МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №4

По дисциплине «Дискретная математика»

За 3 семестр

Тема: «Остовные деревья. Кодирование деревьев»

Выполнила:

студентка 2 курса

группы АС-56

Карпенко М.В.

Проверил:

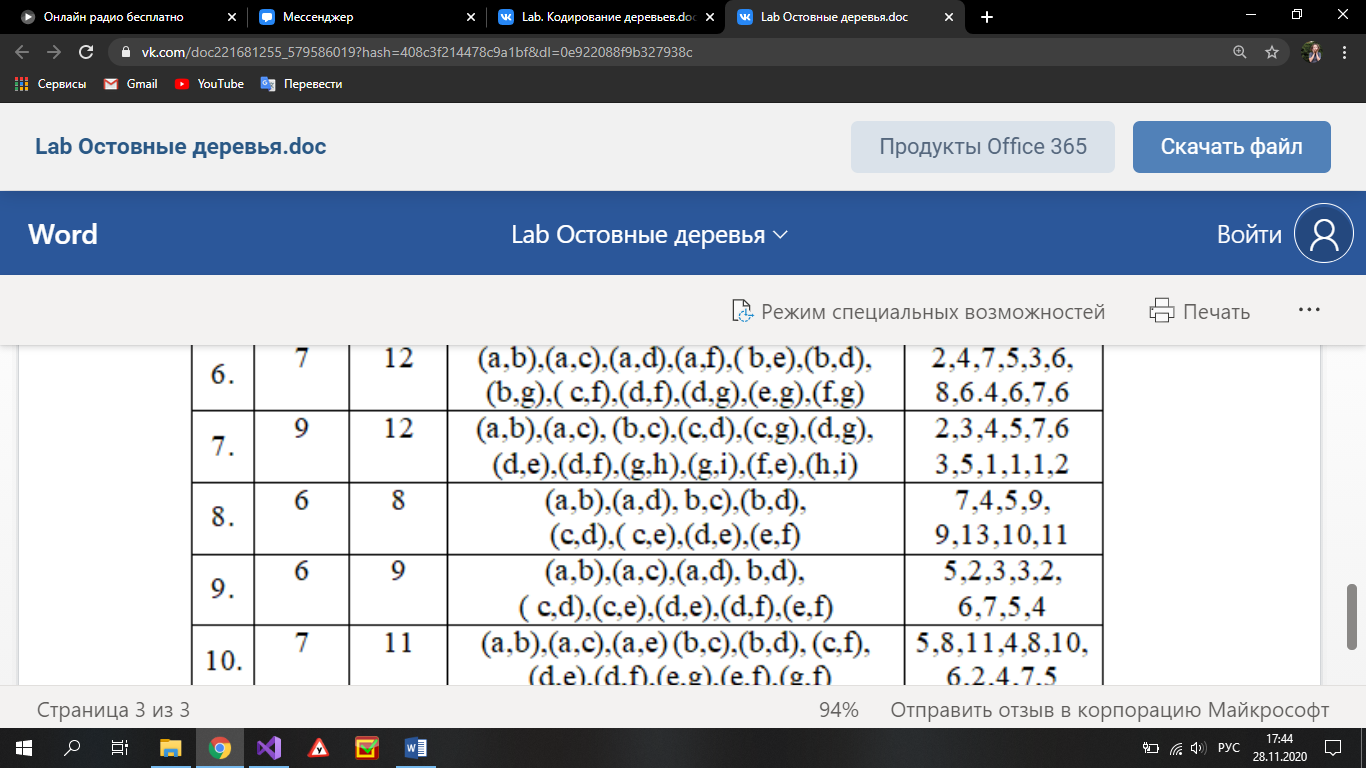
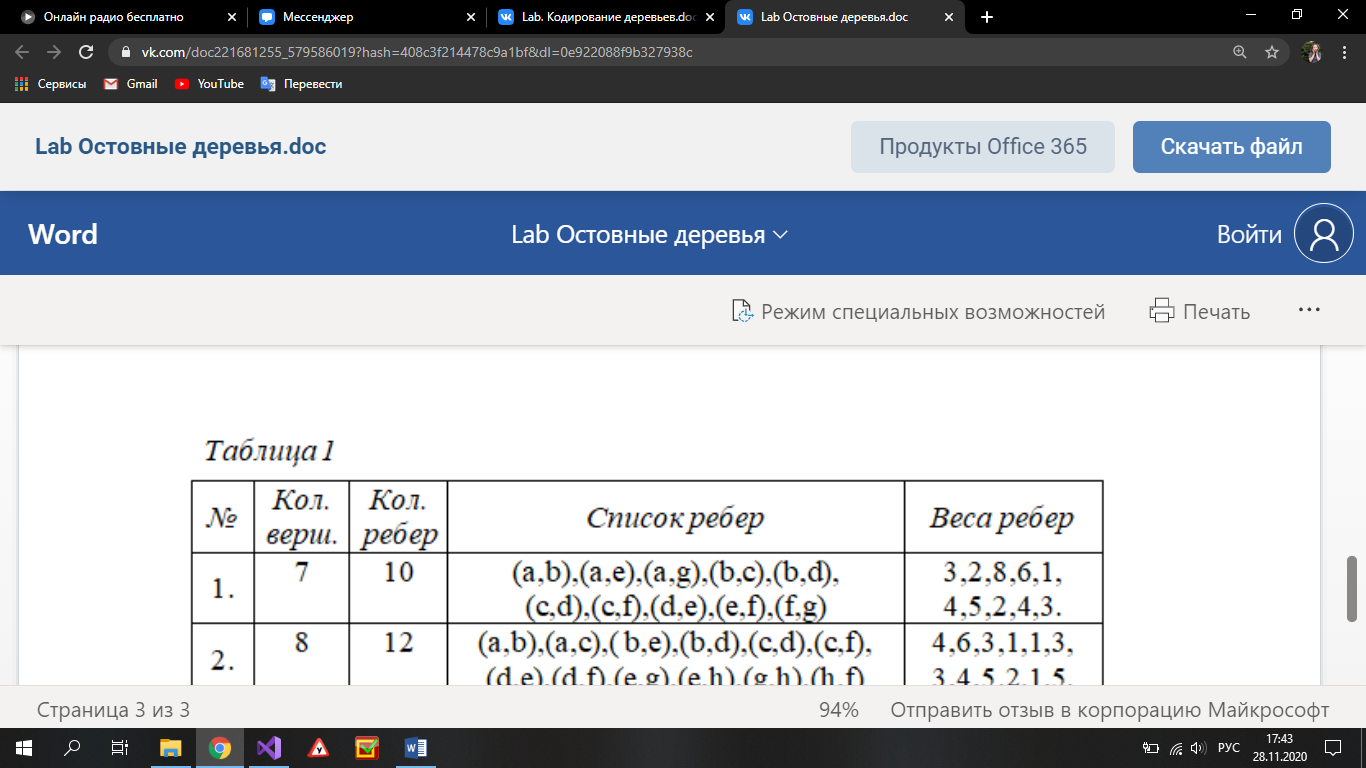
Глущенко Т.А.

Брест 2020

*Задание. Вариант 7*

1. Найти минимальное остовное дерево для заданного графа G алгоритмом Прима и Краскала. Варианты графов указаны в таблице 1. Граф задан списком ребер.

2. Ответить на поставленные вопросы.  
2.1. Для какого графа определяет число остовных деревьев формула Кэли.  
2.2. Подсчитать по формуле Кэли и нарисовать число остовных деревьев для n = 3.  
2.3. \* пункт 2 выполнить для n = 4.  
2.4. Какое остовное дерево находится алгоритмом Дейкстры?  
2.5. Может ли быть несколько минимальных остовых деревьев?

3. Графически изобразить граф и его минимальное остовное дерево.  


*Код программы:*

#include <iostream>

#include <Windows.h>

#define n 9

using namespace std;

void AdjMatr(int[12][2], int[12]);

void Prima(int\*\*);

void Kruskala(int[12][2], int[12]);

void Kirhgof(int\*\*);

int determ(int\*\*, int);

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int GraphEdges[12][2] = { { 1,2 },{ 1,3 },{ 2,3 },{ 3,4 },{ 3,7 },{ 4,7 },{ 4,5 },{ 4,6 },{ 7,8 },{ 7,9 },{6,5},{8,9} };

int Weight[12] = { 2,3,4,5,7,6,3,5,1,1,1,2 };

cout << endl;

AdjMatr(GraphEdges, Weight);

cout << endl;

system("pause");

return 0;

}

void AdjMatr(int GraphEdges[12][2], int Weight[12])

{

int\*\* AdjacencyMatrix;

AdjacencyMatrix = new int\* [9];

for (int h = 0; h < 9; h++)

{

AdjacencyMatrix[h] = new int[9];

}

for (int i = 0; i < 9; i++)

for (int j = 0; j < 9; j++)

AdjacencyMatrix[i][j] = 0;

for (int i = 0; i < 12; i++)

{

AdjacencyMatrix[GraphEdges[i][0] - 1][GraphEdges[i][1] - 1] = Weight[i];

AdjacencyMatrix[GraphEdges[i][1] - 1][GraphEdges[i][0] - 1] = Weight[i];

}

cout << " Матрица весов:" << endl;

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

cout << " " << AdjacencyMatrix[i][j];

cout << endl;

}

Kirhgof(AdjacencyMatrix); //Вычисление количества остовных деревьев

Prima(AdjacencyMatrix); //Вычисление минимального остовного дерева

cout << endl;

Kruskala(GraphEdges, Weight);//Вычисление минимального остовного дерева

for (int i = 0; i < 9; i++)

delete[] AdjacencyMatrix[i];

}

void Prima(int\*\* AdjacencyMatrix)

{

int ver[9] = { 0,0,0,0,0,0,0,0,0 };

int t = 0;

ver[0] = 1;

cout << "Минимальное остовное дерево по алгоритму Прима: " << endl;

for (int i = 1; i < 9; i++)

{

int Min = INT\_MAX;

int k = 0;

for (int h = 0; h < 9; h++)

{

if (ver[h] == 1)

{

for (int b = 0; b < 9; b++)

{

if (AdjacencyMatrix[h][b] < Min && AdjacencyMatrix[h][b] != 0 && ver[b] != 1)

{

Min = AdjacencyMatrix[h][b];

k = b;

t = h;

}

}

}

}

AdjacencyMatrix[t][k] = 0;

AdjacencyMatrix[k][t] = 0;

cout << t + 1 << "-" << k + 1 << "(" << Min << ")" << " ";

ver[k] = 1;

t = k;

}

cout << endl;

}

void Kruskala(int GraphEdges[12][2], int Weight[12])

{

int Wes[9];

for (int i = 0; i < 9; i++)

{

Wes[i] = i;

}

//Сортировка

for (int i = 0; i < 12; i++)

{

for (int j = i + 1; j < 12; j++)

{

if (Weight[i] > Weight[j])

{

int h;

h = Weight[i];

Weight[i] = Weight[j];

Weight[j] = h;

for (int k = 0; k < 2; ++k)

{

int g;

g = GraphEdges[i][k];

GraphEdges[i][k] = GraphEdges[j][k];

GraphEdges[j][k] = g;

}

}

}

}

cout << "Минимальное остовное дерево по алгоритму Крускала:" << endl;

for (int j = 0; j < 12; ++j)

{

if (Wes[GraphEdges[j][0] - 1] != Wes[GraphEdges[j][1] - 1])

{

cout << GraphEdges[j][0] << "-" << GraphEdges[j][1];

Wes[GraphEdges[j][1] - 1] = Wes[GraphEdges[j][0] - 1];

Wes[GraphEdges[j][0] - 1] = 0;

cout << "(" << Weight[j] << ") ";

}

}

}

void Kirhgof(int\*\* AdjacencyMatrix)

{

int\*\* M; //Матрица Кирхгофа

int h = 0;

M = new int\* [n];

for (int i = 0; i < n; i++)

{

M[i] = new int[n];

}

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

M[i][j] = AdjacencyMatrix[i][j];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (M[i][j] != 0)

{

h++;

M[i][j] = -1;

}

}

M[i][i] = h;

h = 0;

}

cout << " Матрица Кирхгофа:" << endl;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

cout << " " << M[i][j];

cout << endl;

}

//поиск алгебраического дополнения;

int\*\* A;

A = new int\* [n - 1];

for (int h = 0; h < n - 1; h++)

{

A[h] = new int[n - 1];

}

for (int i = 0; i < n - 1; i++)

for (int j = 0; j < n - 1; j++)

A[i][j] = M[i][j];

int res = 0;

res = determ(A, n - 1);

cout << endl << "Количество остовных деревьев в графе: " << res << endl << endl;

for (int i = 0; i < n - 1; i++)

delete[] A[i];

for (int i = 0; i < n; i++)

delete[] M[i];

}

int determ(int\*\* Arr, int size) //функция поиска определителя

{

int i, j;

double det = 0; //переменная определителя

int\*\* matr; //указатель

if (size == 1) // 1-е условие , размер 1

{

det = Arr[0][0];

}

else if (size == 2) // 2-е условие , размер 2

{

det = Arr[0][0] \* Arr[1][1] - Arr[0][1] \* Arr[1][0]; //

}

else

{

matr = new int\* [size - 1]; //создание динамического массива

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

{

for (j = 0; j < size - 1; ++j)

{

if (j < i)

{

matr[j] = Arr[j];

}

else

matr[j] = Arr[j + 1];

}

det += pow(-1, (i + j)) \* determ(matr, size - 1) \* Arr[i][size - 1]; //подсчеты

}

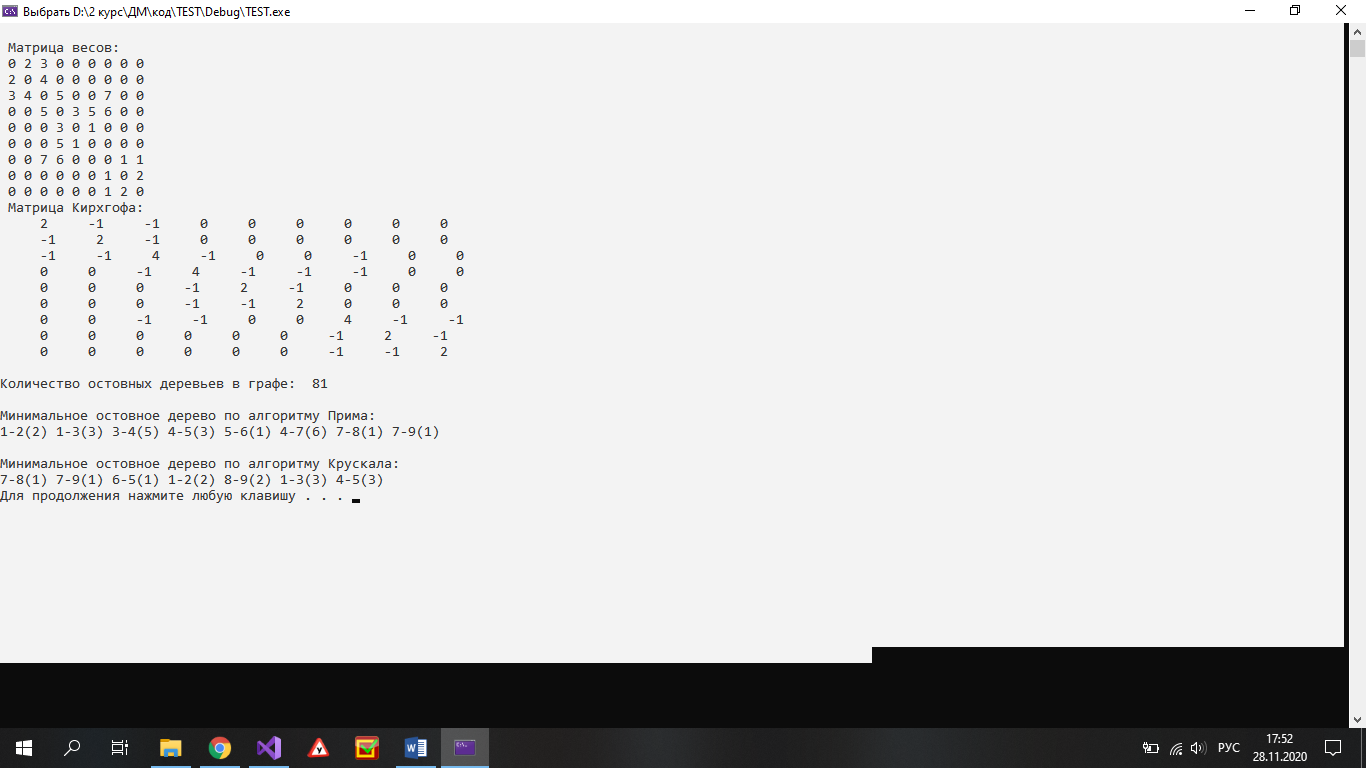
delete[] matr; //удаляем массив

}

return det; //возвращаем значение определителя

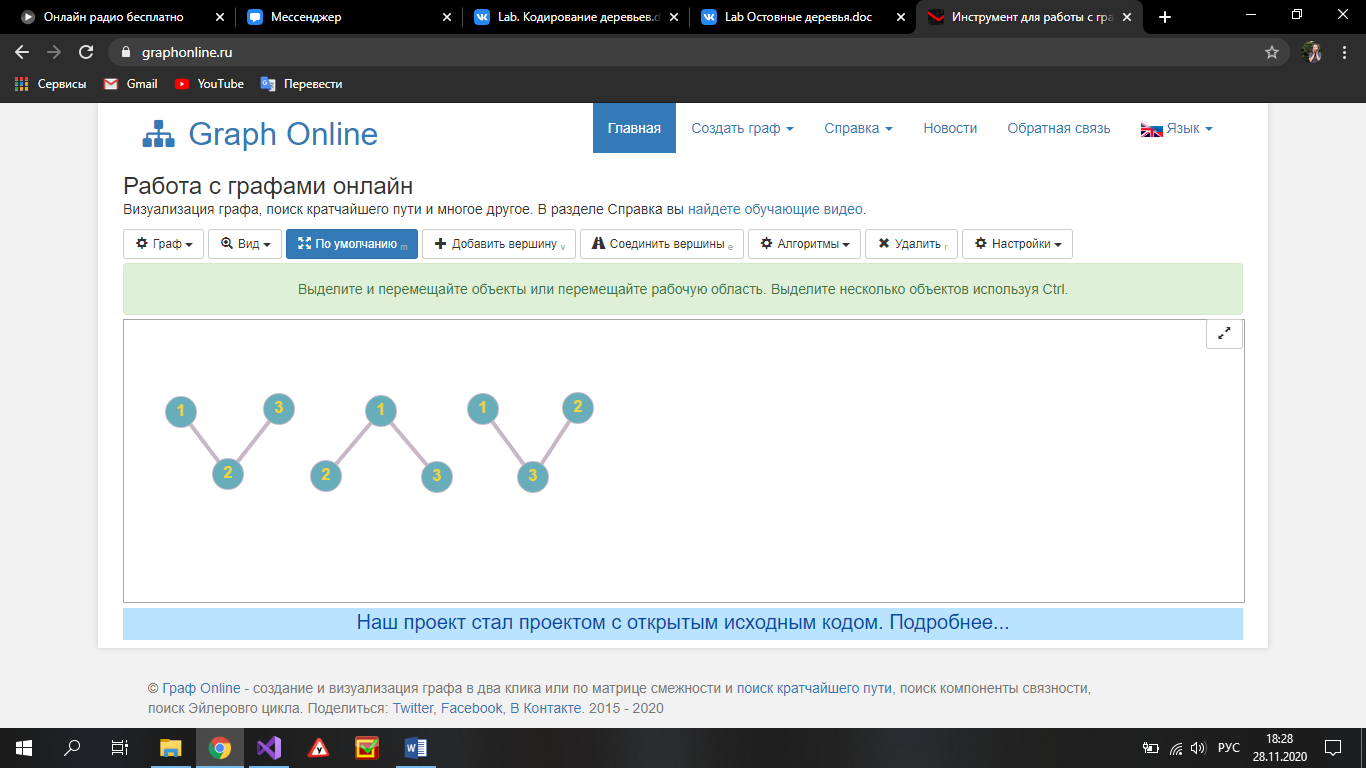
}

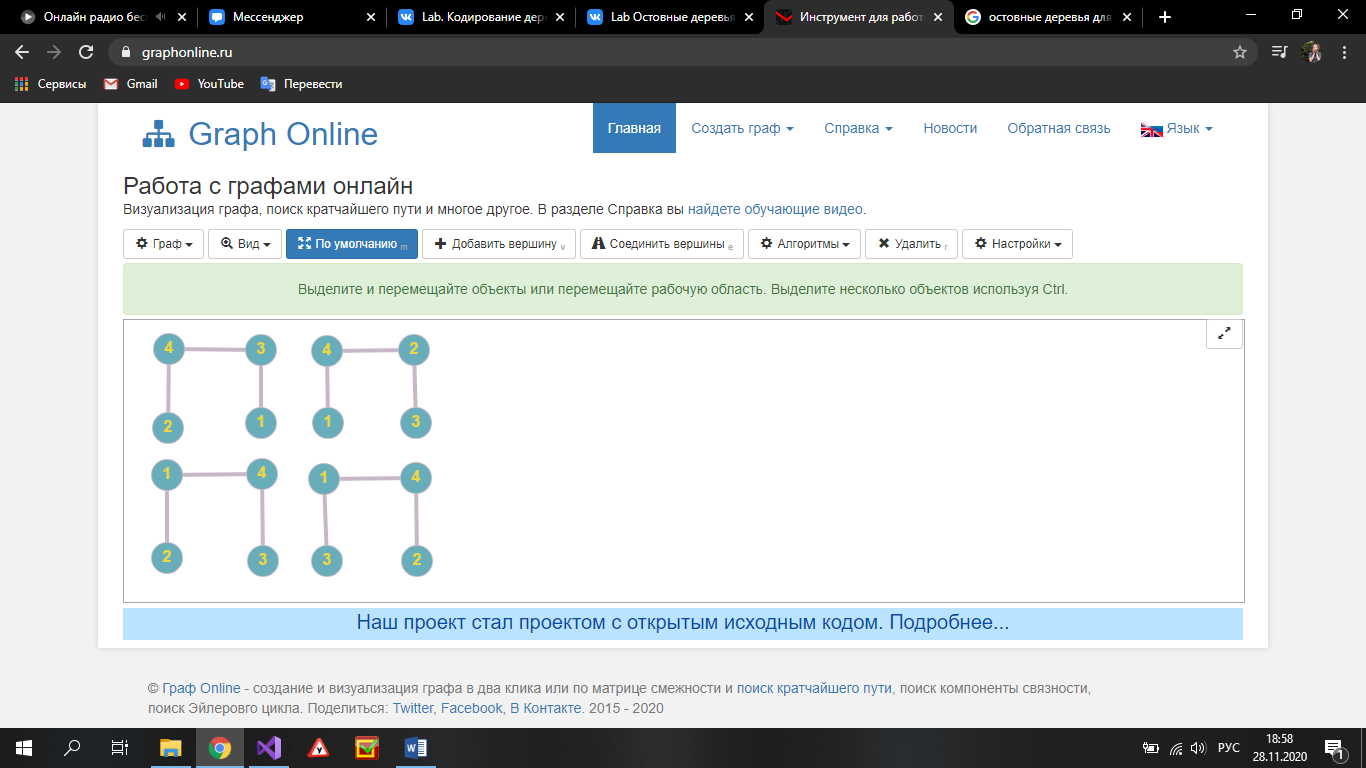
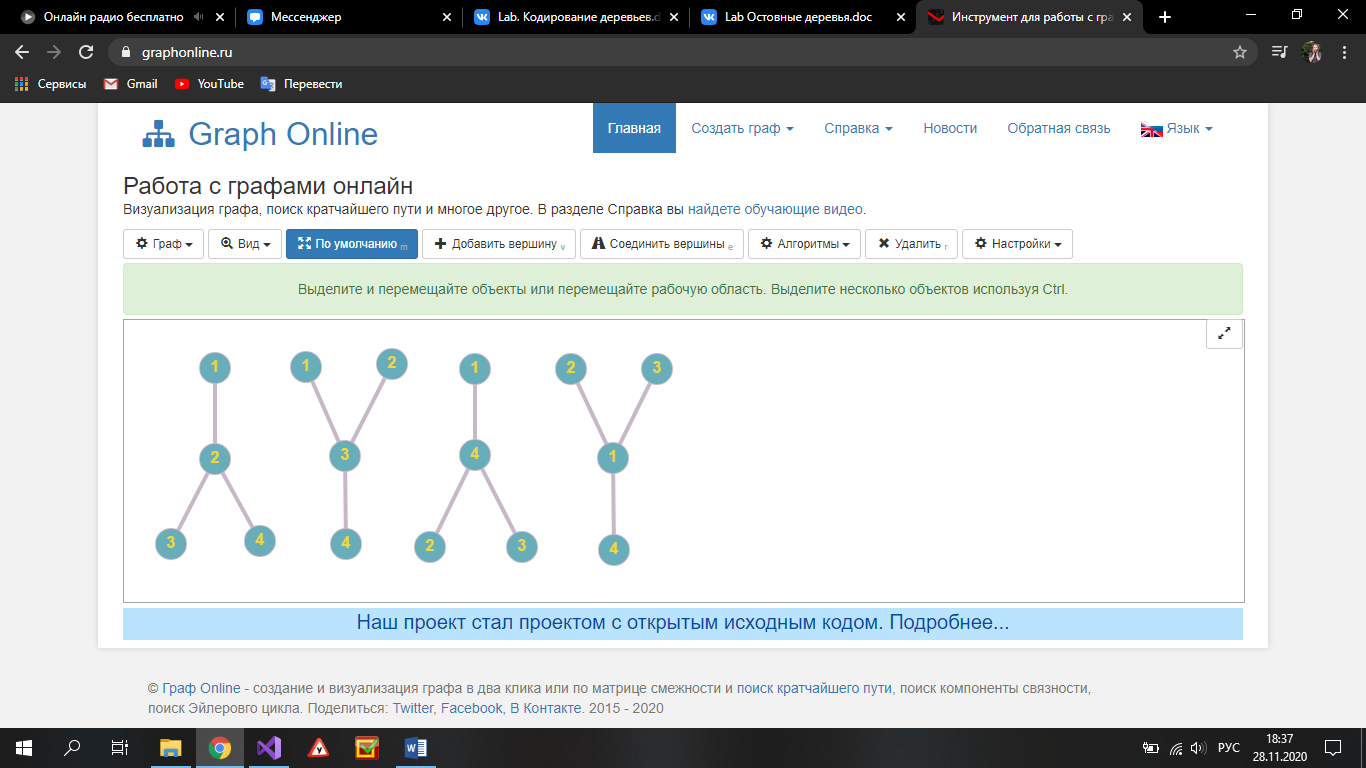
*Результат работы программы:*



Вопросы:

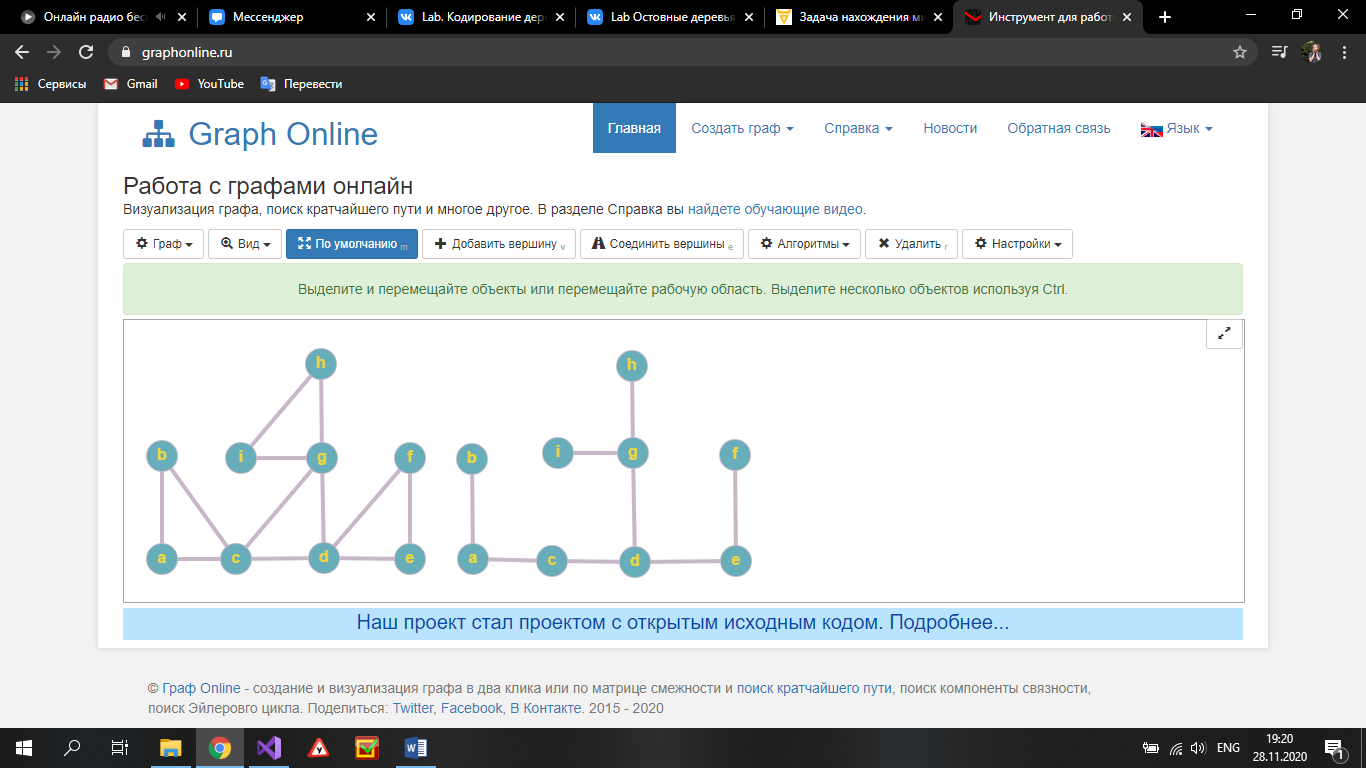
2.1. Для полного графа.

2.2. Формула Келли для n=3:   


2.3. Формула Келли для n=4:   


2.4. Минимальное остовное дерево.

2.5. Да, может.н

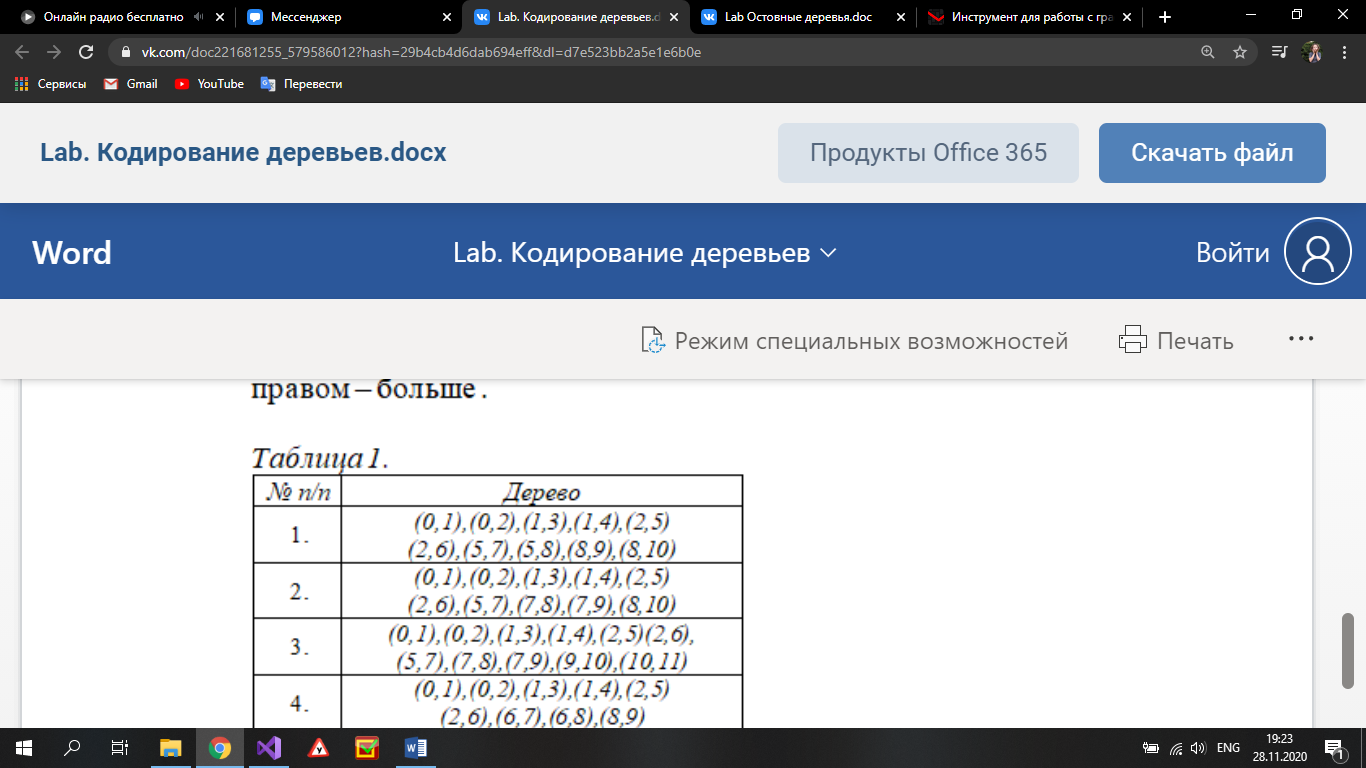
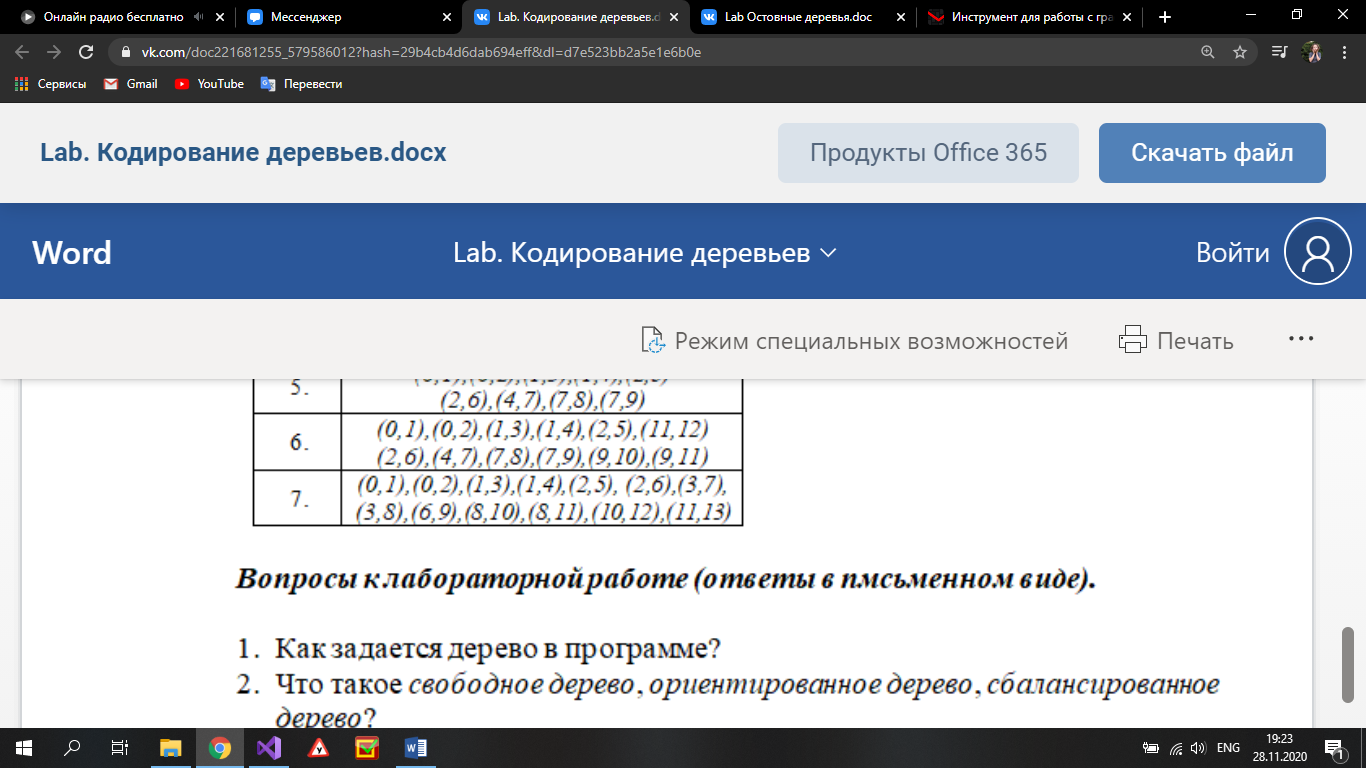
3. Графическое изображение графа и его минимального остового дерева.

*Задание. Вариант 7*

1. Написать программу кодировки/декодировки деревьев кодами Прюфера. Варианты деревьев указаны в таблице 1, деревья заданы списком ребер.

2. Графически изобразить дерево.

3. Рассмотреть указанное дерево как бинарное дерево с корнем в вершине 0. Найти его высоту, проверить является ли дерево сбалансированным. В случае отрицательного ответа, построить сбалансированное бинарное дерево поиска по заданному через таблицу списку вершин дерева. Корень в этом случае выбираем как середину всей последовательности ключей дерева, в левом поддереве все ключи меньше значения ключа корня, в правом – больше.

*Код программы:*

#include <iostream>

#include <Windows.h>

#include<vector>

#include <algorithm>

using namespace std;

void matrix\_creation(int arr[14][2], int matr[14][14])

{

for (int i = 0; i < 14; i++)

{

for (int j = 0; j < 14; j++)

{

for (int k = 0; k < 14; k++)

{

if (arr[k][0] == i && arr[k][1] == j)

{

matr[i][j] = 1;

break;

}

else

matr[i][j] = 0;

}

}

}

}

vector<int> prufer\_code(int arr[14][2])

{

int matr[14][14];

matrix\_creation(arr, matr);

vector<int> result;

bool t = true;

for (int i = 0; i < 14; i++)

{

t = true;

for (int j = 0; j < 14 && t; j++)

{

if (matr[i][j])

break;

if (j == 13)

{

matr[i][0] = 1;

for (int k = 0; k < 14; k++)

{

if (matr[k][i])

{

matr[k][i] = 0;

result.push\_back(k);

if (result.size() == 12)

{

return result;

}

i = 0;

t = false;

break;

}

}

}

}

}

cout << endl;

for (int i = 0; i < 14; i++)

{

for (int j = 0; j < 14; j++)

{

cout << matr[i][j] << " ";

}

cout << endl;

}

for (int i = 0; i < result.size(); i++)

{

cout << result[i] << " ";

}

cout << endl;

}

void decoding(int arr[12])

{

int all[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13 };

int temp;

bool t;

for (int j = 0; j < 12; j++)

{

cout << all[j] << " ";

}

cout << endl;

for (int i = 0; i < 12; i++)

{

for (int j = 0; j < 12; j++)

{

t = true;

for (int k = i; k < 12; k++)

{

if (all[j] == arr[k] && all[j] < 14 && arr[k] < 14)

break;

if (k == 11 && all[j] < 14 && arr[k] < 14)

{

temp = all[j];

all[j] = 22;

t = false;

}

}

if (!t)

break;

}

cout << "(" << arr[i] << "," << temp << "),";

}

cout << "(0,";

for (int j = 0; j < 12; j++)

{

if (all[j] != 22)

{

cout << all[j] << ")";

break;

}

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int Tree[13][2] = { {0,1},{0,2},{1,3},{1,4},{2,5}, {2,6},{3,7}, {3,8},{6,9},{8,10},{8,11},{10,12},{11,13} };  
 cout << "Prufer's code:" << endl;

vector<int> res = prufer\_code(Tree);

for (int i = 0; i < res.size(); i++)

{

cout << res[i] << " ";

}

cout << endl;

int arr[12];

for (int i = 0; i < res.size(); i++)

{

arr[i] = res[i];

}

cout << "Decoding:" << endl;

decoding(arr);

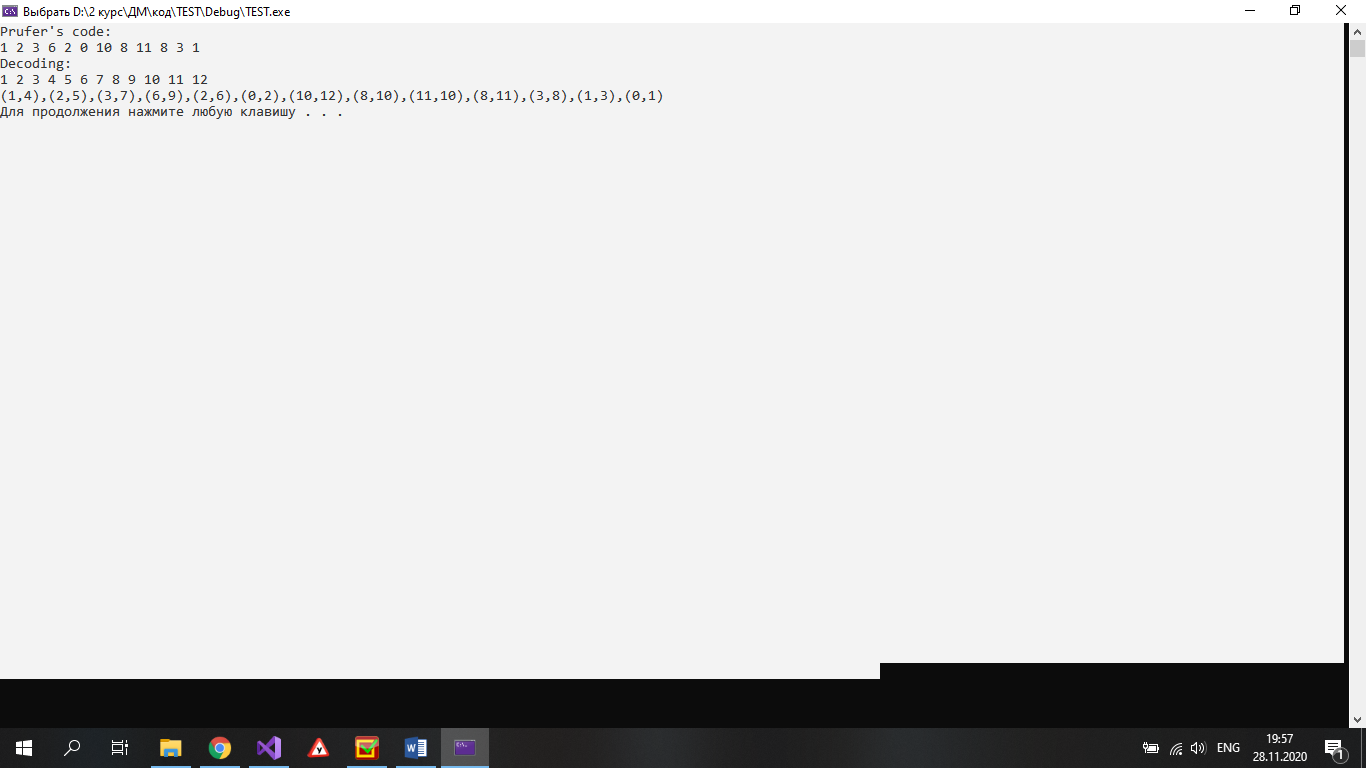
cout << endl;

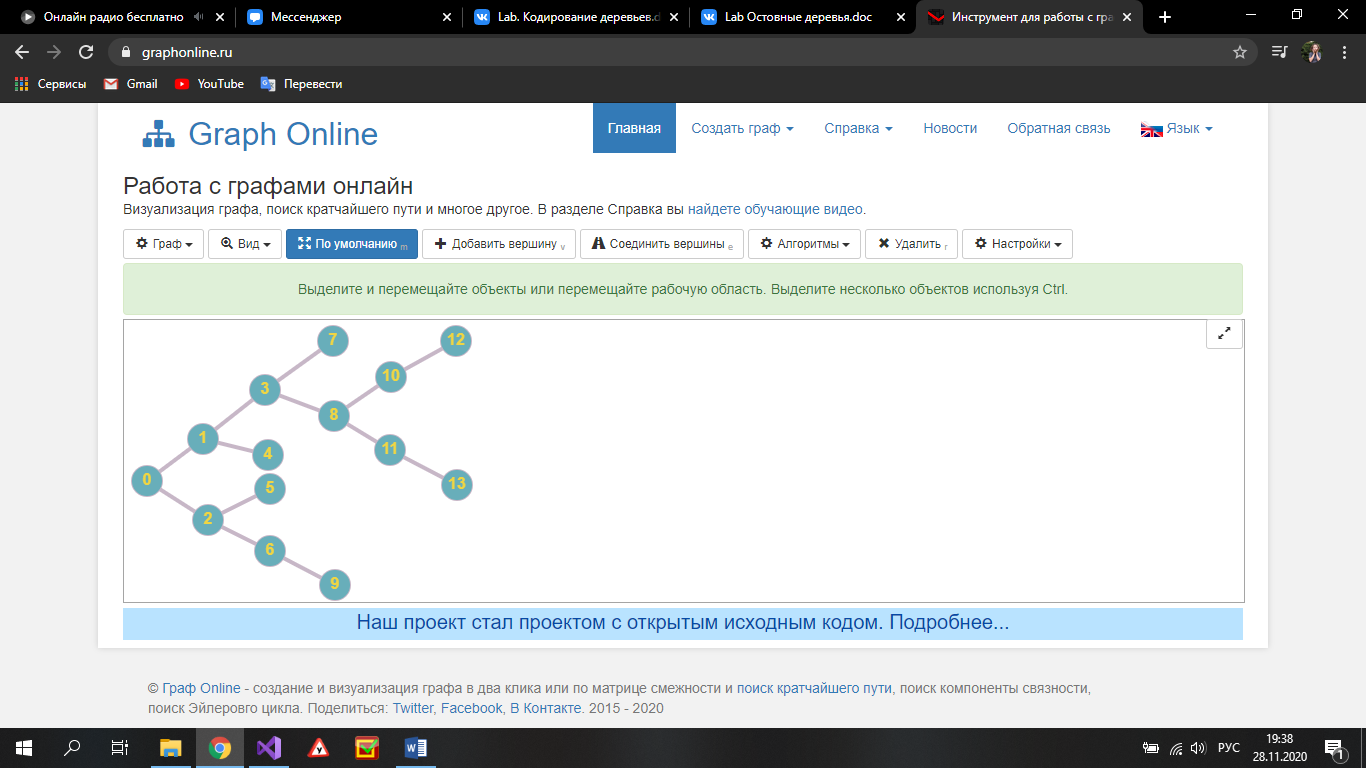
system("pause");

return 0;

}

*Результат работы программы:*



2. Графическое изображение дерева.  


Вопросы:

1. Как задается дерево в программе?  
Списком ребер через массив.

2. Что такое свободное дерево, ориентированное дерево, сбалансированное дерево?  
Свободное дерево - это связный неориентированный граф без циклов. Ориентированное дерево — ацикличный орграф (ориентированный граф, не содержащий циклов), в котором только одна вершина имеет нулевую степень захода (в неё не ведут дуги), а все остальные вершины имеют степень захода 1 (в них ведёт ровно по одной дуге). Сбалансированное дерево - дерево поиска, для каждой вершины которого высота её двух поддеревьев различается не более чем на 1.

3. Как определяется высота дерева, уровень вершины?  
Высотой вершины v  в дереве является длина наибольшего пути из вершины v до одного из его потомков. Уровнем вершины v является разность высоты дерева и глубины вершины v.

4. Для вашего дерева указать все порядки обхода дерева: прямой, обратный, симметричный, в глубину, в ширину.  
Прямой: 0, 2, 6, 9, 5, 1, 4, 3, 8, 11, 13, 7, 10, 12  
Обратный: 9, 5, 6, 2, 4, 13, 12, 7, 11, 10, 8, 3, 1, 0  
Симметричный: 9, 6, 2, 5, 0, 4, 1, 13, 11, 8, 10, 12, 3, 7  
В глубину: 0, 2, 6, 9, 5, 1, 4, 3, 8, 11, 13, 10, 12, 7  
В ширину: 0, 2, 1, 6, 5, 4, 3, 9, 8, 7, 11, 10, 13, 12