Додаток 1

**Main.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Course work</title>  
 <link rel="stylesheet" type="text/css" href="../css/main.css">  
 <script type="text/javascript" src="../js/vis.min.js"></script>  
 <link href="../css/vis.min.css" rel="stylesheet" type="text/css"/>  
  
 <script src="../js/main.js"></script>  
 <script src="../js/network.js"></script>  
 <script type="text/javascript" src="../js/algorithm.js"></script>  
 <script type="text/javascript" src="../js/graph.js"></script>  
 <script type="text/javascript" src="../js/messages.js"></script>  
  
  
</head>  
<body onload="draw();">  
  
<div id="main-container">  
 <div id="header">  
 <div id="random-menu" >  
 <h4>Set Network</h4>  
 <p>Error Probability</p> From<br><input id="rand-error-from" type="number" value="0.1"> <br>To<br><input id="rand-error-to"type="number" value="0.2"><br/>  
  
 <button id="generateRandom" onclick="randomGraphGeneration();">Set graph</button>  
 </div>  
  
 <div id="algorithm-menu">  
 <h4>Show the shortest path</h4>  
 From node<br><input id="alg-from" type="number"> <br>To node<br> <input id="alg-to"type="number" ><br/>  
  
 <button id="startAlg" onclick="processShortestPath();">Run</button>  
 </div>  
 <div id="messages-menu" >  
 <h4>Sending messages</h4>  
 <input type="radio" name="type" value="datagram"> Datagram<br>  
 <input type="radio" name="type" value="logic-link" checked> Logic link<br>  
 <p>Size of message (byte)</p> <input id="message-size" type="number" value="10000"><br/>  
 From node<br> <input id="send-from" type="number"> <br>To node<br> <input id="send-to"type="number"><br/>  
 <button id="processSending" onclick="send();">Send message</button>  
 <button id="showLog" onclick="showLog();">Show stacktrace</button>  
 </div>  
  
 </div>  
  
 <div id="main">  
 <div id="network-popUp">  
 <span id="operation">node</span> <br>  
 <table style="margin:auto;">  
 <tr class="node-prop">  
 <td>label</td>  
 <td>  
 <input id="node-label" value="new value" />  
 </td>  
 </tr>  
 <tr class="edge-prop">  
 <td>Satellite</td>  
 <td>  
 <input id="sat-inp" type="checkbox">  
 </td>  
 </tr>  
 <tr class="edge-prop">  
 <td>Error Probability</td>  
 <td>  
 <input id="error-probability">  
 </td>  
 </tr>  
 <tr class="edge-prop">  
 <td>Weight</td>  
 <td>  
 <input id="weight">  
 </td>  
 </tr>  
 <tr class="edge-prop-edit">  
 <td>Duplex</td>  
 <td>  
 <input id="isDupl" type="checkbox">  
 </td>  
 </tr>  
 <tr class="edge-prop-edit">  
 <td>Turned off</td>  
 <td>  
 <input id="isTurnedOff" type="checkbox">  
 </td>  
 </tr>  
 </table>  
 <button id="saveButton">Save</button>  
 <button id="cancelButton">Cancel</button>  
 </div>  
 <div id="mynetwork"></div>  
 <div id="info">  
 <h3 id="info-title"></h3>  
 <div id="info-body"></div>  
 </div>  
 </div>  
   
 <div id="footer">  
  
  
 </div>  
  
 <dialog id="log-info">  
 <div id="log-body"></div><br/>  
 <button id="hide-dialog" onclick="hideLog();">Close</button>  
 </dialog>  
  
</div>  
  
</body>  
  
  
</html>

Додаток 2

**Algorithm.js**

function processShortestPath() {  
 var from = document.getElementById('alg-from').value;  
 var to = document.getElementById('alg-to').value;  
 var shortestPath = getShortestPath(from, to);  
  
 console.log("My path: " + shortestPath);  
  
 nodes.forEach(function (n) {  
 getNetworkNodes().update({id: n.id, color: {background: "#97C2FC"}})  
 });  
  
 shortestPath.forEach(function (el) {  
 getNetworkNodes().update({id: el, color: {background: "#FF0000"}});  
 getNetworkEdges().update({id: nodes[el].c})  
 });  
}  
  
function getShortestPath(from, to) {  
 var nodes = getNodes();  
 var edges = getEdges();  
  
 for(var i = 0; i< edges.length; i++){  
 var e = edges[i];  
 nodes[e.from].children.push({p: e.to, w: e.weight, id: e.id});  
 nodes[e.to].children.push({p: e.from, w: e.weight, id: e.id});  
 }  
 var map = {};  
 for(var i = 0; i< nodes.length; i++){  
 var n = nodes[i];  
 path = {};  
 n.children.forEach(function (el) {  
 path[el.p] = el.w  
 });  
 map[n.id] = path;  
 }  
 var g = new Graph(map);  
 return g.findShortestPath(from,to);  
}

Додаток 3

**Graph.js**

var Graph = (function (undefined) {  
  
 var extractKeys = function (obj) {  
 var keys = [], key;  
 for (key in obj) {  
 Object.prototype.hasOwnProperty.call(obj,key) && keys.push(key);  
 }  
 return keys;  
 }  
  
 var sorter = function (a, b) {  
 return parseFloat (a) - parseFloat (b);  
 }  
  
 var findPaths = function (map, start, end, infinity) {  
 infinity = infinity || Infinity;  
  
 var costs = {},  
 open = {'0': [start]},  
 predecessors = {},  
 keys;  
  
 var addToOpen = function (cost, vertex) {  
 var key = "" + cost;  
 if (!open[key]) open[key] = [];  
 open[key].push(vertex);  
 }  
  
 costs[start] = 0;  
  
 while (open) {  
 if(!(keys = extractKeys(open)).length) break;  
  
 keys.sort(sorter);  
  
 var key = keys[0],  
 bucket = open[key],  
 node = bucket.shift(),  
 currentCost = parseFloat(key),  
 adjacentNodes = map[node] || {};  
  
 if (!bucket.length) delete open[key];  
  
 for (var vertex in adjacentNodes) {  
 if (Object.prototype.hasOwnProperty.call(adjacentNodes, vertex)) {  
 var cost = adjacentNodes[vertex],  
 totalCost = cost + currentCost,  
 vertexCost = costs[vertex];  
  
 if ((vertexCost === undefined) || (vertexCost > totalCost)) {  
 costs[vertex] = totalCost;  
 addToOpen(totalCost, vertex);  
 predecessors[vertex] = node;  
 }  
 }  
 }  
 }  
  
 if (costs[end] === undefined) {  
 return null;  
 } else {  
 return predecessors;  
 }  
  
 }  
  
 var extractShortest = function (predecessors, end) {  
 var nodes = [],  
 u = end;  
  
 while (u) {  
 nodes.push(u);  
 u = predecessors[u];  
 }  
  
 nodes.reverse();  
 return nodes;  
 }  
  
 var findShortestPath = function (map, nodes) {  
 var start = nodes.shift(),  
 end,  
 predecessors,  
 path = [],  
 shortest;  
  
 while (nodes.length) {  
 end = nodes.shift();  
 predecessors = findPaths(map, start, end);  
  
 if (predecessors) {  
 shortest = extractShortest(predecessors, end);  
 if (nodes.length) {  
 path.push.apply(path, shortest.slice(0, -1));  
 } else {  
 return path.concat(shortest);  
 }  
 } else {  
 return null;  
 }  
  
 start = end;  
 }  
 }  
  
 var toArray = function (list, offset) {  
 try {  
 return Array.prototype.slice.call(list, offset);  
 } catch (e) {  
 var a = [];  
 for (var i = offset || 0, l = list.length; i < l; ++i) {  
 a.push(list[i]);  
 }  
 return a;  
 }  
 }  
  
 var Graph = function (map) {  
 this.map = map;  
 }  
  
 Graph.prototype.findShortestPath = function (start, end) {  
 if (Object.prototype.toString.call(start) === '[object Array]') {  
 return findShortestPath(this.map, start);  
 } else if (arguments.length === 2) {  
 return findShortestPath(this.map, [start, end]);  
 } else {  
 return findShortestPath(this.map, toArray(arguments));  
 }  
 }  
  
 Graph.findShortestPath = function (map, start, end) {  
 if (Object.prototype.toString.call(start) === '[object Array]') {  
 return findShortestPath(map, start);  
 } else if (arguments.length === 3) {  
 return findShortestPath(map, [start, end]);  
 } else {  
 return findShortestPath(map, toArray(arguments, 1));  
 }  
 }  
  
 return Graph;  
  
})();

Додаток 4

**main.js**

var MIN\_NODES = 28;  
var REGIONS\_NUMBER = 4;  
var MIN\_NODES\_IN\_REGION = 7;  
var SATELLITE = 2;  
var AVERAGE\_RANG = 4;  
var WEIGHT\_ARRAY = [1, 2, 3, 4, 5, 7, 10, 12, 14, 17, 19, 21];  
  
var IS\_REGIONAL = true;

Додаток 5

**message.js**

var SERVICE\_PART\_SIZE = 128;  
var INFORM\_PART\_SIZE = 1500;  
var SERVICE\_PACKET = 128;  
var MAX\_CAPACITY = 20000;  
var STEP\_CAPACITY = 50;  
var MIN\_WEIGHT = 50;  
var packets;  
var datagramEdges = [];  
var packetsAndPath = [];  
var packetId = 0;  
var timeSum = [];  
  
var intensity = 0;  
var INTENSITY\_TIME = 1000;  
  
  
  
  
  
function send() {  
 var radios = document.getElementsByName('type');  
 var size = document.getElementById('message-size').value;  
 var count = 1;  
 var from = document.getElementById('send-from').value;  
 var to = document.getElementById('send-to').value;  
 document.getElementById("log-body").innerHTML = "";  
 timeSum = [];  
 packetId = 0;  
 intensity = 0;  
  
 if(radios[0].checked){  
 processDatagram(from, to, size, count);  
 }else{  
 processLogicLink(from, to, size, count);  
 }  
}  
  
  
function processDatagram(from, to, size, count) {  
 var paths = getPathsArray(from, to);  
 createPackets(size, count);  
 packetsAndPath = connectPacketsToPaths(paths);  
 datagramEdges = [];  
 packetsAndPath.forEach(function (r) {  
 var e = getEdgesForSending(r.path);  
 initEdges(e, r.packets.slice());  
 datagramEdges.push(e)  
 });  
 var bytesSum = 0;  
 for(var i = 0; i < packetsAndPath.length; i++){  
 setTimeoutFunction(i, bytesSum);  
 }  
  
 showFinishMessage(from, to, bytesSum, Math.round(timeSum.reduce(function(a, b) { return a + b; }, 0)/paths.length) );  
  
}  
  
function setTimeoutFunction(ind, bytesSum) {  
 var ed = datagramEdges[ind];  
 var path = packetsAndPath[ind].path;  
 bytesSum += process(path, ed, true);  
}  
  
function connectPacketsToPaths(paths) {  
 var res = [];  
 paths.forEach(function (p) {  
 res.push({  
 path: p,  
 packets: []  
 });  
 });  
 var countPerPath = Math.round(packets.length / paths.length);  
 var i = 0;  
 while (packets.length != 0){  
 res[i].packets.push(packets.pop());  
 i = ++i % paths.length;  
 }  
 return res;  
}  
  
function getPathsArray(from, to) {  
 var res = [];  
  
 while(true){  
 var shortestPath = getShortestPath(from, to);  
 var edges = getEdgesForSending(shortestPath);  
 if(isEnd(edges)){  
 break;  
 }  
 makeFakePath(edges);  
 res.push(shortestPath)  
 }  
 for(var i = 0; i< getEdges().length; i++){  
 turnONEdge(i)  
 }  
 return res;  
  
}  
function makeFakePath(edges) {  
 for(var i = 0; i< edges.length; i++){  
 turnOffEdge(edges[i].id)  
 }  
}  
function isEnd(edges) {  
 for(var i = 0; i< edges.length; i++){  
 if(edges[i].weight != getFakeWeight()){  
 return false;  
 }  
 }  
 return true;  
}  
  
function initEdges(edges, data) {  
 for(var i = 0; i< edges.length; i++){  
 var e = edges[i];  
 e.inQ = []; // Входящая в УЗЕЛ  
 e.outQ = []; // Исходящая из УЗЛА  
 e.packets = [];  
 e.deliveredTime = 0;  
 e.countPackets = 0;  
 e.workedTime = 0;  
 edges[i] = e;  
  
 }  
 edges[0].outQ = data;  
}  
  
function processLogicLink(from, to, size, count) {  
 var shortestPath = getShortestPath(from, to);  
 var edges = getEdgesForSending(shortestPath);  
  
 createServicePacket();  
 initEdges(edges, packets.slice());  
 var serviceSize = process(shortestPath, edges, false);  
  
  
 createPackets(size, count);  
 initEdges(edges, packets.slice());  
 var informSize = process(shortestPath, edges, false);  
  
  
 showFinishMessage(from, to, serviceSize+informSize, timeSum.reduce(function(a, b) { return a + b; }, 0));  
}  
  
function process(shortestPath, edges, isDatagram) {  
 var stopFlag = false;  
 var controlSum = edges[0].outQ.length;  
 var pack = edges[0].outQ.slice();  
  
 var startProcessTime = (new Date()).getTime();  
 while (!stopFlag){  
 for(var i = 0; i< shortestPath.length - 1; i++) {  
 var n = shortestPath[i];  
 var e = edges[i];  
  
 //Перенос из входящай очереди в узел в выходящую очередь в узел следующего канала  
 if (e.inQ.length != 0 && i != shortestPath.length - 2) {  
 var p = e.inQ.pop();  
 edges[i + 1].outQ.push(p);  
 showOutQMessage(p.info, p.service,shortestPath[i+1], p.id)  
  
 }  
  
 //Перенос из канала в входящую очередь следующего узла  
 if (e.packets.length != 0) {  
 if (e.deliveredTime <= (new Date()).getTime()) {  
 var p = e.packets.pop();  
 intensity++;  
 if (isError(e)) {  
 if(isDatagram){  
 showErrorMessage(p.id, "(Packet lost)")  
 controlSum--;  
 }else{  
 e.outQ.unshift(p);  
 showErrorMessage(p.id, "")  
 }  
  
 } else {  
 e.inQ.push(p);  
 showInQMessage(p.info, p.service,shortestPath[i], shortestPath[i+1], p.id);  
 if(i == shortestPath.length - 2){  
 var time = (new Date()).getTime() - p.startTime;  
 showFinishPacketMessage(p.id, time)  
 }  
 }  
 }  
 }  
  
 //Перенос из выходящей очереди в канал  
 if (e.outQ.length != 0) {  
 if (e.packets.length == 0) {  
 var p = e.outQ.pop();  
 if(p.startTime == 0){  
 p.startTime = (new Date()).getTime();  
 }  
 var sat = e.sat\_inp ? 3 : 1;  
 var capacity = (MAX\_CAPACITY - (e.weight - MIN\_WEIGHT) \* STEP\_CAPACITY) \* sat;  
 var extraTime = Math.round((p.info + p.service) / capacity \* 1000);  
 e.packets.push(p);  
 e.deliveredTime = (new Date()).getTime() + extraTime;  
 showInChannelMessage(p.info, p.service, shortestPath[i+1], p.id);  
  
  
 }  
 }  
 }  
  
 if(edges[edges.length-1].inQ.length == controlSum){  
 stopFlag = true;  
 }  
 }  
  
 edges[edges.length-1].inQ = [];  
 var processTime = (new Date()).getTime() - startProcessTime;  
 timeSum.push(processTime);  
 return getByteCount(pack)  
}  
  
function getByteCount(pack) {  
 var sum = 0;  
 pack.forEach(function (p) {  
 sum += p.info + p.service;  
 });  
 return sum;  
}  
  
function createPackets(size, count) {  
 packets = [];  
 var allSize = size \* count;  
 var countPackets = Math.round(allSize / INFORM\_PART\_SIZE);  
 var lastPacketSize = allSize % INFORM\_PART\_SIZE;  
 for(var i = 0; i < countPackets; i++){  
 packets.push({  
 id: packetId++,  
 info: INFORM\_PART\_SIZE,  
 service: SERVICE\_PART\_SIZE,  
 startTime: 0  
 });  
 }  
 if(lastPacketSize != 0){  
 packets.push({  
 id: packetId++,  
 info: lastPacketSize,  
 service: SERVICE\_PART\_SIZE,  
 startTime: 0  
 });  
 }  
}  
function createServicePacket() {  
 packets = [];  
 packets.push({  
 id: packetId++,  
 info: 0,  
 service: SERVICE\_PACKET,  
 startTime: 0  
 });  
}  
  
function findEdge(to, children) {  
 for(var i = 0; i < children.length; i++){  
 var e = children[i];  
 if(e.p == to){  
 return JSON.parse(JSON.stringify(getEdges()[e.id]));  
 }  
 }  
}  
  
  
function getEdgesForSending(shortestPath) {  
 var edges = [];  
 for(var i = 0; i < shortestPath.length - 1; i++){  
 var from = shortestPath[i];  
 var to = shortestPath[i+1];  
 edges.push(findEdge(to, getNodes()[from].children))  
 }  
 return edges.slice();  
}  
  
function isError(e) {  
 var error = e.error;  
 var random = getRandomFloat(0, 1);  
 return error > random;  
}  
  
function turnOffEdge(edgeId) {  
 var e = getEdges()[edgeId];  
 e.weight = getFakeWeight();  
 getEdges()[edgeId] = e;  
}  
  
function turnONEdge(edgeId) {  
 var e = getEdges()[edgeId];  
 e.weight = e.saved\_weight;  
 getEdges()[edgeId] = e;  
}  
  
  
function showInQMessage(info, service, nodeIdTo, nodeIdFrom, id) {  
 document.getElementById("log-body").innerHTML += "<p class='info'>Packet "+ id +" which contains "+service+" bytes of service part"+  
 " and "+info+" bytes of info part has been put in input queue of "+nodeIdFrom+" node</p>"  
}  
function showOutQMessage(info, service, nodeId, id) {  
 document.getElementById("log-body").innerHTML += "<p class='info'>Packet "+ id +" which contains "+service+" bytes of service part"+  
 " and "+info+" bytes of info part has been put in out queue of "+nodeId+" node</p>"  
}  
function showInChannelMessage(info, service, nodeId, id) {  
 document.getElementById("log-body").innerHTML += "<p class='yellow'>Packet "+ id +" which contains "+service+" bytes of service part"+  
 " and "+info+" bytes of info part is in channel to "+nodeId+" node</p>"  
}  
function showLog() {  
 document.getElementById("log-info").showModal()  
}  
function hideLog() {  
 document.getElementById("log-info").close()  
}  
  
  
function showFinishPacketMessage(id, time) {  
 document.getElementById("log-body").innerHTML += "<p class='green'>Packet "+ id +" has sent ("+ time +"ms)</p>"  
}  
  
function showErrorMessage(id, message) {  
 document.getElementById("log-body").innerHTML += "<p class='red'>Error occurred in packet "+id+" "+message+"</p>"  
}  
  
function showFinishMessage(idFrom, idTo, byteCount, time) {  
 document.getElementById("log-body").innerHTML += "<h4 class='green'>Message of "+ byteCount +"bytes has been sent from "+ idFrom +" node to "+ idTo +" node("+ time +"ms)</h4>";  
 document.getElementById("log-body").innerHTML += "<h4 class='green'>Average intensity: "+intensity/(time/1000)+" packets/second</h4>"  
  
}

Додаток 6

**network.js**

var nodes = null;  
var edges = null;  
var network = null;  
var FAKE\_WEIGHT = 99999;  
  
var networkDataNodes = null;  
var networkDataEdges = null;  
  
var newNodeInd;  
var newEdgeInd;  
var regionNodeId = 0;  
  
var options = {  
 physics:{  
 enabled: false  
 },  
 interaction:{  
 hover:true,  
 selectConnectedEdges: false  
 },  
 manipulation: {  
 addNode: function (data, callback) {  
 // filling in the popup DOM elements  
 console.log("addNode: " + JSON.stringify(data));  
 document.getElementById('operation').innerHTML = "Add Node";  
 document.getElementById('node-label').value = newNodeInd;  
 document.getElementById('saveButton').onclick = saveNode.bind(this, data, callback);  
 document.getElementById('cancelButton').onclick = clearPopUp.bind();  
 document.getElementById('network-popUp').style.display = 'block';  
 },  
 editNode: function (data, callback) {  
 console.log("editNode: " + JSON.stringify(data));  
 // filling in the popup DOM elements  
 document.getElementById('operation').innerHTML = "Edit Node";  
 document.getElementById('node-label').value = data.label;  
 document.getElementById('saveButton').onclick = saveNode.bind(this, data, callback);  
 document.getElementById('cancelButton').onclick = cancelEdit.bind(this,callback);  
 document.getElementById('network-popUp').style.display = 'block';  
 },  
 addEdge: function (data, callback) {  
 console.log("addEdge: " + JSON.stringify(data));  
 if (data.from != data.to) {  
 var props = document.getElementsByClassName("node-prop");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'none'  
 }  
 props = document.getElementsByClassName("edge-prop");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'initial'  
 }  
 props = document.getElementsByClassName("edge-prop-edit");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'initial'  
 }  
 document.getElementById('network-popUp').style.display = 'block';  
 document.getElementById('operation').innerHTML = "Add Edge";  
 document.getElementById('saveButton').onclick = saveEdge.bind(this, data, callback);  
 document.getElementById('cancelButton').onclick = clearPopUp.bind();  
 }  
  
 },  
 editEdge: function (data, callback) {  
 document.getElementById('operation').innerHTML = "Edit Edge";  
 var e = edges[data.id];  
 document.getElementById('isDupl').checked = e.isDupl;  
  
 var props = document.getElementsByClassName("node-prop");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'none'  
 }  
 props = document.getElementsByClassName("edge-prop");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'none'  
 }  
 props = document.getElementsByClassName("edge-prop-edit");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'initial'  
 }  
 document.getElementById('saveButton').onclick = editEdge.bind(this, data, callback);  
 document.getElementById('cancelButton').onclick = clearPopUp.bind();  
 document.getElementById('network-popUp').style.display = 'block';  
  
 }  
 }  
};  
  
  
  
  
  
function draw() {  
 // create an array with nodes  
 nodes = [];  
 // create an array with edges  
 edges = [];  
  
  
 var container = document.getElementById('mynetwork');  
 // create a network  
  
 createNetwork(container);  
}  
  
function createNetwork(container) {  
 networkDataNodes = new vis.DataSet(nodes);  
 networkDataEdges = new vis.DataSet(edges)  
 var data = {  
 nodes: networkDataNodes,  
 edges: networkDataEdges  
  
 };  
  
 destroy();  
  
 network = new vis.Network(container, data, options);  
  
 network.on("selectNode", function (params) {  
 console.log(JSON.stringify(params));  
 showNodeInfo(params);  
 });  
 network.on("selectEdge", function (params) {  
 console.log(JSON.stringify(params));  
 showEdgeInfo(params);  
 });  
  
}  
  
function correctData(d) {  
 d.nodes.forEach(function (n) {  
 n.id = regionNodeId++;  
 })  
 d.edges.forEach(function (e) {  
 e.to = d.nodes[e.to].id  
 e.from = d.nodes[e.from].id  
 })  
}  
function makeRouting() {  
 for(var i = 1; i< nodes.length; i++){  
 var nFrom = nodes[i];  
 for(var j = 1; j < nodes.length; j++){  
 if(i==j){  
 continue;  
 }  
 var nTo = nodes[j];  
 var path = getShortestPath(nFrom.id, nTo.id);  
 var eds = getEdgesForSending(path)  
 var sum = 0;  
 eds.forEach(function (e) {  
 sum += e.weight;  
 });  
 nFrom.routing.push({  
 to: nTo.id,  
 path: path,  
 weight: sum  
 });  
 }  
 }  
}  
function randomGraphGeneration() {  
 newNodeInd = 0;  
 newEdgeInd = 0;  
 regionNodeId = 0;  
 var errorFrom= document.getElementById('rand-error-from').value;  
 var errorTo= document.getElementById('rand-error-to').value;  
 var randParam = {  
 errorFrom: errorFrom,  
 errorTo: errorTo  
 };  
 if(IS\_REGIONAL){  
 var datas = [];  
 for(var i = 0; i < REGIONS\_NUMBER; i++){  
 var d = getRandomGraph(MIN\_NODES\_IN\_REGION, randParam);  
 correctData(d);  
 datas.push(d)  
 }  
 var n = {  
 id: newNodeInd,  
 label: String(newNodeInd),  
 children: [],  
 routing: []  
 };  
 newNodeInd++;  
  
 for(var i = 0; i < datas.length; i++){  
 var d = datas[i];  
 var from = n.id;  
 var to = d.nodes[getRandomInt(0, MIN\_NODES\_IN\_REGION -1 )].id;  
 // var w = getRandomInt(randParam.weightFrom, randParam.weightTo);  
 var w = WEIGHT\_ARRAY[getRandomInt(0, WEIGHT\_ARRAY.length -1)];  
 var error = getRandomFloat(randParam.errorFrom, randParam.errorTo);  
 var e = {  
 id: newEdgeInd,  
 from: from,  
 to: to,  
 weight: w,  
 error: error,  
 sat\_inp: true,  
 isDupl: isDupl(),  
 label: w,  
 font: {align: 'top'},  
 saved\_weight: w  
 };  
 newEdgeInd++;  
 datas[datas.length-1].edges.push(e)  
 }  
 datas[REGIONS\_NUMBER-1].nodes.push(n);  
 var data = {  
 edges: [],  
 nodes: []  
 }  
 datas.forEach(function (d) {  
 d.nodes.forEach(function (n) {  
 data.nodes.push(n);  
 });  
 d.edges.forEach(function (e) {  
 data.edges.push(e);  
 });  
 });  
  
 }else{  
 var data = getRandomGraph(MIN\_NODES, randParam);  
 }  
  
 edges = data.edges;  
 nodes = data.nodes;  
 makeRouting();  
 var container = document.getElementById('mynetwork');  
 createNetwork(container);  
}  
function getRandomGraph(minNodes, randParam) {  
 var data = getScaleFreeNetwork(minNodes, randParam);  
 for(var i = data.edges.length; i < minNodes \* AVERAGE\_RANG - data.edges.length; i++){  
 var from = 0;  
 var to = 0;  
 do {  
 from = getRandomInt(0, minNodes-1);  
 to = getRandomInt(0, minNodes-1);  
 }while(from == to || isEdgeExist(from, to, data.edges));  
  
  
  
 // var w = getRandomInt(randParam.weightFrom, randParam.weightTo);  
 var w = WEIGHT\_ARRAY[getRandomInt(0, WEIGHT\_ARRAY.length -1)];  
 var error = getRandomFloat(randParam.errorFrom, randParam.errorTo);  
 data.edges.push({  
 id: newEdgeInd,  
 from: from,  
 to: to,  
 weight: w,  
 error: error,  
 sat\_inp: isSat(),  
 isDupl: isDupl(),  
 label: w,  
 font: {align: 'top'},  
 saved\_weight: w  
 });  
 newEdgeInd++;  
 }  
 return data;  
}  
function isEdgeExist(from, to, edges) {  
 for(var i = 0; i< edges.length; i++){  
 var e = edges[i];  
 if(e.to == to && e.from == from || e.to == from && e.from == to){  
 return true;  
 }  
 }  
 return false;  
}  
  
function isSat() {  
 return false;  
}  
  
function getRandomInt(min, max) {  
 min = Math.ceil(Number(min));  
 max = Math.floor(Number(max));  
 return Math.floor(Math.random() \* (max - min + 1)) + min;  
}  
function getRandomFloat(min, max) {  
 return Math.random() \* (Number(max) - Number(min)) + Number(min)  
}  
  
  
function getScaleFreeNetwork(nodeCount, randParam) {  
 var nodes = [];  
 var edges = [];  
 var connectionCount = [];  
  
 // randomly create some nodes and edges  
 for (var i = 0; i < nodeCount; i++) {  
 nodes.push({  
 id: newNodeInd,  
 label: String(newNodeInd),  
 children: [],  
 routing: []  
 });  
 newNodeInd++;  
  
 connectionCount[i] = 0;  
  
 // create edges in a scale-free-network way  
 if (i == 1) {  
 var from = i;  
 var to = 0;  
 // var w = getRandomInt(randParam.weightFrom, randParam.weightTo);  
 var w = WEIGHT\_ARRAY[getRandomInt(0, WEIGHT\_ARRAY.length -1)];  
  
 var error = getRandomFloat(randParam.errorFrom, randParam.errorTo);  
 edges.push({  
 id: newEdgeInd,  
 from: from,  
 to: to,  
 weight: w,  
 error: error,  
 sat\_inp: isSat(),  
 isDupl: isDupl(),  
 label: w,  
 font: {align: 'top'},  
 saved\_weight: w  
 });  
 newEdgeInd++;  
 connectionCount[from]++;  
 connectionCount[to]++;  
 }  
 else if (i > 1) {  
 var conn = edges.length \* 2;  
 var rand = Math.floor(Math.random() \* conn);  
 var cum = 0;  
 var j = 0;  
 while (j < connectionCount.length && cum < rand) {  
 cum += connectionCount[j];  
 j++;  
 }  
  
  
 var from = i;  
 var to = j;  
 // var w = getRandomInt(randParam.weightFrom, randParam.weightTo);  
 var w = WEIGHT\_ARRAY[getRandomInt(0, WEIGHT\_ARRAY.length -1)];  
 var error = getRandomFloat(randParam.errorFrom, randParam.errorTo);  
 edges.push({  
 id: newEdgeInd,  
 from: from,  
 to: to,  
 weight: w,  
 error: error,  
 sat\_inp: isSat(),  
 isDupl: isDupl(),  
 label: w,  
 font: {align: 'top'},  
 saved\_weight: w  
 });  
 newEdgeInd++;  
 connectionCount[from]++;  
 connectionCount[to]++;  
 }  
 }  
  
 return {nodes:nodes, edges:edges};  
  
}  
  
function isDupl() {  
 return false;  
}  
  
function clearPopUp() {  
 document.getElementById('saveButton').onclick = null;  
 document.getElementById('cancelButton').onclick = null;  
 document.getElementById('network-popUp').style.display = 'none';  
 var props = document.getElementsByClassName("node-prop");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'initial'  
 }  
 props = document.getElementsByClassName("edge-prop");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'none'  
 }  
 props = document.getElementsByClassName("edge-prop-edit");  
 for(var i = 0; i< props.length; i++){  
 props[i].style.display = 'none'  
 }  
  
}  
  
function cancelEdit(callback) {  
 clearPopUp();  
 callback(null);  
}  
  
function saveNode(data,callback) {  
 var id = newNodeInd;  
 var label = document.getElementById('node-label').value;  
 data.id = id;  
 data.label = label;  
 newNodeInd++;  
 nodes.push({  
 id: id,  
 label: label,  
 children: [],  
 routing: []  
 });  
 // networkDataNodes.add({id: id, label: label});  
 clearPopUp();  
 callback(data);  
}  
  
function saveEdge(data,callback) {  
 var weight = document.getElementById('weight').value;  
 var error = document.getElementById('error-probability').value;  
 var sat\_inp = document.getElementById('sat-inp').checked;  
 var dup = document.getElementById('isDupl').checked;  
 data.id = newEdgeInd;  
 data.label = weight;  
 data.font = {align: 'top'};  
  
 edges.push({  
 id: data.id,  
 from: data.from,  
 to: data.to,  
 weight: weight,  
 error: error,  
 sat\_inp: sat\_inp,  
 isDupl: dup,  
 label: weight,  
 font: {align: 'top'},  
 saved\_weight: weight  
 });  
 newEdgeInd++;  
 // networkDataEdges.add({id: data.id, from: data.from, to: data.to});  
 console.log("My out: " + JSON.stringify(edges));  
 clearPopUp();  
 callback(data);  
}  
  
function editEdge(data,callback) {  
 var weight = edges[data.id].saved\_weight;  
 var error = edges[data.id].error;  
 var sat\_inp = edges[data.id].sat\_inp;  
 var dup = document.getElementById('isDupl').checked;  
  
 var isTurnedOff = document.getElementById('isTurnedOff').checked;  
  
  
 edges[data.id] = {  
 id: data.id,  
 from: data.from,  
 to: data.to,  
 weight: isTurnedOff?FAKE\_WEIGHT:weight,  
 error: error,  
 sat\_inp: sat\_inp,  
 isDupl: dup,  
 label: weight,  
 font: {align: 'top'},  
 saved\_weight: weight  
 };  
 if(isTurnedOff){  
 data.label = "";  
 }else{  
 data.label = weight;  
 }  
 // networkDataEdges.add({id: data.id, from: data.from, to: data.to});  
 console.log("My out: " + JSON.stringify(edges));  
 clearPopUp();  
 callback(data);  
}  
  
function destroy() {  
 if (network !== null) {  
 network.destroy();  
 network = null;  
 }  
}  
  
function showNodeInfo(params) {  
 var nodeInd = params.nodes[0];  
 var node = nodes[nodeInd];  
  
 document.getElementById('info-title').innerHTML = "About node";  
 var info = document.getElementById('info-body');  
 info.innerHTML = "";  
  
 p = document.createElement('p');  
 p.innerHTML = "Label: " + node.label;  
 info.appendChild(p);  
  
 p = document.createElement('h5');  
 p.innerHTML = "Routing: ";  
 info.appendChild(p);  
  
 node.routing.forEach(function (el) {  
 p = document.createElement('p');  
 p.innerHTML = "" + node.id + "&#8594;" + el.to + " : " + JSON.stringify(el.path) + " Minimal weight:&#8594;" + el.weight;  
 info.appendChild(p);  
 });  
  
 document.getElementById('info').style.display = 'block';  
 console.log("Children: " + JSON.stringify(nodes[nodeInd]))  
}  
  
function showEdgeInfo(params) {  
 var edgeInd = params.edges[0];  
 var edge = edges[edgeInd];  
  
 document.getElementById('info-title').innerHTML = "About edge";  
 var info = document.getElementById('info-body');  
 info.innerHTML = "";  
 var p = document.createElement('p');  
 p.innerHTML = "ID: " + edge.id;  
 info.appendChild(p);  
  
 p = document.createElement('p');  
 p.innerHTML = "From: " + edge.from;  
 info.appendChild(p);  
  
 p = document.createElement('p');  
 p.innerHTML = "To: " + edge.to;  
 info.appendChild(p);  
  
 p = document.createElement('p');  
 p.innerHTML = "Weight: " + edge.weight;  
 info.appendChild(p);  
  
 p = document.createElement('p');  
 p.innerHTML = "Error Probability: " + edge.error;  
 info.appendChild(p);  
  
  
  
 document.getElementById('info').style.display = 'block';  
}  
  
function getNodes() {  
 return nodes;  
}  
function getEdges() {  
 return edges;  
}  
  
function getNetworkNodes() {  
 return networkDataNodes;  
}  
function getNetworkEdges() {  
 return networkDataEdges;  
}  
  
function getNetwork() {  
 return network;  
}  
  
function getFakeWeight() {  
 return FAKE\_WEIGHT  
}