e){  
 switch (fieldState){  
 case 'move':  
 if (vertex\_to\_move) {  
 resetVertex(vertex\_to\_move);  
 vertex\_to\_move = null;  
 running = false;  
 main\_graph.draw();  
 }  
 break;  
 case 'addV':  
 break;  
 case 'addEO':  
 if (vertex\_to\_reset) {  
 resetVertex(vertex\_to\_reset);  
 main\_graph.draw();  
 }  
 break;  
 case 'addENO':  
 if (vertex\_to\_reset) {  
 resetVertex(vertex\_to\_reset);  
 main\_graph.draw();  
 }  
 break;  
 case 'remove':  
 break;  
  
 }  
};  
  
el('range-edge-thickness').oninput = function (e){  
 main\_graph.model.edgeThickness = Number(e.target.value)/5;  
 main\_graph.draw();  
};  
el('range-vertex-radius').oninput = function (e){  
 for (let i=0; i<main\_graph.model.vertexes.length; i++){  
 main\_graph.model.vertexes[i].radius = Number(e.target.value);  
 }  
 main\_graph.draw();  
};  
  
document.addEventListener('keypress', function (e){  
 switch (e.code){  
 case 'KeyM':  
 fieldState = 'move';set\_primary\_button('Default');  
 break;  
 case 'KeyV':  
 fieldState = 'addV';set\_primary\_button('AddGraph');  
 break;  
 case 'KeyR':  
 fieldState = 'remove';set\_primary\_button('DeleteObject');  
 break;  
 }  
});  
el('edge-color').onclick = function (e){standoutEdges = e.target.checked;};  
el('Default').onclick = function (){  
 fieldState = 'move';  
 set\_primary\_button('Default');  
 *HideAll*();}  
el('AddGraph').onclick = function (){  
 fieldState = 'addV';  
 set\_primary\_button('AddGraph');  
 *HideAll*();}  
el('oriented').onclick = function (){  
 fieldState = 'addEO';  
 *HideAll*();}  
el('not-oriented').onclick = function (){  
 fieldState = 'addENO';  
 *HideAll*();}  
el('DeleteObject').onclick = function (){  
 fieldState = 'remove';  
 set\_primary\_button('DeleteObject');  
 *HideAll*();}  
el('color').onclick = function () {  
 const data = *matrixToString*(main\_graph.matrix,' ');  
 console.log(main\_graph.matrix);  
 console.log("client:\n");  
 console.log(data);  
  
 let sets;  
 if (data) {  
 $.ajax({  
 async: true,  
 url: "cgi-bin/Coloring.exe",  
 type: "post",  
 data: data,  
 dataType: "text",  
 success: function (response) {  
 console.log("server:\n");  
 console.log(response);  
 sets = *readIndpSets*(String(response));  
 *colorGraph*(main\_graph.model,sets);  
 main\_graph.draw();  
 }  
 });  
 }  
  
 else{  
 *alert*("Граф не задан");  
 }  
};  
el('confirm-matrix').onclick = function (){  
 let textarea = el('matrix-input');  
 main\_graph.matrix = *readMatrix*(textarea.value);  
 if (main\_graph.matrix){  
 let names = [];  
 for (let i = 0; i < main\_graph.matrix.length; i++) {names[i]=i;}  
 main\_graph.model.vertexes = *generateVertexes*(main\_graph.model,names,main\_graph.matrix.length);  
 main\_graph.model.edges = *generateEdges*(main\_graph.matrix,main\_graph.model.vertexes);  
 main\_graph.draw();  
 *HideAll*();  
 }else{  
 textarea.value = "";  
 *alert*("Не удалось прочитать матрицу");  
 main\_graph.matrix = [];  
 }  
}  
el('create-graph').onclick = function (){  
 const size = Number(el("gr-size").value);  
 if (size < 1 || size > 200){  
 *alert*("Размер графа должен быть больше 1, и не привышать 200")  
 }  
 else{  
 const kind = Number(el("graph-kind").children[1].checked);  
 const density = Number(el("gr-density").value);  
 main\_graph.matrix = *inti\_randomMatrix*(kind,size,density);  
  
 let names = [];  
 for (let i = 0; i < main\_graph.matrix.length; i++) {names[i]=i;}  
 main\_graph.model.vertexes = *generateVertexes*(main\_graph.model,names,main\_graph.matrix.length);  
 main\_graph.model.edges = *generateEdges*(main\_graph.matrix,main\_graph.model.vertexes);  
 main\_graph.draw();  
 }  
  
}  
el('save-image').onclick = function(){  
 const url = el("field").toDataURL('image/png')  
 let link = document.createElement("a");  
 link.download = url.substring((url.lastIndexOf("/") + 1), url.length);  
 link.href = url;  
 document.body.appendChild(link);  
 link.click();  
 document.body.removeChild(link);  
}  
el('DeleteAll').onclick = function (){  
 main\_graph.matrix = [[]];  
 main\_graph.model.vertexes = [];  
 main\_graph.model.edges = [];  
 main\_graph.draw();  
 *HideAll*();  
};  
el('ShowAdjacencyMatrix').onclick = function (){  
 const text = *matrixToString*(main\_graph.matrix,',');  
 if (text){  
 el('matrix-input').value = text;  
 }  
}  
  
window.onresize = resizeField;

**graph.js**

import {*canvas\_arrow*, *canvas\_line*, *clear*} from "../canvasTools.js";  
import {*coordinates\_on\_line*} from "../math.js";  
import {*getOneOrZero*} from "../math.js";  
  
let main\_colors = [  
 'DarkBlue',  
 'DarkGreen',  
 'DarkCyan',  
 'DarkRed',  
 'DarkMagenta',  
 '#FFC138',  
 'Gray',  
 'DarkGray',  
 'Blue',  
 'Green',  
 'Cyan',  
 'Red',  
 'Magenta',  
 'Yellow',  
 'White']  
  
let id\_colors = [  
 'White',  
 'White',  
 'White',  
 'White',  
 'White',  
 'Black',  
 'Black',  
 'Black',  
 'White',  
 'White',  
 'White',  
 'White',  
 'White',  
 'Black',  
 'Black']  
  
export let *readIndpSets* = function (data){  
 let sets = [[]];  
 let strNumber='';  
 let setCount=0;  
 let j=0;  
 for (let i=0; i<data.length; i++){  
 if (data[i] !== ','){  
 if (data[i]!=='\n'){  
 strNumber+=data[i];  
 }  
 else{  
 sets.push([]);  
 setCount++;  
 j=0;  
 }  
 }  
 else{  
 sets[setCount][j] = Number(strNumber);  
 strNumber='';  
 j++;  
 }  
 }  
 return sets;  
}  
  
export let *readMatrix* = function (data){  
 let str\_len = data.indexOf('\n');  
  
 let order = Math.floor((str\_len+1)/2);  
 let matrix = [];  
 let next\_num;  
  
 if (order){  
 for (let i=0; i<order; i++){  
 matrix.push([]);  
 for (let j=0; j<order; j++){  
 try{  
 next\_num = Number(data[2\*j+i\*(str\_len+1)]);  
 if (next\_num){  
 matrix[i][j] = next\_num;  
 } else{  
 return null;  
 }  
 } catch (err){  
 return null;  
 }  
  
 }  
 }  
 } else{  
 return null;  
 }  
  
 return matrix;  
}  
  
export let *inti\_randomMatrix* = function (kind,size,density){  
 let matrix = [];  
 let i,j,c;  
 c=0;  
  
 for (let k = 0; k < size; k++) {  
 matrix.push([]);  
 }  
 for (i = c; i < size; i++) {  
 for (j = c; j < size; j++) {  
 if (i!==j){  
 if (kind){  
 matrix[i][j] = matrix[j][i] = *getOneOrZero*(density);  
 }  
 else{  
 matrix[i][j] = *getOneOrZero*(density);  
 matrix[j][i] = 0;  
 if (!matrix[i][j]){  
 matrix[j][i] = *getOneOrZero*(density);;  
 }  
 }  
 }  
 else{  
 matrix[i][j] = 0;  
 }  
 }  
 c++;  
 }  
  
 return matrix;  
}  
  
export let *colorGraph* = function (graphModel,sets){  
 for (let i=0; i<sets.length; i++) {  
 for (let j=0; j<sets[i].length; j++){  
 graphModel.vertexes[sets[i][j]].main\_color = main\_colors[i];  
 graphModel.vertexes[sets[i][j]].id\_color = id\_colors[i];  
 }  
 }  
}  
  
export function *Graph*(model,matrix){  
 this.model = model;  
 this.matrix = matrix;  
 this.draw = function (){  
 *clear*(this.model.canvas,this.model.context);  
 this.model.drawEdges();  
 this.model.drawVertexes();  
 }  
}  
  
export let *matrixToString* = function (matrix,\_break){  
 if (matrix[0][0] !== undefined){  
 let str = '';  
 for (let i = 0; i < matrix.length; i++) {  
 for (let j = 0; j < matrix.length - 1; j++) {  
 str += String(matrix[i][j])+\_break;  
 }  
 str+= String(matrix[i][matrix.length - 1]) + '\n';  
 }  
 return str;  
 }else{  
 return null;  
 }  
}

**manually.js**

import {*EdgeModel*} from "./models.js";  
  
export let *addVertex* = function(graph,vrtx){  
 let gSize = graph.model.vertexes.length;  
  
 if (gSize){  
 graph.matrix.push([]);  
 for (let i = 0; i < gSize; i++) {  
 graph.matrix[i].push(0);  
 graph.matrix[gSize].push(0);  
 }  
 graph.matrix[gSize].push(0);  
 }  
 else{  
 graph.matrix[0][0] = 0;  
 }  
 graph.model.vertexes.push(vrtx);  
}  
  
export let *addEdge* = function(graph,vrtx1,vrtx2,kind){  
 *//kind = 0 - oriented, 1 - not oriented* const next\_index = graph.model.vertexes.length;  
 if (kind){  
 graph.matrix[vrtx1.index][vrtx2.index] = graph.matrix[vrtx2.index][vrtx1.index] = 1;  
 }else{  
 graph.matrix[vrtx1.index][vrtx2.index] = 1;  
 }  
 graph.model.edges.push(new *EdgeModel*(next\_index,vrtx1,vrtx2,kind));  
}  
  
export let *remVertex* = function (graph,vertex\_index){  
 let gSize = graph.model.vertexes.length;  
  
 if (gSize>1){  
 let count = graph.model.edges.length;  
 for (let i = 0; i < count; i++) {  
 if (graph.model.edges[i].vertexes[0].index === vertex\_index ||  
 graph.model.edges[i].vertexes[1].index === vertex\_index)  
 {  
 graph.model.edges.splice(i,1);  
 i--;  
 count--;  
 }  
 }  
  
 for (let i = 0; i < gSize; i++) {  
 graph.matrix[i].splice(vertex\_index,1);  
 }  
  
 graph.matrix.splice(vertex\_index,1);  
 graph.model.vertexes.splice(vertex\_index,1);  
 for (let i = vertex\_index; i < graph.model.vertexes.length; i++) {  
 graph.model.vertexes[i].index --;  
 }  
 }  
 else{  
 graph.matrix = [[]];  
 graph.model.vertexes = [];  
 graph.model.edges = [];  
 }  
  
}  
  
export let *remEdge* = function(graph,edge){  
 *//kind = 0 - oriented, 1 - not oriented* if (edge.kind){  
 graph.matrix[edge.vertexes[0]][edge.vertexes[1]] = 0;  
 }  
 else{  
 graph.matrix[edge.vertexes[0]][edge.vertexes[1]] = graph.matrix[edge.vertexes[0]][edge.vertexes[1]] = 0;  
 }  
}

**models.js**

import {*canvas\_arrow*,*canvas\_line*,*Dot*} from "../canvasTools.js";  
import {*coordinates\_on\_line*} from "../math.js"  
  
export function *VertexModel*(id,index,x,y,main\_color,id\_color,context) {  
 this.id = id;  
 this.index = index;  
 this.context = NaN;  
 this.x = x;  
 this.y = y;  
 this.radius = 18;  
 this.main\_color = main\_color;  
 this.id\_color = id\_color;  
 this.id\_offset = Math.floor(this.radius/4.5);  
 this.draw = function(ctx) {  
 ctx.beginPath();  
 ctx.arc(this.x, this.y, this.radius, 0, Math.PI \* 2, true);  
 ctx.closePath();  
 ctx.fillStyle = this.main\_color;  
 ctx.fill();  
  
 ctx.fillStyle = this.id\_color;  
 ctx.font = String(this.radius \* 1.2)+'px serif';  
 ctx.fillText(this.id, this.x-this.id\_offset, this.y+this.id\_offset);  
  
 }  
}  
  
export function *EdgeModel*(id,vertex1,vertex2,kind){  
 this.id = id;  
 this.vertexes = [vertex1,vertex2];  
 this.kind = kind;*// 0 - oriented, 1 - not oriented* this.draw = function (context,thickness,color){  
 if (this.kind){  
 *canvas\_line*(  
 context,  
 this.vertexes[0].x,  
 this.vertexes[0].y,  
 this.vertexes[1].x,  
 this.vertexes[1].y,  
 thickness,  
 color)  
 }  
 else{  
 let to = *coordinates\_on\_line*(  
 this.vertexes[0].x,  
 this.vertexes[0].y,  
 this.vertexes[1].x,  
 this.vertexes[1].y,  
 this.vertexes[1].radius);  
  
 *canvas\_arrow*(  
 context,  
 this.vertexes[0].x,  
 this.vertexes[0].y,  
 to.x,  
 to.y,  
 thickness,  
 color)  
 }  
 }  
}  
  
  
export function *GraphModel*(name,canvas,context) {  
 this.name = name;  
 this.vertexes = [];  
 this.edges = [];  
 this.edgeThickness = 5;  
 this.edgesColor = 'black';  
 this.canvas = canvas;  
 this.context = context;  
 this.drawVertexes = function (){  
 for (let i=0; i<this.vertexes.length; i++){  
 this.vertexes[i].draw(context);  
 }  
 }  
 this.drawEdges = function (){  
 for (let i=0; i<this.edges.length; i++){  
 this.edges[i].draw(context,this.edgeThickness,this.edgesColor);  
 }  
 }  
}

**vertexGeneration.js**

import {Dot} from "../canvasTools.js";  
import {vector\_length} from "../math.js";  
import {EdgeModel, VertexModel} from "./models.js";  
import {*getRandomInt*} from "../math.js";  
  
  
export let *getNewPosition* = function (graphModel,vertexes){  
 let position = new *Dot*(0,0);  
 let len;  
 let posIsNotFine = true;  
  
 while (posIsNotFine){  
 position.x = *getRandomInt*(graphModel.canvas.width);  
 position.y = *getRandomInt*(graphModel.canvas.height);  
 posIsNotFine = false;  
 for (let i=0; i<vertexes.length; i++){  
 len = *vector\_length*(position.x,position.y,vertexes[i].x,vertexes[i].x);  
 if (len<vertexes[i].radius){  
 posIsNotFine = true;  
 break;  
 }  
 }  
 }  
 return position;  
}  
  
export let *generateVertexes* = function (graphModel,names,count){  
 let newPosition;  
 let vertexes = [];  
 for (let i=0; i<count;i++){  
 newPosition = *getNewPosition*(graphModel,vertexes);  
 vertexes.push(new *VertexModel*(  
 names[i],  
 i,  
 newPosition.x,  
 newPosition.y,  
 '#f0a74f',  
 '#000000',  
 graphModel.context)  
 );  
 }  
 return vertexes;  
}  
  
export let generateEdges = function (matrix,vertexes){  
 let c=0;  
 let i,j;  
 let edges = [];  
 for (i = c; i < matrix.length; i++) {  
 for (j = c; j < matrix.length; j++) {  
 if (matrix[i][j] && matrix[j][i]){  
 edges.push(new EdgeModel(  
 c,  
 vertexes[i],  
 vertexes[j],  
 1));  
 }  
 else if (matrix[i][j]){  
 edges.push(new EdgeModel(  
 c,  
 vertexes[i],  
 vertexes[j],  
 0));  
 }  
 else if (matrix[j][i]){  
 edges.push(new EdgeModel(  
 c,  
 vertexes[j],  
 vertexes[i],  
 0));  
 }  
 }  
 c++;  
 }  
 return edges;  
}

**mainMenu.js**

*//macros*function el(elementName){let el = document.getElementById(elementName);if (el){return el;} else{ return document.getElementsByClassName(elementName);}}  
  
let dropDownButtons = el('btn-sm dropdown-toggle');  
  
let menuGraph = dropDownButtons[0];  
let menuLook = dropDownButtons[1];  
let menuConnect = dropDownButtons[2];  
  
let matrixAdjButton = el('ShowAdjacencyMatrix');  
let randomGraphButton = el('RandomGraph')  
let matrixAdjModal = el('modal')[0];  
let randomGraphModal = el('modal')[1];  
let matrixInputClose = el('close');  
let activeModal;  
  
function FindActiveButton(){  
 let stack = [];  
 for (let i = 0; i < dropDownButtons.length; i++) {  
 if (dropDownButtons[i].nextSibling.nextSibling.className.indexOf("active")!==-1){  
 stack.push(i);  
 }  
 }  
 return stack;  
}  
  
  
let showHide = function (elem ) {  
 let elCn = elem.className;  
 if (elCn === 'dropdown-menu') {  
 elem.className += "-active";  
  
 } else {  
 elem.className = 'dropdown-menu';  
 }  
}  
  
export let *HideAll* = function (){  
 let elem;  
 let elCn;  
  
 const btn = FindActiveButton();  
 if (btn.length !== -1){  
 elem = dropDownButtons[btn[btn.length-1]].nextSibling.nextSibling;  
 elCn = elem.className;  
 if (elCn === 'dropdown-menu-active'){  
 elem.className = 'dropdown-menu';  
 }  
 }  
}  
  
let switchDropDown = function (){  
 const stack = FindActiveButton();  
 if (stack.length>0){  
 *HideAll*();  
 }  
}  
  
let showHideModal = function (elem) {  
 let elCn = elem.className;  
 if (elCn === 'modal') {  
 elem.className += "-active";  
 } else {  
 elem.className = 'modal';  
 }  
}  
  
document.addEventListener('keydown', function (e){  
 if (e.code === 'Escape'){  
 *HideAll*();  
 }  
});  
  
menuGraph.addEventListener('click', function (){  
 switchDropDown();  
 showHide(menuGraph.nextSibling.nextSibling);  
}, false);  
menuLook.addEventListener('click', function (){  
 switchDropDown();  
 showHide(menuLook.nextSibling.nextSibling);  
}, false);  
menuConnect.addEventListener('click', function (){  
 switchDropDown();  
 showHide(menuConnect.nextSibling.nextSibling);  
}, false);  
  
  
matrixAdjButton.addEventListener('click', function (){  
 activeModal = matrixAdjModal;  
 showHideModal(matrixAdjModal);  
 *HideAll*();  
}, false);  
  
randomGraphButton.addEventListener('click', function (){  
 activeModal = randomGraphModal;  
 showHideModal(randomGraphModal);  
 *HideAll*();  
}, false);  
  
for (let i = 0; i < matrixInputClose.length; i++) {  
 matrixInputClose[i].addEventListener('click', function (){  
 showHideModal(activeModal);  
 },false)  
}

**canvasTools.js**

export function *Dot*(x,y) {  
 this.x = x;  
 this.y = y;  
}  
  
export function *clear*(canvas,context){  
 context.clearRect(0,0, canvas.width, canvas.height);  
}  
  
export function *canvas\_line*(context, fromx, fromy, tox, toy, r, color){  
 context.beginPath();  
 context.moveTo(fromx,fromy);  
 context.lineTo(tox,toy);  
 context.strokeStyle = color;  
 context.lineWidth = r;  
 context.stroke();  
}  
  
export function *canvas\_arrow*(context, fromx, fromy, tox, toy, r, color){  
 var headlen = 10; *// length of head in pixels* var dx = tox - fromx;  
 var dy = toy - fromy;  
 var angle = Math.atan2(dy, dx);  
  
 context.beginPath();  
 context.moveTo(fromx, fromy);  
 context.lineTo(tox, toy);  
 context.lineTo(tox - headlen \* Math.cos(angle - Math.PI / 6), toy - headlen \* Math.sin(angle - Math.PI / 6));  
 context.moveTo(tox, toy);  
 context.lineTo(tox - headlen \* Math.cos(angle + Math.PI / 6), toy - headlen \* Math.sin(angle + Math.PI / 6));  
  
 context.strokeStyle = color;  
 context.lineWidth = r;  
 context.stroke();  
}

**maths.js**

import {*Dot*} from "./canvasTools.js";  
  
export function *getRandomInt*(max) {  
 return Math.floor(Math.random() \* Math.floor(max));  
}  
  
export function *getOneOrZero*(chance) {  
 let tmp = *getRandomInt*(101);  
 if (tmp<=chance){  
 return 1;  
 }  
 else{  
 return 0;  
 }  
}  
  
export let *vector\_length* = function (x1,y1,x2,y2){  
 return Math.sqrt((x2-x1)\*\*2+(y1-y2)\*\*2);  
}  
  
export let *coordinates\_on\_line* = function(ax,ay,bx,by,d){  
 let c = new *Dot*(0,0);  
 let k = d/*vector\_length*(ax,ay,bx,by);  
  
 let katetX = Math.abs(ax - bx)\*k;  
 let katetY = Math.abs(ay - by)\*k;  
  
 const arr\_ptr\_offset = 4;  
 if (bx>ax){  
 if (by>ay){ *//case →↓* c.x = bx - katetX - arr\_ptr\_offset;  
 c.y = by - katetY - arr\_ptr\_offset;  
 }  
 else{ *//case →↑* c.x = bx - katetX - arr\_ptr\_offset;  
 c.y = by + katetY + arr\_ptr\_offset;  
 }  
 }  
 else{  
 if (by>ay){ *//case ←↓* c.x = bx + katetX + arr\_ptr\_offset;  
 c.y = by - katetY - arr\_ptr\_offset;  
 }  
 else{ *//case ←↑* c.x = bx + katetX + arr\_ptr\_offset;  
 c.y = by + katetY + arr\_ptr\_offset;  
 }  
 }  
  
 return c;  
}