МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

|  |
| --- |
|  |

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК**

**Кафедра информационной безопасности**

**Интеллектуальные компьютерные системы**

**ОТЧЕТ**по лабораторной работе №1

на тему «**Алгоритм поиска решения с помощью оценочной функции**»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Выполнил студент гр. Б8116-09.03.01 | | |  | Гусев М.Д. | |  | | | Проверил | | |  | Ю.С. Москаленко | | дата | | |  | | | зачтено/не зачтено | | |

г. Владивосток

2019

**Постановка задачи и входные данные**

В качестве входных данных выступает две матрицы 3х3. Первая- начальная конфигурация, вторая – то, какой должна получиться итоговая конфигурация

Задача состоит в том, чтобы начальная матрица с помощью оценочной функции приняла вид итоговой матрицы.

**Реализация**

Изначально пользователю предоставляются поля, в которых можно задать все необходимые входные данные: начальную и конечную матрицы. Сделано это с помощью 3 стандартных форм ввода textarea.

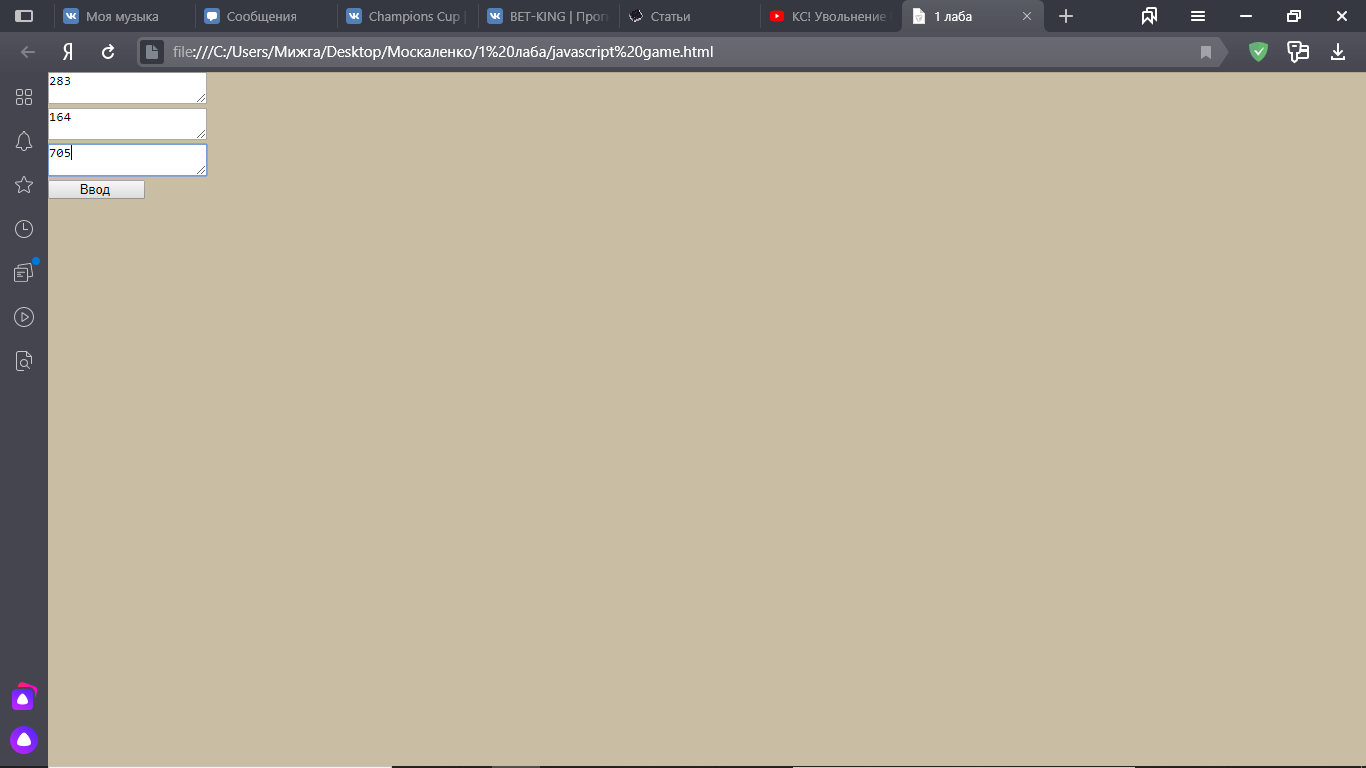


рис. 1 – ввод начальной конфигурации матрицы

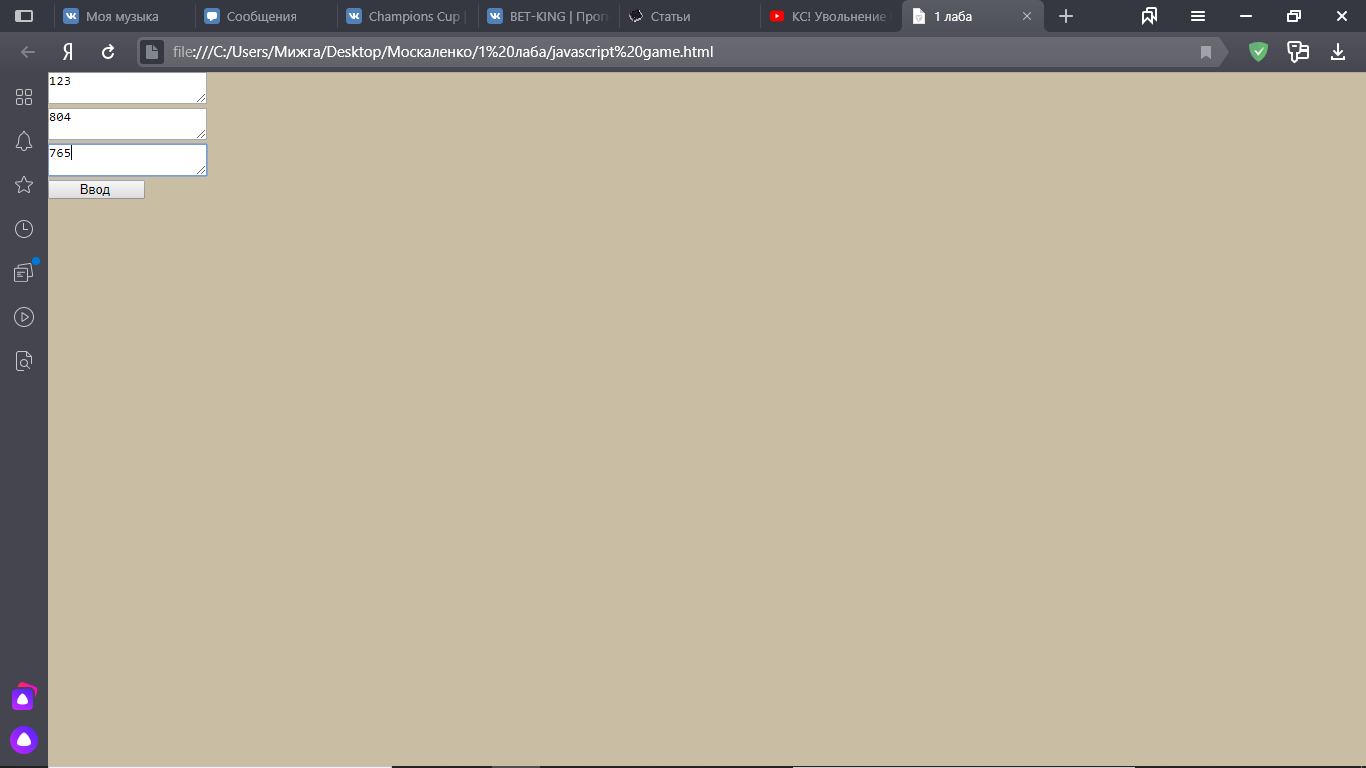


рис. 2 – ввод конечной конфигурации матрицы

После ввода данных начинается основная работа программы. Она состоит из четырех основных функций: функция для раскрытия первой вершины(корня дерева), функция для раскрытия ветки с минимальным количеством несовпадений, функция для раскрытия нескольких веток с минимальным количеством несовпадений. Все эти функции похожи по своей логике.

Первой начинает работу функция для раскрытия первой вершины. Она проверяет, является ли начальная конфигурация конечной(изначально верный вариант), если это не так, то вычисляет, на каком месте стоит ноль, в зависимости от этого добавляет нужное количество веток с новыми конфигурациями, запускает основную функцию работы, передавая в нее количество новых веток.

function vershina(){

count = 0;

for(var i =0;i<9;i++){

if(question[i] == answer[i]) count++;

copy\_question[i] = question[i];

}

count = 0;

if(count == 9)alert("Уже победа");

else if(question[0]==0){

tree[0] = [];tree[1] = [];

copy = question[0]; question[0] = question[1];question[1] = copy; alert(question[0]);

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[0]; question[0] = question[3]; question[3] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 2;

work();

return;

}

else if(question[1]==0){

tree[0] = [];tree[1] = [];tree[2] = [];

copy = question[1]; question[1] = question[0];question[0] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[1]; question[1] = question[2]; question[2] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[1]; question[1] = question[4]; question[4] = copy;

for(var i=0;i<9;i++){

tree[2][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 3;

work();

return;

}

else if(question[2]==0){

tree[0] = [];tree[1] = [];

copy = question[2]; question[2] = question[1]; question[1] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[2]; question[2] = question[5]; question[5] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 2;

work();

return;

}

else if(question[3]==0){

tree[0] = [];tree[1] = [];tree[2] = [];

copy = question[3]; question[3] = question[0]; question[0] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[3]; question[3] = question[4]; question[4] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[3]; question[3] = question[6]; question[6] = copy;

for(var i=0;i<9;i++){

tree[2][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 3;

work();

return;

}

else if(question[4]==0){

tree[0] = [];tree[1] = [];tree[2] = [];tree[3] = [];

copy = question[4]; question[4] = question[3]; question[3] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[4]; question[4] = question[1]; question[1] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[4]; question[4] = question[5]; question[5] = copy;

for(var i=0;i<9;i++){

tree[2][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[4]; question[4] = question[7]; question[7] = copy;

for(var i=0;i<9;i++){

tree[3][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 4;

work();

return;

}

else if(question[5]==0){

tree[0] = [];tree[1] = [];tree[2] = [];

copy = question[5]; question[5] = question[4]; question[4] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[5]; question[5] = question[2]; question[2] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[5]; question[5] = question[8]; question[8] = copy;

for(var i=0;i<9;i++){

tree[2][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 3;

work();

return;

}

else if(question[6]==0){

tree[0] = [];tree[1] = [];

copy = question[6]; question[6] = question[3]; question[3] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[6]; question[6] = question[7]; question[7] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 2;

work();

return;

}

else if(question[7]==0){

tree[0] = [];tree[1] = [];tree[2] = [];

copy = question[7]; question[7] = question[6]; question[6] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[7]; question[7] = question[4]; question[4] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[7]; question[7] = question[8]; question[8] = copy;

for(var i=0;i<9;i++){

tree[2][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 3;

work();

return;

}

else if(question[8]==0){

tree[0] = [];tree[1] = [];

copy = question[8]; question[8] = question[5]; question[5] = copy;

for(var i=0;i<9;i++){

tree[0][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

copy = question[8]; question[8] = question[7]; question[7] = copy;

for(var i=0;i<9;i++){

tree[1][i] = question[i];

}

for(var i=0;i<9;i++){

question[i] = copy\_question[i] ;

}

vetki = 2;

work();

return;

}

return;

}

Основная функция работы берет последние новые ветки (количество которых в нее передается с каждым новым ярусом) и сравнивает их между собой. Если выявлена единственная ветка с самым минимальным количеством несовпадений то запускается функция, которая очень похожа на функцию раскрытия корня дерева лишь с теми поправками, что эта функция анализирует, в какой ячейке находится ноль, на основе этого создает нужное количество веток, а затем проверяет, совпадают ли эти ветки с предыдущими ветками дерева. Если ветка совпадает, то она удаляется. Далее снова запускается основная функция работы, в которую также передается количество новых веток.

function where\_zero(){

if(tree[count][0]==0){

tree[count+1] = [];tree[count+2] = [];

tree.splice(count+3,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

}

tree[count+1][1] = tree[count][0]; tree[count+1][0] = tree[count][1];

tree[count+2][3] = tree[count][0]; tree[count+2][0] = tree[count][3];

vetki2();

return false;

}

else if(tree[count][1]==0){

tree[count+1] = [];tree[count+2] = [];tree[count+3] = [];

tree.splice(count+4,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

tree[count+3][i] = tree[count][i];

}

tree[count+1][0] = tree[count][1]; tree[count+1][1] = tree[count][0];

tree[count+2][2] = tree[count][1]; tree[count+2][1] = tree[count][2];

tree[count+3][4] = tree[count][1]; tree[count+3][1] = tree[count][4];

vetki3();

return false;

}

else if(tree[count][2]==0){

tree[count+1] = [];tree[count+2] = [];

tree.splice(count+3,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

}

tree[count+1][1] = tree[count][2]; tree[count+1][2] = tree[count][1];

tree[count+2][5] = tree[count][2]; tree[count+2][2] = tree[count][5];

vetki2();

return false;

}

else if(tree[count][3]==0){

tree[count+1] = [];tree[count+2] = [];tree[count+3] = [];

tree.splice(count+4,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

tree[count+3][i] = tree[count][i];

}

tree[count+1][0] = tree[count][3]; tree[count+1][3] = tree[count][0];

tree[count+2][4] = tree[count][3]; tree[count+2][3] = tree[count][4];

tree[count+3][6] = tree[count][3]; tree[count+3][3] = tree[count][6];

vetki3();

return false;

}

else if(tree[count][4]==0){

//alert('нолик в середине');

tree[count+1] = [];tree[count+2] = [];tree[count+3] = [];tree[count+4] = [];

tree.splice(count+5,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

tree[count+3][i] = tree[count][i];

tree[count+4][i] = tree[count][i];

}

tree[count+1][3] = tree[count][4]; tree[count+1][4] = tree[count][3];

tree[count+2][1] = tree[count][4]; tree[count+2][4] = tree[count][1];

tree[count+3][5] = tree[count][4]; tree[count+3][4] = tree[count][5];

tree[count+4][7] = tree[count][4]; tree[count+4][4] = tree[count][7];

vetki4();

return false;

}

else if(tree[count][5]==0){

tree[count+1] = [];tree[count+2] = [];tree[count+3] = [];

tree.splice(count+4,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

tree[count+3][i] = tree[count][i];

}

tree[count+1][4] = tree[count][5]; tree[count+1][5] = tree[count][4];

tree[count+2][2] = tree[count][5]; tree[count+2][5] = tree[count][2];

tree[count+3][8] = tree[count][5]; tree[count+3][5] = tree[count][8];

vetki3();

return false;

}

else if(tree[count][6]==0){

tree[count+1] = [];tree[count+2] = [];

tree.splice(count+3,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

}

tree[count+1][3] = tree[count][6]; tree[count+1][6] = tree[count][3];

tree[count+2][7] = tree[count][6]; tree[count+2][6] = tree[count][7];

vetki2();

return false;

}

else if(tree[count][7]==0){

tree[count+1] = [];tree[count+2] = [];tree[count+3] = [];

tree.splice(count+4,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

tree[count+3][i] = tree[count][i];

}

tree[count+1][6] = tree[count][7]; tree[count+1][7] = tree[count][6];

tree[count+2][4] = tree[count][7]; tree[count+2][7] = tree[count][4];

tree[count+3][8] = tree[count][7]; tree[count+3][7] = tree[count][8];

vetki3();

return false;

}

else if(tree[count][8]==0){

tree[count+1] = [];tree[count+2] = [];

tree.splice(count+3,10);

for(var i=0;i<9;i++){

tree[count+1][i] = tree[count][i];

tree[count+2][i] = tree[count][i];

}

tree[count+1][7] = tree[count][8]; tree[count+1][8] = tree[count][7];

tree[count+2][5] = tree[count][8]; tree[count+2][8] = tree[count][5];

vetki2();

return false;

}

}

function vetki2(){

ravno = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

for(var i=0;i<tree.length-2;i++){

ravno[0]=ravno[1]=0;

for(var j=0;j<9;j++){

if(tree[tree.length-2][j] == tree[i][j])ravno[0]++;

if(tree[tree.length-2][j] == question[j])ravno[1]++;

}

if(ravno[0]==9 || ravno[1]==9){

break;

}

}

for(var i=0;i<tree.length-2;i++){

ravno[2]=ravno[3]=0;

for(var j=0;j<9;j++){

if(tree[tree.length-1][j] == tree[i][j])ravno[2]++;

if(tree[tree.length-1][j] == question[j])ravno[3]++;

}

if(ravno[2]==9 || ravno[3]==9){

break;

}

}

if(ravno[0]==9 || ravno[1]==9){

if(ravno[2]==9 || ravno[3]==9){

tree.splice(tree.length-2,10);

vetki=0;

count=count+vetki;

work();

}

else if(ravno[0]==9 || ravno[1]==9){

for(var i=0;i<9;i++){

tree[tree.length-2][i] = tree[tree.length-1][i];

}

tree.splice(tree.length-1,10);

vetki=1;

count=count+vetki;

work();

}

}

else if(ravno[2]==9 || ravno[3]==9){

tree.splice(tree.length-1,10);

vetki=1;

count=count+vetki;

work();

}

else{

vetki = 2;

count=count+vetki;

work();

}

return;

}

Если же при работе основной функции будет выявлено, что минимальных веток несколько, то запускается еще одна функция, в которую передается количество минимальных веток. В этой функции сразу запускается цикл, который работает столько раз, сколько минимальных веток нужно раскрыть. Внутри цикла функция фактически является копией функции для раскрытия одной вершины, за тем лишь исключением, что здесь еще записывается количество одинаковых вершин для дальнейшего их вывода на экран.

function work\_ravno(){

vetki = 0;

for(var k=0;k<count\_vershin;k++){

count = tree.length-1;

(здесь идет фактически копия функции where\_zero(), приведенной выше)

}

}

Если текущая конфигурация совпадает с конечной, то запускается функция, которая выводит сначала длину полученного дерева без единицы, что соответствует количеству раскрытых вершин (без единицы потому, что последняя ветка уже не раскрывается), а затем и выводит их поярусно. Если на одном ярусе было раскрыто несколько вершин, то они также будут выводиться в одну строчку.

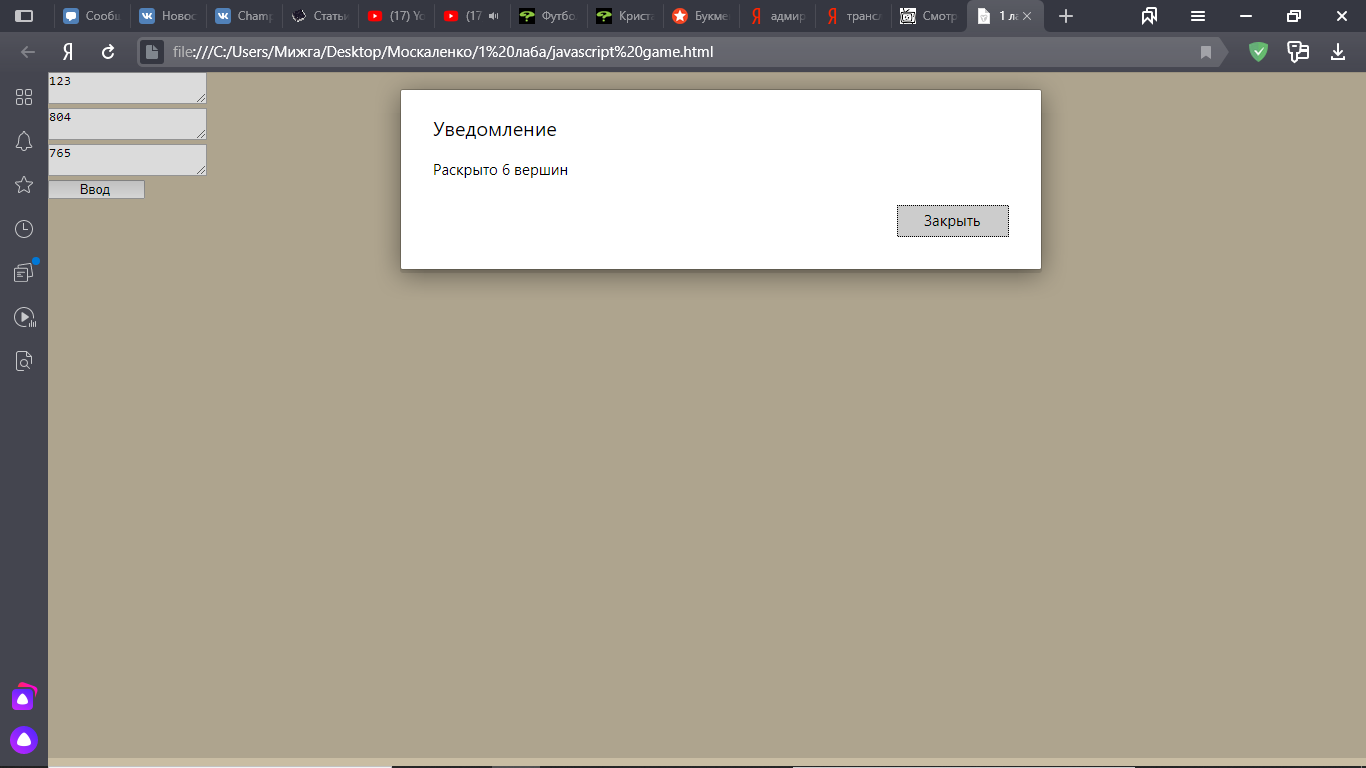
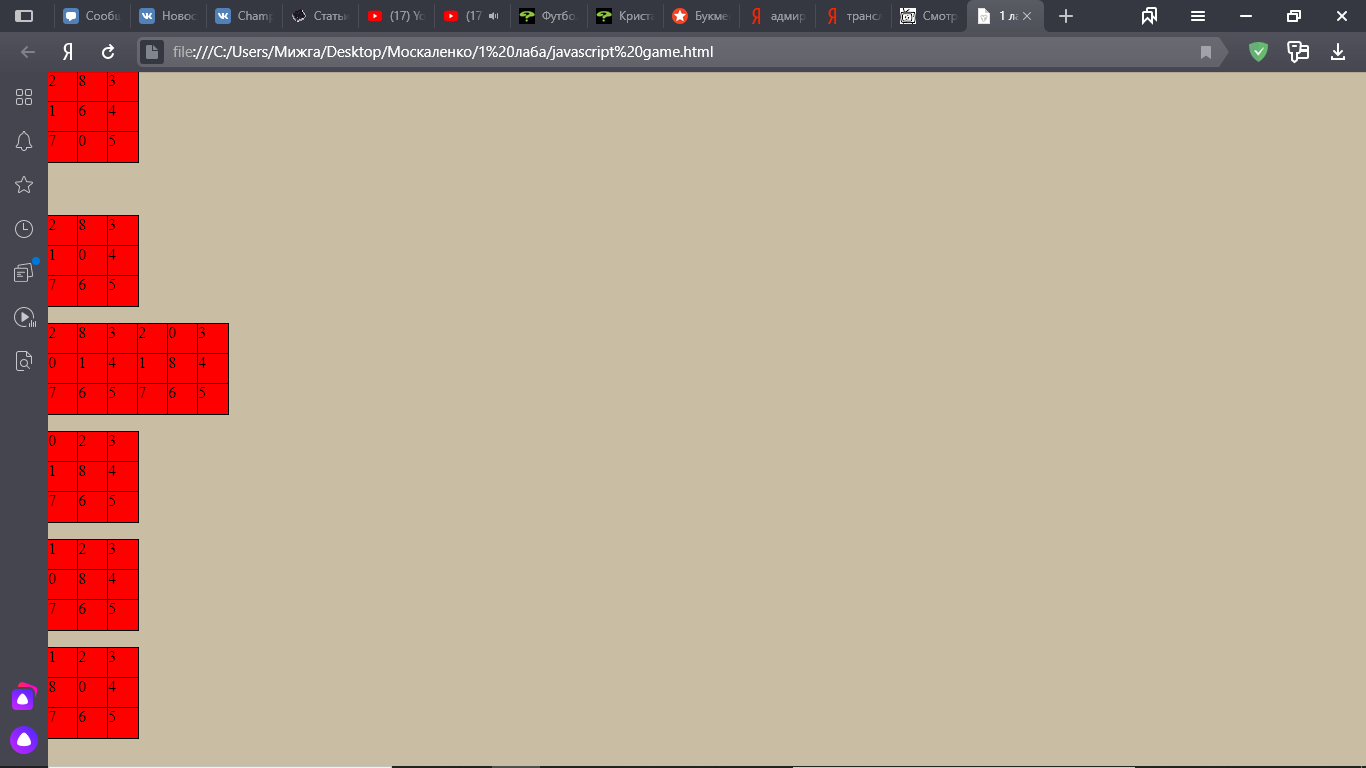
  
Рис.3 – отображение количества раскрытых вершин.  


Рис.4 – отображение всех вершин с минимальным количеством несовпадений с финальной конфигурацией на каждом ярусе.

Как видим, самая нижняя матрица совпадает с финальной конфигурацией, которая была задана в рис.2, а значит программа выполняет заданные функции.