МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионально образования

“Московский авиационный институт (национальный исследовательский университет)”

Реферат по курсу:

II семестр

на тему:

Перечисление латиницей и Гамильтоновы контуры

Работу выполнил:

студент 1 курса

Группы М8О-107Б-2018

Гамов Павел Антонович

Проверила:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Научный руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

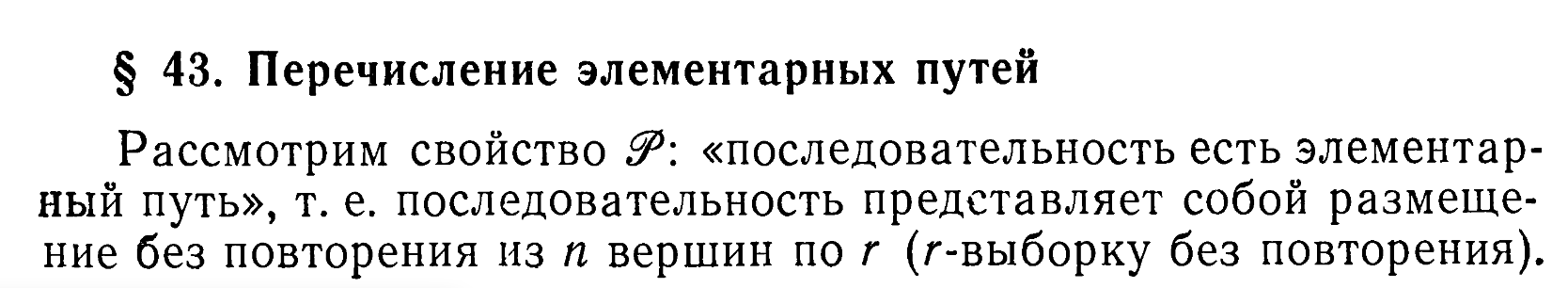
Москва, 2019

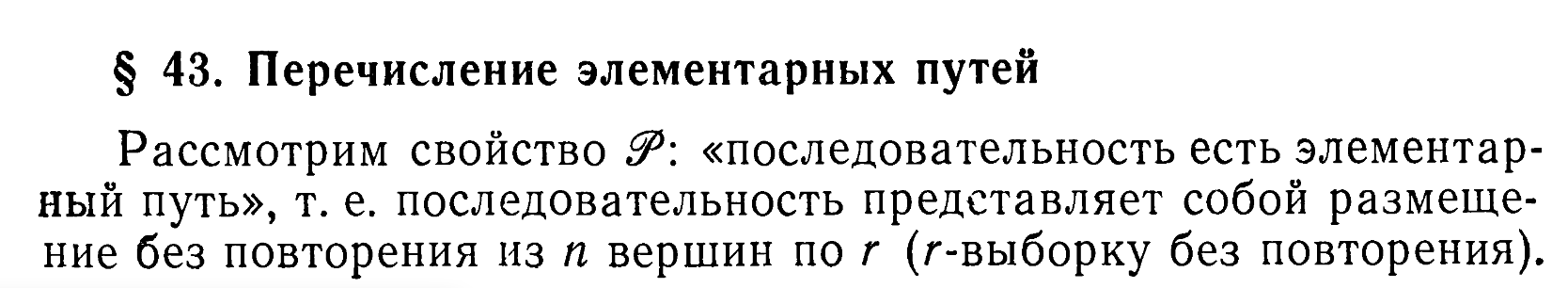
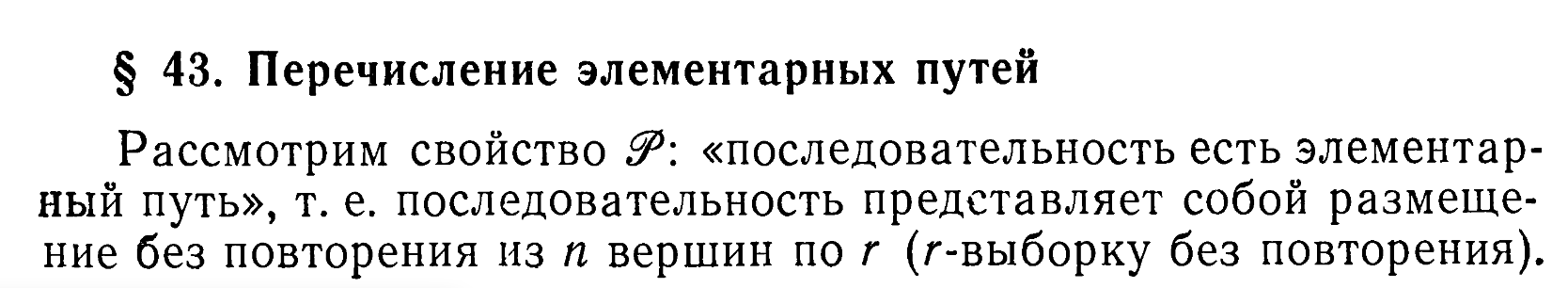
**Введение**

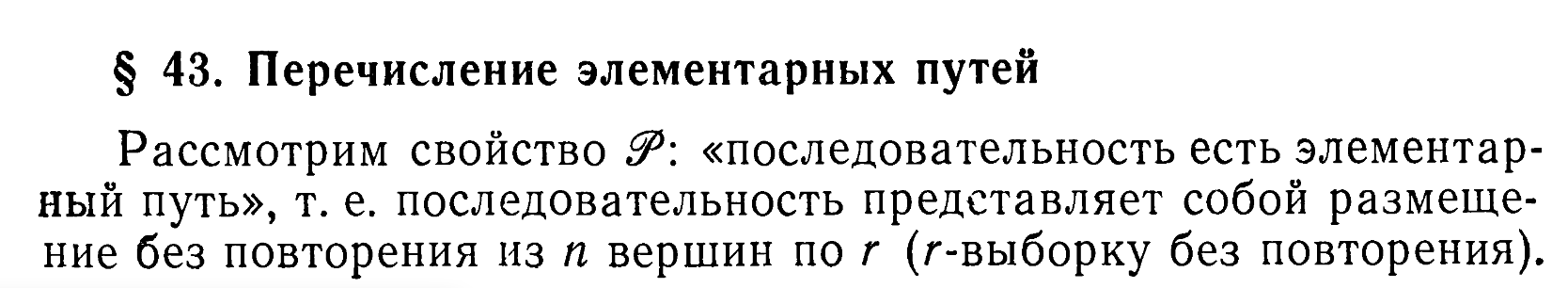
Использование языков программирования высокого уровня таких как Python или JavaScript для реализации быстрых проектов и различных алгоритмов в исследовательской работе набирает популярность. Данные языки позволяют не особо смотреть на внутренне устройство объектов как допустим в Си. Таким образом скорость разработки значительно увеличивается. Еще одним преимуществом является возможность использования графического интерфейса для отображения процесса работы. Данная техника более визуально понятна нежели вывод в консоль. Стоит также поговорить про веб разработку, которая в данной работе играет роль. Ведь современные браузеры поддерживают множество модулей и языков. Позволяют написать в короткие сроки веб сервер и запустить на нем какой-либо софт, позволяя пользователям использовать предоставленный функционал просто пройдя по ссылке.

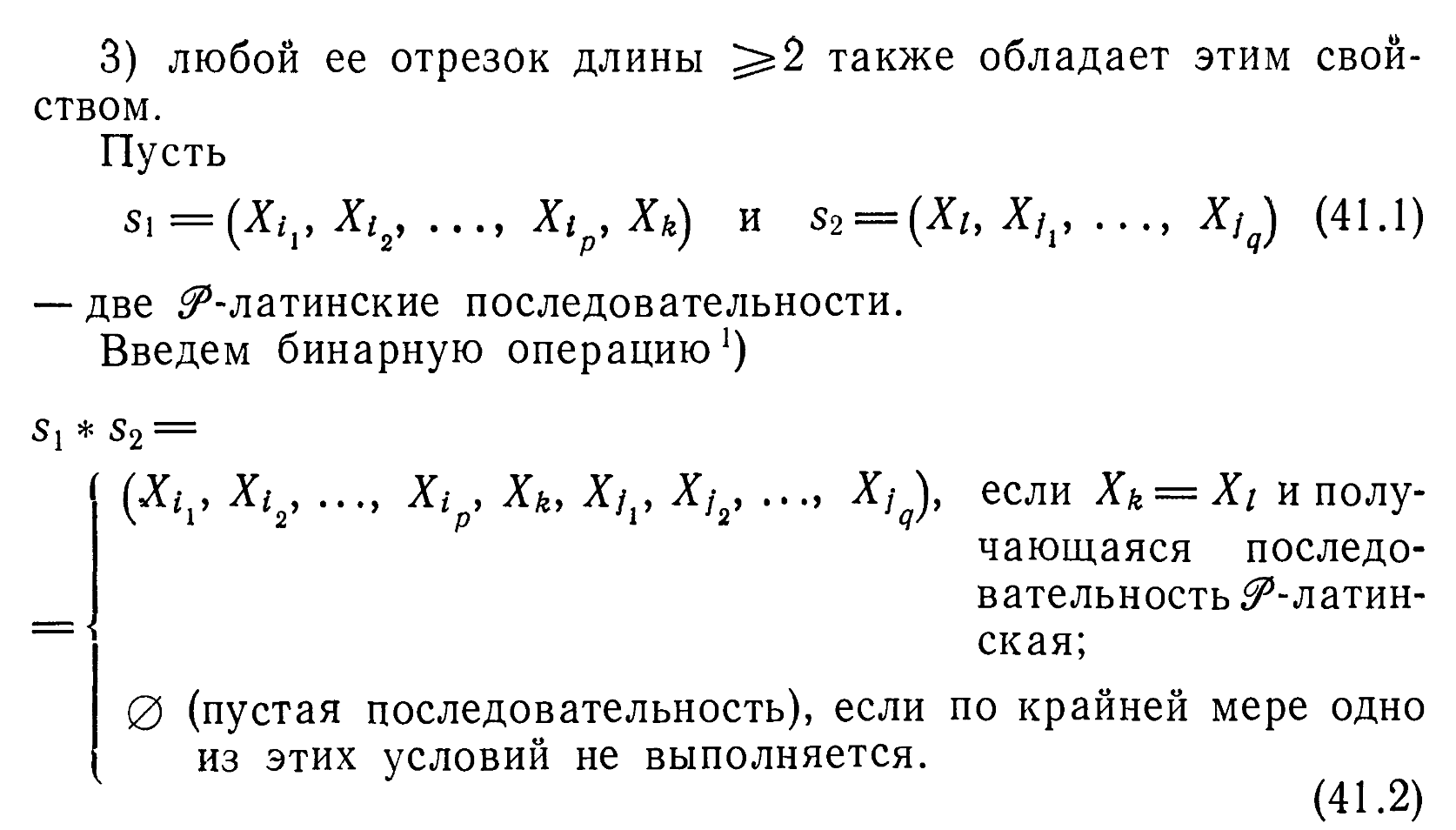
**Постановка задачи**

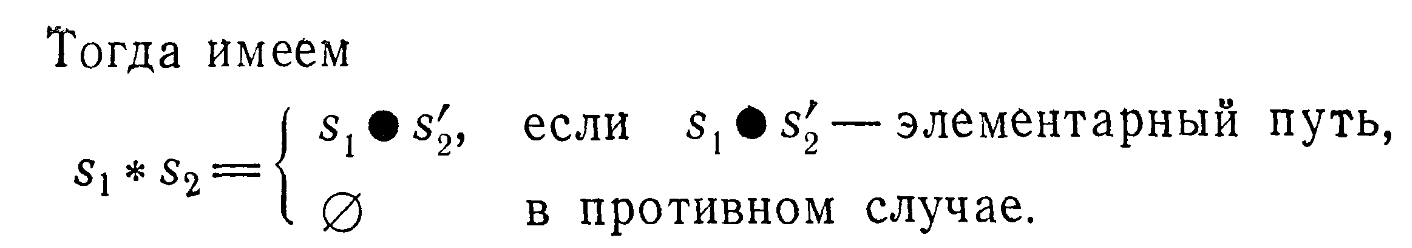
Создать программу, которая позволяет строить графы и находить Гамильтоновы контуры методом латинской композиции.

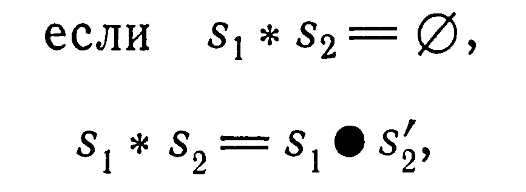
Рассмотрим свойство  “последовательность есть элементарный путь”, т. е. последовательность представляет собой размещение без повторения из n вершин по r (r – выборку без повторения).

Конечная последовательность (Xi1, Xi2, …, Xin) вершин графа G = (E, Г) называется латинской последовательностью со свойством  или  - латинской последовательностью, если:

1. она является путем.
2. она обладает свойствам 
3. любой отрезок длины>= 2 также обладает этим свойством

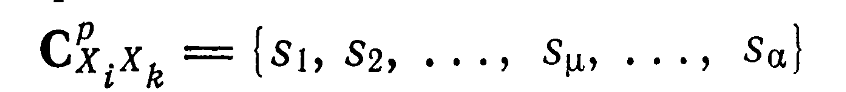




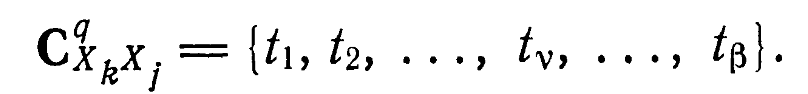


Где  последовательность без правой вершины.

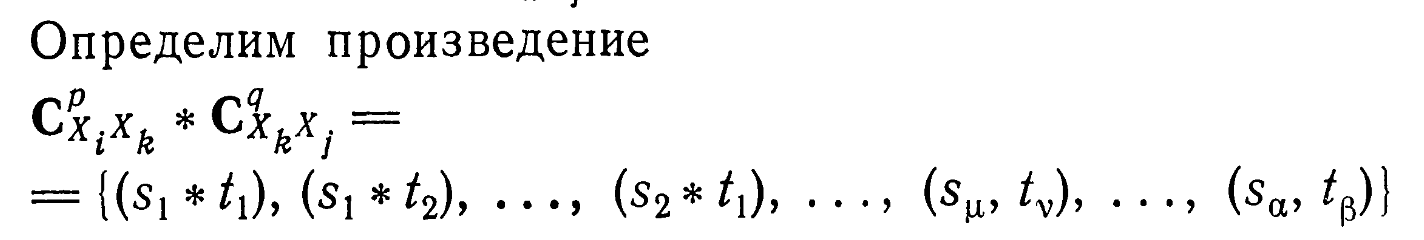
Обозначим через



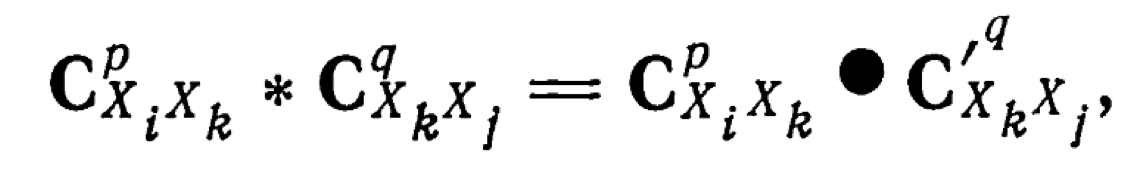
Подмножество - латинских последовательностей с p + 1 вершинами, начинающихся на Xi и оканчивающихся на Xk. Аналогично



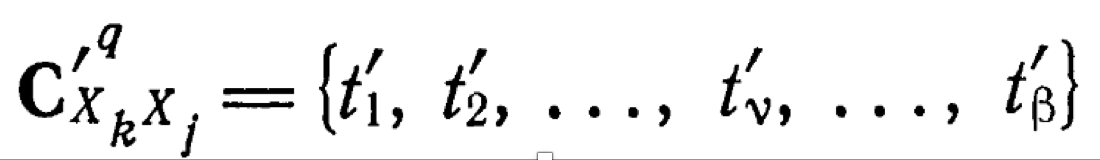
Определим произведение



Можно записать как

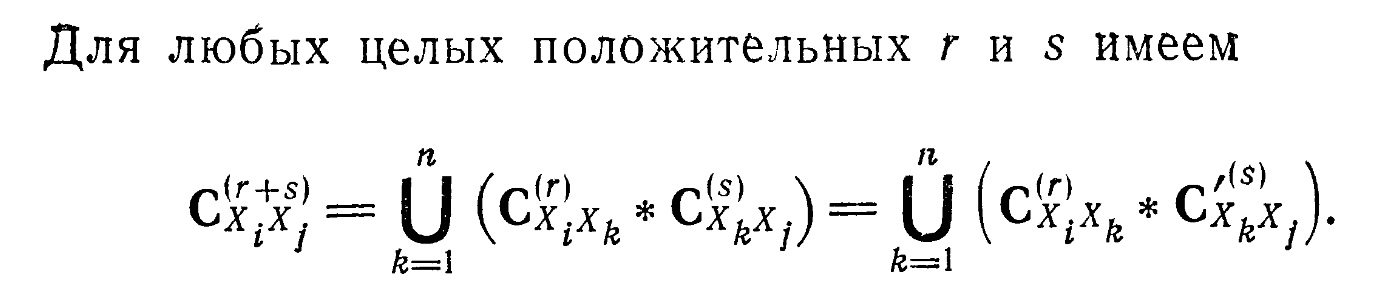


где

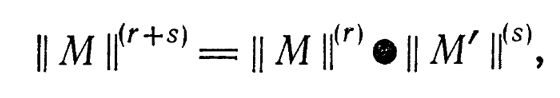


подмножество, состоящее из последовательностей t1, t2, …, t, у которых удалены первые вершины.

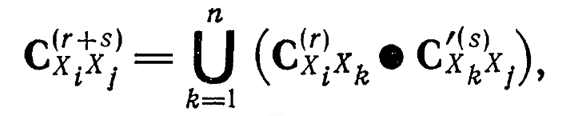
Для любых положительных r и s имеем



Исходя из вышеперечисленного, введем матрицы, которые назовем латинскими:



где на пересечении i-й строки и j-го столбца матрицы  стоит



Данный способ позволяет находить все пути длины до n+1 включительно. Нет гарантий, что полученные последовательности будут элементарными, поэтому для конкретных задач, допустим поиск всех путей из А в Б минуя С, требует отсекания всех лишних последовательностей, которые не подходят под наши условия.

Гамильтонов контур – последовательность не повторяющихся букв, за исключением конца и начала, которая является путем, таким, что по нему можно выйдя из вершины обойти весь граф, побывав в каждой вершине отличной от начала всего 1 раз, а потом вернуться обратно, образовав цикл, замкнутый путь.

Находится он по алгоритму, выполняя умножение латинских матриц до размера их n+1, далее, необходимо отсечь лишние пути, которые имеют повторяющиеся буквы или заканчиваются и начинаются не на одну букву.

**Разбор кода**

Стоит начать с устройства функционала библиотеки. Сама пустая программа состоит из двух функций setup() и draw(). Можно понять, что первая является пред загрузкой всех нужных вещей, такие как файлы, объявление структур, заполнение массивов данными и прочего. Вторая функция формально является вечным циклом, в котором происходит вся анимация, вызываются функции, которые двигают и меняют что-то на экране.

Также есть различные функции эвенты, которые выполняются асинхронно с основным циклом, выполняя свои действия, к примеру функция нажатия мыши, с которыми мы познакомимся позднее.

MacBook-Pro-Pavel:sem2\_ref pavelgamov$ ls

action.js addons lines.js sketch.js

actions.js buttons.js index.html matrix.js

Проект состоит из данных файлов. Так как язык позволяет разбивать проект как угодно, я разбил его по тематике, линейно будет проходить по всем файлам разбирая функции и алгоритмы, осознавая все аспекты языка и идеи.

**sketch.js**

MacBook-Pro-Pavel:sem2\_ref pavelgamov$ cat sketch.js

'use strict'

new p5();

var wight = 1000, hight = wight \* 2 / 3;

var fat = 25, dotArr = [], dotsize = 0, linesArr = [];

var bdot = [], bline = [], bor = [], bres = [], bsave = [];

var pivetXY = [];

var smegMatrix, nMatrix, oneMatrix, gamPath, drawGamPathPass = 0;

var number2letter = new Array(26);

var bstatus = null, bstatusColor = null, bstatusR = "OR", dmoveindex = -1, dataDotArr;

Объявление основных переменных.

function setup () {

createCanvas(wight, hight);

background(220);

var start = wight \* 2 / 3;

var h1 = 8; //10

var h2 = 40; //50

var d1 = 58; //68

var d2 = 50; //60

number2letter = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'];

bdot.push(new Button(start + h1, h2, 2\*d1+h1, d2, "Dot mode"));

bdot.push(new Button(start + 3\*h1 + 2\*d1, h2, d1, d2, "Dot dell"));

bdot.push(new Button(start + 4\*h1 + 3\*d1, h2, d1, d2, "Dot move"));

bline.push(new Button(start + h1, 2\*h2+d2, 2\*d1 + h1, d2, "Line mode"));

bline.push(new Button(start + 3\*h1 + 2\*d1, 2\*h2+d2, d1, d2, "Line dell"));

bres.push(new Button(start + h1, 3\*h2+2\*d2, 2\* d1 + h1, d2, "Res"));

bres.push(new Button(start + 3\*h1 + 2\*d1, 3\*h2+2\*d2, 2\* d1 + h1, d2, "RePos"));

bor.push(new Button(start + 4\*h1 + 3\*d1, 2\*h2+d2, d1, d2, "OR"));

bor.push(new Button(start + 5\*h1 + 4\*d1, 2\*h2+d2, d1, d2, "nOR"));

bsave.push(new Button(start + h1, hight - d2 - h2, d1, d2, "Save"));

}

setup функция предзагрузки.

function draw () {

background(220);

lineShow();

for (var i = 0; i < dotsize; i++) dotArr[i].show(i);

if (nodesRePosPass != 0) nodesRePos();

if (drawGamPathPass) drawGamPath();

whenPressed();

bshow();

}

Функция draw основной цикл, рисует цвет фона, показывает все линии, об этом позже, а так в общем отрисовывает всю графику поверх канваса каждый вызов.

**buttons.js**

function Button(x, y, xw, yw, btext = 0, color = [0])

{

this.x = x;

this.y = y;

this.xw = xw;

this.yw = yw;

this.btext = btext;

this.bcolor = [];

this.bselected = false;

this.show = function() {

if (this.bselected == true) {

fill(180);

stroke(190);

} else {

fill(200);

stroke(10);

}

rect(this.x, this.y, this.xw, this.yw);

if (this.btext != 0) {

fill(10);

textSize(12);

text(this.btext, this.x + this.xw / 10, this.y + this.yw / 1.5);

}

}

this.itenteract = function(x, y) {

if (!(x < this.x || x > this.x + this.xw || y < this.y || y > this.y + this.yw))

return 1;

}

}

function getFalse() {

for (var j = 0; j < bdot.length; j++) bdot[j].bselected = false;

for (var j = 0; j < bline.length; j++) bline[j].bselected = false;

}

function bshow() {

stroke(10);

line(wight \* 2 / 3, 0, wight \* 2 / 3, hight);

for (var i = 0; i < bdot.length; i++) bdot[i].show();

for (var i = 0; i < bline.length; i++) bline[i].show();

for (var i = 0; i < bor.length; i++) bor[i].show();

for (var i = 0; i < bres.length; i++) bres[i].show();

for (var i = 0; i < bsave.length; i++) bsave[i].show();

}

Набор данных функций содержит в себе класс кнопок, функций их отрисовки на экран, а также взаимодействия с ними.

**actions.js**

MacBook-Pro-Pavel:sem2\_ref pavelgamov$ cat actions.js

function mouseReleased() {

switch (bstatus) {

case "Dot mode":

if (mouseX < wight \* 2 / 3 - fat && dotArr.length < 26) {dotArr.push(new Dot\_(mouseX, mouseY));dotsize++;smegMatrix = reOplyMatrixSmeG(dotsize);if (drawGamPathPass) drawGamPathPass = 0;}

break;

case "Dot dell":

var povet = searchInter(mouseX, mouseY);

if (povet >= 0) {

for (var i = 0; i < dotsize; i++) {

if (i == povet)continue;

for (var j = 0; j < dotArr[i].path.length; j++)if (dotArr[i].path[j] == povet)dotArr[i].path.splice(j, 1);

}

for (var i = 0; i < dotsize; i++) for (var j = 0; j < dotArr[i].path.length; j++) if (povet < dotArr[i].path[j]) dotArr[i].path[j]--;

dotArr.splice(povet, 1);

dotsize--;

smegMatrix = reOplyMatrixSmeG(dotsize);

if (drawGamPathPass) drawGamPathPass = 0;

}

break;

case "Dot move":

dmoveindex = -1;

break;

case "Line mode":

var povet = searchInter(mouseX, mouseY);

if (pivetXY[2] == -1) {

pivetXY[0] = null; pivetXY[1] = null; pivetXY[2] = null;

} else {

if (povet >= 0 && povet != pivetXY[2]) {

if (drawGamPathPass) drawGamPathPass = 0;

if (bstatusR == "OR") {

if (!checkLine(povet, pivetXY[2])) {dotArr[povet].path.push(pivetXY[2]); dotArr[povet].path.sort();}

if (!checkLine(pivetXY[2], povet)) {dotArr[pivetXY[2]].path.push(povet); dotArr[pivetXY[2]].path.sort();}

} else if (bstatusR == "nOR") {

if (!checkLine(pivetXY[2], povet)) {dotArr[pivetXY[2]].path.push(povet); dotArr[pivetXY[2]].path.sort();}

}

}

pivetXY[0] = null; pivetXY[1] = null; pivetXY[2] = null; smegMatrix = reOplyMatrixSmeG(dotsize);

}

break;

case "Line dell":

var povet = searchInter(mouseX, mouseY);

if (povet >= 0 && povet != pivetXY[2]) {if (drawGamPathPass) drawGamPathPass = 0; lineDell(pivetXY[2], povet);}

pivetXY[0] = null; pivetXY[1] = null; pivetXY[2] = null; smegMatrix = reOplyMatrixSmeG(dotsize);

break;

}

}

MouseReleased – функция p5.js отвечающая за эвент отжатия кнопки мыши. Когда кнопка поднимается, срабатывает функции, в зависимости от положения курсора. Если была нажата кнопка dotMode, после поднятия кнопки, если условие выполнится, добавится новый объект в массив нодов, после чего он станет орисовываться в draw, ожидая новых эвентов. Дальше мы познакомимся с другими, но похожими функциями, такими как удерживание, клик и еще что-то.

function mouseClicked() {

if (mouseX > wight \* 2 / 3) {

for (var i = 0; i < bdot.length; i++) {

if (bdot[i].itenteract(mouseX, mouseY)) {getFalse(); bdot[i].bselected = true; bstatus = bdot[i].btext;}

}

for (var i = 0; i < bline.length; i++) {

if (bline[i].itenteract(mouseX, mouseY)) {getFalse(); bline[i].bselected = true; bstatus = bline[i].btext;}

}

for (var i = 0; i < bor.length; i++) {

if (bor[i].itenteract(mouseX, mouseY)) {for (var j = 0; j < bor.length; j++) bor[j].bselected = false; bor[i].bselected = true; bstatusR = bor[i].btext;}

}

for (var i = 0; i < bres.length; i++)

if (bres[i].itenteract(mouseX, mouseY))

if (i == 0) {

galminton();

} else if (i == 1) {

switch (nodesRePosPass) {

case 0: nodesRePosPass = 1; break;

case 1: nodesRePosPass = 0; break;

}

}

for (var i = 0; i < bsave.length; i++) {if (bsave[i].itenteract(mouseX, mouseY)) {if (dotArr.length != 0) {var json = dotArr; saveJSON(json, 'dotArr.json');}}}

}

}

MouseClicked – быстрый клик мыши по обьекту, если клик попал на одну из кнопок, старые сбрасываются, обновляется состояние, попадание по кнопке регистрирует функция класса кнопок. Также при нажатии на кнопку сохранить, программа собирает файл json и скачивает его на ваш компьютер.

function mousePressed() {

switch (bstatus) {

case "Line mode":pivetXY[0] = mouseX; pivetXY[1] = mouseY; pivetXY[2] = searchInter(mouseX, mouseY); break;

case "Line dell":pivetXY[0] = mouseX; pivetXY[1] = mouseY; pivetXY[2] = searchInter(mouseX, mouseY); break;

case "Mass mode":pivetXY[0] = mouseX; pivetXY[1] = mouseY; pivetXY[2] = searchInter(mouseX, mouseY); break;

}

}

MousePressed – срабатывает однократно и действует пока нажата кнопка. Предшественник функции MouseReleased.

function whenPressed() {

if (mouseIsPressed) {

switch (bstatus) {

case "Dot mode": if (mouseX < wight \* 2 / 3 - fat) {fill(255); stroke(3); ellipse(mouseX, mouseY, fat, fat);} break;

case "Dot move": if (mouseX < wight \* 2 / 3 - fat) {dmoveindex = searchInter(mouseX, mouseY); if (searchInter(mouseX, mouseY) >= 0) {dotArr[dmoveindex].x = mouseX; dotArr[dmoveindex].y = mouseY;}}break;

case "Line mode": stroke(10); line(pivetXY[0], pivetXY[1], mouseX, mouseY); break;

case "Line dell": stroke(10); line(pivetXY[0], pivetXY[1], mouseX, mouseY); break;

case "Mass mode": stroke(10); line(pivetXY[0], pivetXY[1], mouseX, mouseY); break;

}

}

}

WhenPressed – функция работающая пока нажата кнопка мыши. Тут нужна для анимации соединения нодов, а также их перемещения, пока она нажата, выбранный нод меняет свое положение в соответствии с положением мышки.

**lines.js**

function Dot\_(x, y)

{

this.x = x;

this.y = y;

this.fat = fat;

this.path = [];

this.show = function(number) {

fill(255);

stroke(10);

ellipse(this.x, this.y, this.fat, this.fat);

fill(10);

textSize(this.fat / 1.5);

text(number2letter[number], this.x - this.fat / 2, this.y - this.fat / 2);

}

}

Класс нодов графа.

function searchInter(x, y) {for (var i = 0; i < dotsize; i++) if (sqrt((x - dotArr[i].x)\*\*2 + (y - dotArr[i].y)\*\*2) < dotArr[i].fat) return i; return -1;}

function lineDell(index, povet) {

for (var i = dotArr[povet].path.length - 1; i >= 0; i--) if (dotArr[povet].path[i] == index) dotArr[povet].path.splice(i, 1);

for (var i = dotArr[index].path.length; i >= 0 ; i--) if (dotArr[index].path[i] == povet) dotArr[index].path.splice(i, 1);

}

function lineShow() {

for (var i = 0; i < dotsize; i++) {

for (var j = 0; j < dotArr[i].path.length; j++) {

stroke(10);

line(dotArr[i].x, dotArr[i].y, dotArr[dotArr[i].path[j]].x, dotArr[dotArr[i].path[j]].y);

var s = [(dotArr[dotArr[i].path[j]].x + dotArr[i].x)/2, (dotArr[dotArr[i].path[j]].y + dotArr[i].y)/2];

var n = [dotArr[dotArr[i].path[j]].y - dotArr[i].y, - dotArr[dotArr[i].path[j]].x + dotArr[i].x];

var nap = [dotArr[dotArr[i].path[j]].x - dotArr[i].x, dotArr[dotArr[i].path[j]].y - dotArr[i].y];

line(s[0] + n[0]/40 - nap[0]/20, s[1] + n[1]/40 - nap[1]/20, s[0] + n[0]/40 + nap[0]/20, s[1] + n[1]/40 + nap[1]/20);

line(s[0] + n[0]/40 + nap[0]/20, s[1] + n[1]/40 + nap[1]/20, s[0] + 2\*n[0]/60, s[1] + 2\*n[1]/60);

}

}

}

function checkLine(index, search) {for (var i = 0; i < dotArr[index].path.length; i++) if (dotArr[index].path[i] == search) return 1; return 0;}

function letterToInt(target) {for (var i = 0; i < number2letter.length; i++) if (number2letter[i] == target) return i;}

function drawGamPath() {for (var i = 0; i < gamPath.length - 1; i++) {stroke(255, 165, 0);line(dotArr[letterToInt(gamPath[i])].x, dotArr[letterToInt(gamPath[i])].y, dotArr[letterToInt(gamPath[i+1])].x, dotArr[letterToInt(gamPath[i+1])].y);}}

Данный файл содержит функции отрисовки на экран дуг, соединяющих вершины в графе, определяющие их направленность. Также содержит конструктор класса Вершин.

**matrix.js**

Основной файл, в котором находятся функции считающие матрицы с латинским перечислением, а также находящие Гамильтоновы контуры.

function makeMatrixSmeG(size) {

var arr = new Array(size);

for (var i = 0; i < size; i++) arr[i] = new Array(size);

for (var i = 0; i < size; i++) for (var j = 0; j < size; j++) arr[i][j] = 0;

return arr;

}

function reOplyMatrixSmeG(size) {

var arr = makeMatrixSmeG(size);

for (var i = 0; i < dotArr.length; i++) for (var j = 0; j < dotArr[i].path.length; j++) arr[i][dotArr[i].path[j]] = 1;

return arr;

}

Функции создания пустой матрицы а также матрицы смежности.

function makeLatMatrix(size) {

if (size == 0) return null;

var arr = new Array(size);

for (var i = 0; i < size; i++) { arr[i] = new Array(size); for (var j = 0; j < size; j++) arr[i][j] = [];}

for (var i = 0; i < size; i++) for (var j = 0; j < size; j++) if (smegMatrix[i][j] != 0) arr[i][j].push((number2letter[i] + number2letter[j]));

return arr;

}

Функция, возвращающая матрицу заполненную латинскими путями длины 2, она понадобится для дальнейшего алгоритма.

function makeLatMatrixDelLeft(size) {

if (size == 0) return null;

arr = new Array(size);

for (var i = 0; i < size; i++) arr[i] = new Array(size);

for (var i = 0; i < size; i++) for (var j = 0; j < size; j++) { if (smegMatrix[i][j] != 0) arr[i][j] = number2letter[j]; else arr[i][j] = "";}

return arr;

}

Функция, возвращающая матрицу, заполненную латинскими путями длины 1, то есть с обрезанными правыми краями, она понадобится для дальнейшего алгоритма, для умножения на нее матрицы предыдущей функции.

function composLatMatrix(mat1, mat2) {

var size = dotsize, arr = new Array(size);

for (var i = 0; i < size; i++) {arr[i] = new Array(size); for (var j = 0; j < size; j++) arr[i][j] = [];}

for (var i = 0; i < size; i++)

for (var j = 0; j < size; j++)

for (var k = 0; k < size; k++)

if (mat1[i][k].length != 0 && mat2[k][j] != 0)

for (var l = 0; l < mat1[i][k].length; l++)

arr[i][j].push(mat1[i][k][l] + mat2[k][j]);

return arr;

}

Функция, возвращающая результат выполнения умножения матрицы на матрицу. Заметим, что функция обрабатывает также и ячейки имеющие множество путей внутри.

function inStack(arr, target) {

for (var i = 0; i < arr.length; i++) if (arr[i] == target) return 0;

return 1;

}

function checkPathLite(matrix) {

var size = dotsize, stringPivet, stack = [];

for (var i = 0; i < size; i++)

for (var j = 0; j < size; j++) {

for (var k = 0; k < matrix[i][j].length; k++)

for (var l = 0; l < matrix[i][j][k].length - 2; l++) {

stringPivet = matrix[i][j][k][l];

for (var q = l + 1; q < matrix[i][j][k].length - 1; q++) if (matrix[i][j][k][q] == stringPivet) if (inStack(stack, k)) stack.push(k);

}

for (var pp = stack.length - 1; pp >= 0; pp--) matrix[i][j].splice(stack[pp], 1); stack = [];

}

}

function checkPath(matrix) {

var size = dotsize, stringPivet, stack = [];

for (var i = 0; i < size; i++)

for (var j = 0; j < size; j++) {

for (var k = 0; k < matrix[i][j].length; k++) {

for (var l = 0; l < matrix[i][j][k].length - 2; l++) {

stringPivet = matrix[i][j][k][l];

for (var q = l + 1; q < matrix[i][j][k].length - 1; q++) if (matrix[i][j][k][q] == stringPivet) if (inStack(stack, k)) stack.push(k);

}

if (matrix[i][j][k][0] != matrix[i][j][k][matrix[i][j][k].length - 1]) if (inStack(stack, k)) stack.push(k);

}

for (var pp = stack.length - 1; pp >= 0; pp--) matrix[i][j].splice(stack[pp], 1); stack = [];

}

}

Данные три функции нужны для проверки матрицы на наличие лишних, не элементарных путей. Те, которые имеют кольцевание, идут по одной вершине больше одного раза заносятся в список на удаление. После чего, функция берет значение из списка и удаляет путь из ячейки.

function galminton() {

if (dotsize == 0) return null;

nMatrix = makeLatMatrix(dotsize);

oneMatrix = makeLatMatrixDelLeft(dotsize);

for (var i = 0; i < dotsize - 1; i++) {nMatrix = composLatMatrix(nMatrix, oneMatrix); checkPathLite(nMatrix);}

checkPath(nMatrix);

//saveJSON(nMatrix, 'nMatrix.json');

for (var i = 0; i < nMatrix.length; i++) for (var j = 0; j < nMatrix[i].length; j++) if (nMatrix[i][j].length != 0) {gamPath = nMatrix[i][j].pop();drawGamPathPass = 1;}

}

Главная функция, использующая все предыдущее. Она в цикле вызывает умножения матрицы на матрицу, после чего отправляет ее на чистку. После выполнения цикла, матрица идет на финальную чистку, где отсеиваются оставшиеся пути. После чего вызывается цикл по матрице, возвращающий в глобальную переменную строку литер, также делает флаг равный 1. Если вернутся в draw, там есть функция, вызывающаяся при наличии флага. Данная функция прописана в lines.js и она красит граф по Гамильтонову контуру, давая визуальное представление об контуре.

**action.js**

MacBook-Pro-Pavel:sem2\_ref pavelgamov$ cat action.js

document.getElementById('import').onclick = function() {

var files =

document.getElementById('selectFiles').files;

console.log(files);

if (files.length <= 0) return false;

var fr = new FileReader();

fr.onload = function(e) {

console.log(e);

var resulted = JSON.parse(e.target.result), formatted = JSON.stringify(resulted, null, 2);

dotArr = [];

for (var i = 0; i < resulted.length; i++) {

dotArr.push(new Dot\_(resulted[i].x, resulted[i].y));

for (var j = 0; j < resulted[i].path.length; j++)

dotArr[i].path.push(resulted[i].path[j]);

dotsize++;

}

}

fr.readAsText(files.item(0));

};

var nodesRePosPass = 0;

function nodeDistance(A, B){return sqrt(pow(abs(A.x - B.x), 2) + pow(abs(A.y - B.y), 2));}

function nodeDistanceX(A, B){return abs(A.x - B.x);}

function nodeDistanceY(A, B){return abs(A.y - B.y);}

function xRel(A, B){if (A.x > B.x)return 1;else return -1;}

function yRel(A, B){if (A.y > B.y)return 1;else return -1;}

function inArr(a, arr){for (var i = 0; i < arr.length; i++){if(arr[i] == a)return 1;}return 0;}

function collectGradient(arrOfDots, number) {

var res = [0, 0], F;

for (var i = 0; i < arrOfDots.length; i++)

if (i != number) {

F = 5000/(nodeDistance(arrOfDots[i], arrOfDots[number]) \* nodeDistance(arrOfDots[i], arrOfDots[number]));

res[0] += F\*(nodeDistanceX(arrOfDots[number], arrOfDots[i])/nodeDistance(arrOfDots[number], arrOfDots[i]))\*xRel(arrOfDots[number], arrOfDots[i]);

res[1] += F\*(nodeDistanceY(arrOfDots[number], arrOfDots[i])/nodeDistance(arrOfDots[number], arrOfDots[i]))\*yRel(arrOfDots[number], arrOfDots[i]);

}

for (var i = 0; i < arrOfDots.length; i++)

if ((i != number) && inArr(i, arrOfDots[number].path)) {

F = (nodeDistance(arrOfDots[i], arrOfDots[number]));

res[0] -= 0.5\*(nodeDistanceX(arrOfDots[number], arrOfDots[i])/nodeDistance(arrOfDots[number], arrOfDots[i]))\*xRel(arrOfDots[number], arrOfDots[i]);

res[1] -= 0.5\*(nodeDistanceY(arrOfDots[number], arrOfDots[i])/nodeDistance(arrOfDots[number], arrOfDots[i]))\*yRel(arrOfDots[number], arrOfDots[i]);

}

return res;

}

function nodesRePos() //меняет позиции всех нодов

{

var rePos = [];

for (var i = 0; i < dotArr.length; i++)

rePos.push(collectGradient(dotArr, i));

for (var i = 0; i < dotArr.length; i++)

for (var j = 0; j < dotArr.length; j++)

if (i != j) {

if (dotArr[i].x + rePos[i][0] + dotArr[i].fat < wight \* 2 / 3 && dotArr[i].x + rePos[i][0] - dotArr[i].fat > 0). dotArr[i].x += rePos[i][0];

if (dotArr[i].y + rePos[i][1] + dotArr[i].fat < hight && dotArr[i].y + rePos[i][1] - dotArr[i].fat > 0). dotArr[i].y += rePos[i][1];

}

}

Секция кода, отвечающая за функцию расстановки вершин на графе. Используется планетарный физический смысл.

**index.html**

MacBook-Pro-Pavel:sem2\_ref pavelgamov$ cat index.html

<html>

<head>

</head>

<body>

<input type="file" id="selectFiles" value="Import" />

<button id="import">Import</button><br>

<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.7.2/p5.js"></script>

<script src="sketch.js"></script>

<script src="actions.js"></script>

<script src="buttons.js"></script>

<script src="lines.js"></script>

<script src="matrix.js"></script>

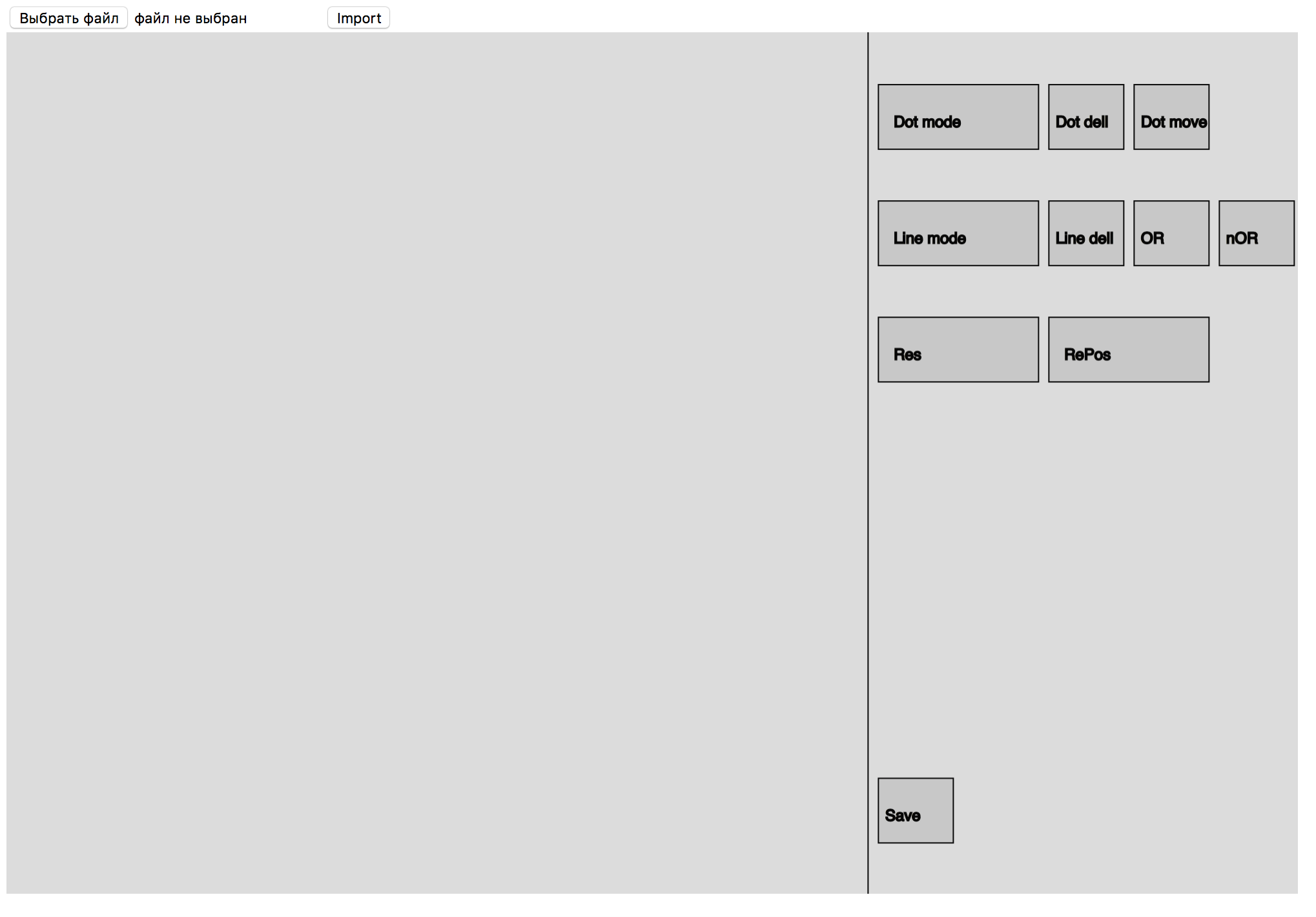
<script src="action.js"></script>

</body>

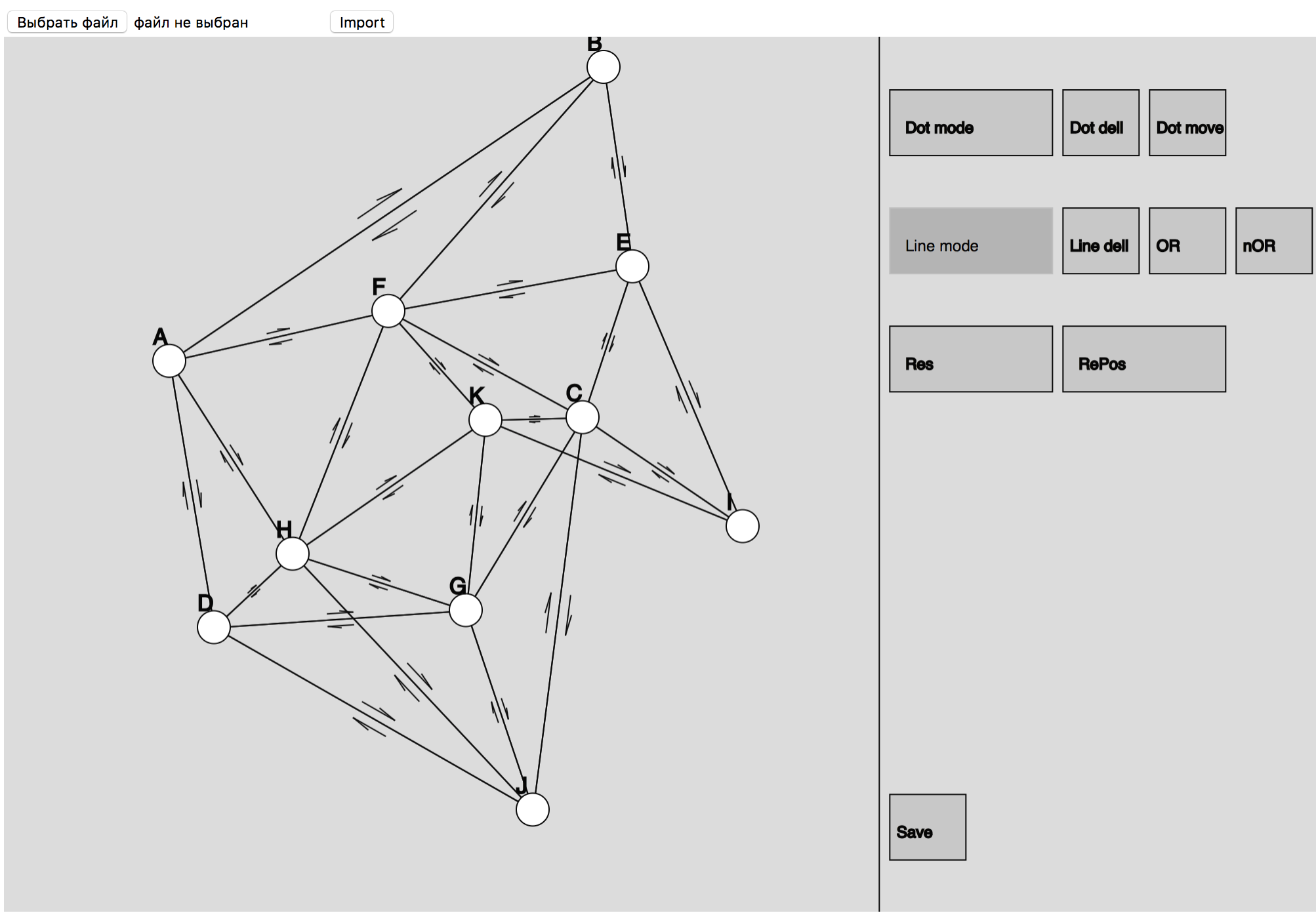
</html>

Наконец то язык разметки, head пустой, в body пишем все скрипты и размечаем кнопки с их id, для доступа к ним из скриптов и навешивания эвентов.

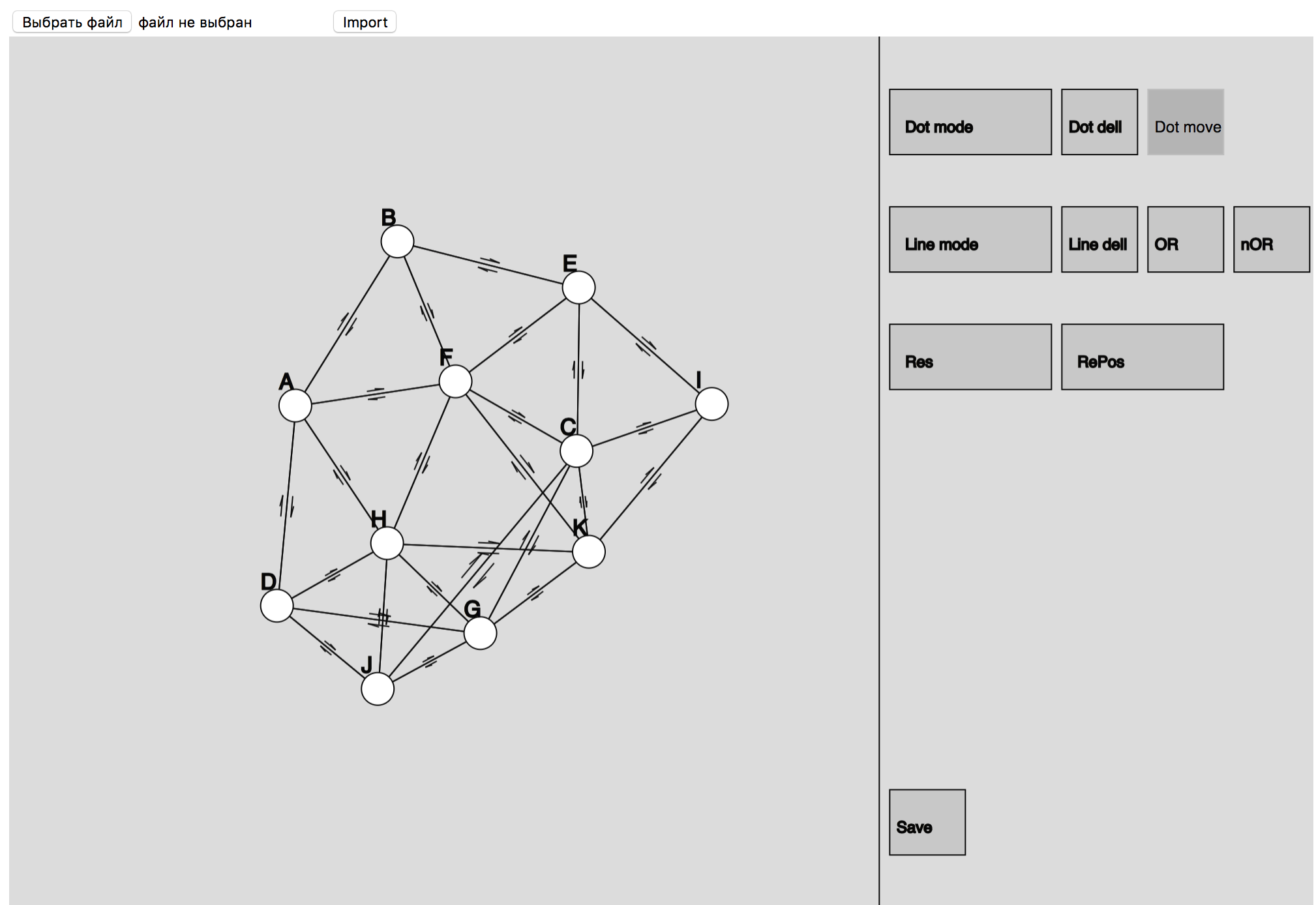
**Пример работы**



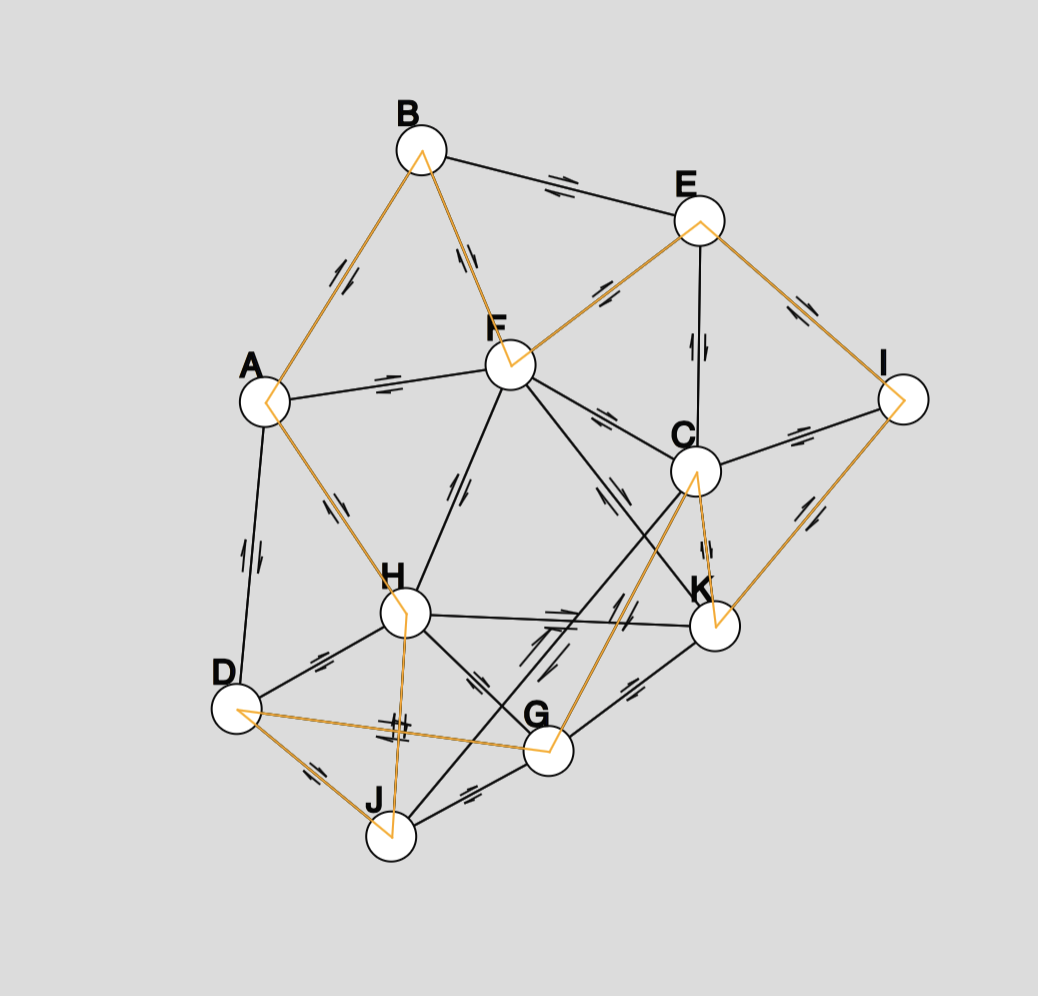
Так выглядит сайт при запуске. Вверху кнопки для загрузки используя чистый JavaScript, слева большую часть экрана занимает поле для отображения вершин и графа в целом, справа мы можем видеть кнопки отвечающие (дальше слева направо вниз) рисование вершин, удаление вершин, изменение положения вершин, создания линий, удаление линий, выбор между двунаправленным соединением вершин или однонаправленным, кнопкой получения файла всех Гамильтоновых путей на графе, кнопки разбиения и перемещения вершин графа в соответствии с их положением и связностью, ниже всех кнопка сохранения графа в файл json.



Введем произвольный граф, хотим узнать есть ли у него какие-либо Гамильтоновы пути. Для начала посмотрим, что сделает функция изменения положения.



Жмем функцию нахождения путей.



Программа покрасила нам Граф в соответствии с найденным Гамильтоновым контуром.

**Заключение**

Использование таких языков как JavaScript и возможности библиотек для визуализации событий, рисования, использования графики, позволяют более комфортно общаться с программой, позволяя пользоваться ее возможностями всем, кому захочется, ведь с интерфейсом каждый сможет понять функционал программы и научиться с ней работать.

Возможности языка JavaScript огромны, динамическое и объектное программирование позволяет быстро и эффективно писать как маленькие компактные скрипты, так и большие алгоритмы высокой сложности.

**Список использованной литературы**

Кофман А. – Введение в прикладную комбинаторику

Библиотека p5.js – <https://p5js.org/reference/>

Основы HTML – <https://developer.mozilla.org/ru/docs/Learn/HTML/Введение_в_HTML>

JavaScript – <https://learn.javascript.ru>

Теги HTML – <https://html5book.ru/html-tags/>