МІНІСТЕРСТВО КУЛЬТУРИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КУЛЬТУРИ І МИСТЕЦТВ

ФАКУЛЬТЕТ ЕКОНОМІКИ І ІТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Звіт

з дисципліни «Системний аналіз»

на тему «Обробка графової структури»

Виконав:

студент 3 курсу

групи КН-23

Коваль Максим

Київ –2015

ЗМІСТ

[РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ 3](#_Toc434920183)

[РОЗДІЛ 2 ОПИСИ АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ 4](#_Toc434920184)

[2.1 Опис алгоритму додавання нової вершини з відповідними дугами до графа 4](#_Toc434920185)

[2.2 Опис алгоритму додавання нової дуги до графу G 5](#_Toc434920186)

[2.3 Опис алгоритму видалення дуги 5](#_Toc434920187)

[2.4 Опис алгоритму видалення дуги 6](#_Toc434920188)

[2.5 Опис алгоритму видалення дуги (зі збереженням зв’язків) 6](#_Toc434920189)

[РОЗДІЛ 3 БЛОК-СХЕМИ ПРОГРАМНОГО ПРОДУКТУ 8](#_Toc434920190)

[РОЗДІЛ 4 ЛІСТИНГ КОДУ 11](#_Toc434920191)

[4.1 Файл index.html 11](#_Toc434920192)

[4.2 Файл style.css 11](#_Toc434920193)

[4.3 Файл script.js 12](#_Toc434920194)

[РОЗДІЛ 5 SCREENSHOTS 21](#_Toc434920195)

# РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ

Написати програму обробки графової структури, яка передбачає:

* Додавання нової дуги до графа G.
* Додавання нової вершини з відповідними дугами до графа G.
* Видалення дуги (з перевіркою збереження основних вимог до графа).
* Видалення вершини без збереження зв’язків (з перевіркою збереження основних вимог до графа).
* Видалення вершини зі збереженням зв’язків.

# РОЗДІЛ 2 ОПИСИ АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

Задача була розв’язана за допомогою мови програмування JavaScript і її бібліотеки jQuery. Для візуалізації задачі в браузері були використані такі засоби верстки як HTML5 і CSS3. Переглянути роботу програми можна в будь-якому сучасному браузері.

Візуалізація графа була виконана за допомогою бібліотеки vis.js, яка допомагає вирішити завдання такого типу. Граф будувався відповідно своєї матриці суміжності, яка була програмно представлена у виді двохвимірного масиву.

## 2.1 Опис алгоритму додавання нової вершини з відповідними дугами до графа

1. До кінця кожного рядку матриці додаємо елемент 0.
2. В кінець матриці додаємо новий рядок, який містить стільки ж елементів, скільки мають його вище розміщенні рядки.
3. Збільшуємо число вершин графа.
4. Запитуємо у користувача від якої вершини потрібно провести дугу до нової вершини.
5. Запитуємо у користувача до якої вершини потрібно провести дугу від нової вершини.
6. Проводимо відповідні дуги замінюючи 0 на 1 в матриці суміжності.
7. Проводимо перевірку на ізольовані вершини. Якщо є ізольована вершина, то виводимо попередження і повертаємо матрицю до початкового стану, який був перед натисканням кнопки.
8. Проводимо перевірку на розділений граф. Якщо граф розділений, то виводимо попередження і повертаємо матрицю до початкового стану.
9. Проводимо перевірку на початкові вершини графа. Якщо граф має дві або більше початкові вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
10. Проводимо перевірку на кінцеві вершини графа. Якщо граф має дві або більше кінцеві вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
11. Перебудовуємо візуально матрицю.
12. Перебудовуємо візуально граф.

## 2.2 Опис алгоритму додавання нової дуги до графу G

1. Запитуємо у користувача від якої вершини потрібно провести дугу.
2. Запитуємо користувача до якої вершини потрібно провести дугу.
3. Змінюємо значення 0 на 1 в матриці суміжності відповідно до введених користувачем даних.
4. Проводимо перевірку на ізольовані вершини. Якщо є ізольована вершина, то виводимо попередження і повертаємо матрицю до початкового стану, який був перед натисканням кнопки.
5. Проводимо перевірку на розділений граф. Якщо граф розділений, то виводимо попередження і повертаємо матрицю до початкового стану.
6. Проводимо перевірку на початкові вершини графа. Якщо граф має дві або більше початкові вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
7. Проводимо перевірку на кінцеві вершини графа. Якщо граф має дві або більше кінцеві вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
8. Перебудовуємо візуально матрицю.
9. Перебудовуємо візуально граф.

## 2.3 Опис алгоритму видалення дуги

1. Запитуємо у користувача від якої вершини потрібно видалити дугу.
2. Запитуємо користувача до якої вершини потрібно видалити дугу.
3. Змінюємо значення 1 на 0 в матриці суміжності відповідно до введених користувачем даних.
4. Проводимо перевірку на ізольовані вершини. Якщо є ізольована вершина, то виводимо попередження і повертаємо матрицю до початкового стану, який був перед натисканням кнопки.
5. Проводимо перевірку на розділений граф. Якщо граф розділений, то виводимо попередження і повертаємо матрицю до початкового стану.
6. Проводимо перевірку на початкові вершини графа. Якщо граф має дві або більше початкові вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
7. Проводимо перевірку на кінцеві вершини графа. Якщо граф має дві або більше кінцеві вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
8. Перебудовуємо візуально матрицю.
9. Перебудовуємо візуально граф.

## 2.4 Опис алгоритму видалення дуги

1. Запитуємо у користувача яку вершину потрібно видалити.
2. Відповідно до введених даних видаляємо рядок з індексом вершини який ввів користувач.
3. Видаляємо стовбець з індексом вершини який ввів користувач.
4. Зменшуємо кількість вершин графа.
5. Проводимо перевірку на ізольовані вершини. Якщо є ізольована вершина, то виводимо попередження і повертаємо матрицю до початкового стану, який був перед натисканням кнопки.
6. Проводимо перевірку на розділений граф. Якщо граф розділений, то виводимо попередження і повертаємо матрицю до початкового стану.
7. Проводимо перевірку на початкові вершини графа. Якщо граф має дві або більше початкові вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
8. Проводимо перевірку на кінцеві вершини графа. Якщо граф має дві або більше кінцеві вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
9. Перебудовуємо візуально матрицю.
10. Перебудовуємо візуально граф.

## 2.5 Опис алгоритму видалення дуги (зі збереженням зв’язків)

1. Запитуємо у користувача яку вершину потрібно видалити.
2. Перевіряємо, чи є вершини, які з’єднанні між собою за допомогою вершини, яку потрібно видалити. Якщо є, то з’єднуємо їх напряму без використання допоміжної вершини.
3. Відповідно до введених даних видаляємо рядок з індексом вершини який ввів користувач.
4. Видаляємо стовбець з індексом вершини який ввів користувач.
5. Зменшуємо кількість вершин графа.
6. Проводимо перевірку на ізольовані вершини. Якщо є ізольована вершина, то виводимо попередження і повертаємо матрицю до початкового стану, який був перед натисканням кнопки.
7. Проводимо перевірку на розділений граф. Якщо граф розділений, то виводимо попередження і повертаємо матрицю до початкового стану.
8. Проводимо перевірку на початкові вершини графа. Якщо граф має дві або більше початкові вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
9. Проводимо перевірку на кінцеві вершини графа. Якщо граф має дві або більше кінцеві вершини, або зовсім їх не має, то виводимо попередження і повертаємо матрицю до початкового стану.
10. Перебудовуємо візуально матрицю.
11. Перебудовуємо візуально граф.

# РОЗДІЛ 3 БЛОК-СХЕМИ ПРОГРАМНОГО ПРОДУКТУ

D:\ThirdCourse15_16\System Analysis\Koval\графи\Головна блок-схема.png

Рис. 1. Головна блок-схема

D:\ThirdCourse15_16\System Analysis\Koval\графи\Блок-схема А.png

Рис. 2. Блок-схема А

D:\ThirdCourse15_16\System Analysis\Koval\графи\Блок-схема В.png

Рис. 3. Блок-схема В

D:\ThirdCourse15_16\System Analysis\Koval\графи\Блок-схема С.png

Рис. 4. Блок-схема С

D:\ThirdCourse15_16\System Analysis\Koval\графи\Блок-схема D.png

Рис. 5. Блок-схема D

D:\ThirdCourse15_16\System Analysis\Koval\графи\Блок-схема Е.png

Рис. 6. Блок-схема Е

# РОЗДІЛ 4 ЛІСТИНГ КОДУ

## 4.1 Файл index.html

<!DOCTYPE HTML>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script src="jquery-2.1.4.min.js"></script>

<script src="jquery-ui.js"></script>

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

<link rel="stylesheet" type="text/css" href="style.css">

<title>graph</title>

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

</head>

<body>

<div id="task">

<span>Number of nodes: </span>

<input type="text" id="col"/>

<button id="apply">Apply</button>

</div>

</body>

</html>

## 4.2 Файл style.css

body {

background-color: #FFE773;

}

button, input, #task, #parent, #matrix {

border-radius: 5px;

}

#task {

width: 95%;

margin: 0 auto;

height: 70px;

background-color: #BFA730;

padding: 5px 0;

}

#task button {

width: 20%;

font-size: 15px;

height: 26px;

margin-top: 4px;

}

#task input {

width: 10%;

height: 18px;

}

#matrix {

width: 60%;

margin: 5px auto;

background-color: #BFA730;

text-align: center;

}

#inputs, .marks {

width: 30px;

height: 30px;

text-align: center;

}

.marks {

background-color: #ddd;

}

#parent {

position: relative;

width: 1000px;

height: 600px;

margin: 5px auto;

background-color: #BFA730;

}

## 4.3 Файл script.js

$(document).ready(function () {

var matrix = [];

var twoFirstNodes = function() {

var firstNodes = 0;

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

var kol = 0;

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

kol += parseInt(matrix[j][i]);

}

if (kol === 0) {

firstNodes++;

}

}

if (firstNodes === 0) {

return 0;

}

else if (firstNodes > 1) {

return 1;

}

};

var isolNode = function() {

var isolNodes = 0;

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

var iKol = 0;

var jKol = 0;

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

iKol += parseInt(matrix[i][j]);

jKol += parseInt(matrix[j][i]);

}

if (iKol + jKol === 0) {

isolNodes++;

}

}

if (isolNodes > 0) {

return 1;

}

};

var twoLastNodes = function() {

var lastNodes = 0;

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

var kol = 0;

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

kol += parseInt(matrix[i][j]);

}

if (kol === 0) {

lastNodes++;

}

}

if (lastNodes > 1) {

return 1;

}

else if (lastNodes === 0) {

return 0;

}

};

var dividedGraph = function() {

var firstNodes = 0;

var lastNodes = 0;

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

var colFirst = 0;

var colLast = 0;

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

colFirst += parseInt(matrix[j][i]);

colLast += parseInt(matrix[i][j]);

}

if (colFirst === 0) {

firstNodes++;

}

if (colLast === 0) {

lastNodes++;

}

}

if (lastNodes > 1 && firstNodes > 1) {

return 1;

}

};

var drawMatrix = function() {

$("#matrix").remove();

$("body").append("<div id='matrix'></div>");

var col = parseInt($('#col').val());

$("#matrix").append("<input class='marks' value='i/j'></input>");

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

$("#matrix").append("<input class='marks' value="+(i+1).toString()+"></input>");

}

$("#matrix").append("<br>");

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

$("#matrix").append("<input class='marks' value="+(i+1).toString()+"></input>");

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

var element = document.createElement('input');

element.value = matrix[i][j];

element.id = 'inputs';

element.className = 'element'+i.toString();

$('#matrix').append(element);

}

$("#matrix").append("<br>");

}

$("#matrix").append("<button id='build'>Build graph</button>");

};

var readMatrix = function() {

var col = parseInt($('#col').val());

matrix = []; //clear matrix

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

var inputs = document.getElementsByClassName( "element"+i.toString() ),

elements = [].map.call(inputs, function( input ) {

return input.value;

}).join( ',' );

var dopMatrix = elements.split(",");

matrix.push(dopMatrix);

}

}

var drawGraph = function() {

$("#parent").remove();

$("body").append("<div id='parent'></div>");

var col = parseInt($('#col').val());

var nodes = new vis.DataSet();

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

nodes.add({id: i, label: (i+1).toString()});

};

var edges = new vis.DataSet();

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

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

if (matrix[i][j] === "1") {

edges.add({from: i, to: j, arrows:'to'});

}

}

}

// create a network

var container = document.getElementById('parent');

var data = {

nodes: nodes,

edges: edges

};

var options = {};

options.nodes = {

color: '#ff0'

}

var network = new vis.Network(container, data, options);

};

$('#apply').click(function() {

var col = parseInt($('#col').val());

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

var dopMatrix = [];

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

dopMatrix.push(0);

}

matrix.push(dopMatrix);

}

drawMatrix();

$('#apply').remove();

});

$(document).on("click", "#build", function(){

readMatrix();

if (isolNode() === 1) {

alert("Graph has insulated node!");

}

else if (dividedGraph() === 1) {

alert("Divided graph!");

}

else if (twoFirstNodes() === 1) {

alert("Graph has two or more first nodes!");

}

else if (twoFirstNodes() === 0) {

alert("Graph doesn't have first node!");

}

else if (twoLastNodes() === 0) {

alert("Graph doesn't have last node!");

}

else if (twoLastNodes() === 1) {

alert("Graph has two or more last nodes!");

}

else {

$("body").append("<div id='parent'></div>");

drawGraph();

$("#task").append("<br/>");

$("#task").append("<button id='addNode'>Add node</button>");

$("#task").append("<button id='addEdge'>Add edge</button>");

$("#task").append("<button id='deleteNode'>Delete node</button>");

$("#task").append("<button id='deleteEdge'>Delete edge</button>");

$("#task").append("<button id='deleteNodeSave'>Delete node with saving</button>");

$("#build").remove();

}

});

$(document).on("click", "#addNode", function(){

var col = parseInt($('#col').val());

var dopMatrix = [];

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

dopMatrix.push('0');

matrix[i].push('0');

}

dopMatrix.push('0');

matrix.push(dopMatrix);

col++;

var i = prompt("Enter a number of node from which you want to draw edge to new node(Enter '-' if you dont need to draw edge to new node)");

var j = prompt("Enter a number of node to which you want to draw edge from new node(Enter '-' if you dont need to draw edge from new node)");

i--;

j--;

if (i != '-') {

matrix[i][col-1] = '1';

}

if (j != '-') {

matrix[col-1][j] = '1';

}

if (isolNode() === 1) {

alert("Graph has insulated node!");

matrix.splice(col-1, 1);

col--;

for (var k=0; k<col; k++) {

matrix[k].splice(col-1, 1);

}

}

else if (dividedGraph() === 1) {

alert("Divided graph!");

matrix.splice(col-1, 1);

col--;

for (var k=0; k<col; k++) {

matrix[k].splice(col-1, 1);

}

}

else if (twoFirstNodes() === 1) {

alert("Graph has two or more first nodes!");

matrix.splice(col-1, 1);

col--;

for (var k=0; k<col; k++) {

matrix[k].splice(col-1, 1);

}

}

else if (twoFirstNodes() === 0) {

alert("Graph doesn't have first node!");

matrix.splice(col-1, 1);

col--;

for (var k=0; k<col; k++) {

matrix[k].splice(col-1, 1);

}

}

else if (twoLastNodes() === 0) {

alert("Graph doesn't have last node!");

matrix.splice(col-1, 1);

col--;

for (var k=0; k<col; k++) {

matrix[k].splice(col-1, 1);

}

}

else if (twoLastNodes() === 1) {

alert("Graph has two or more last nodes!");

matrix.splice(col-1, 1);

col--;

for (var k=0; k<col; k++) {

matrix[k].splice(col-1, 1);

}

}

$('#col').val(col);

drawMatrix();

drawGraph();

$("#build").remove();

});

$(document).on("click", "#addEdge", function(){

var i = prompt("Enter a number of node from wich you want to add edge");

var j = prompt("Enter a number of node to wich you want to add edge");

i--;

j--;

if (matrix[i][j] != '1') {

matrix[i][j] = '1';

if (isolNode() === 1) {

alert("Graph has insulated node!");

matrix[i][j] = '0';

}

else if (dividedGraph() === 1) {

alert("Divided graph!");

matrix[i][j] = '0';

}

else if (twoFirstNodes() === 1) {

alert("Graph has two or more first nodes!");

matrix[i][j] = '0';

}

else if (twoFirstNodes() === 0) {

alert("Graph doesn't have first node!");

matrix[i][j] = '0';

}

else if (twoLastNodes() === 0) {

alert("Graph doesn't have last node!");

matrix[i][j] = '0';

}

else if (twoLastNodes() === 1) {

alert("Graph has two or more last nodes!");

matrix[i][j] = '0';

}

}

else {

alert("This edge is already exist!");

}

drawMatrix();

drawGraph();

$("#build").remove();

});

$(document).on("click", "#deleteNode", function(){

var col = parseInt($('#col').val());

var i = prompt("Enter a number of node which you want to delete");

i--;

var dopMatrix = [];

for (var m = 0; m<col-1; m++) {

dopMatrix[m] = matrix[i][m];

}

var lastElements = [];

for (var k = 0; k<col; k++) {

lastElements[k] = matrix[k][col-1];

}

for (var n=0; n<col; n++) {

matrix[n].splice(col-1, 1);

}

matrix.splice(i, 1);

col--;

if (dividedGraph() === 1) {

alert("Divided graph!");

matrix.splice(i, 0, dopMatrix);

col++;

for (var a=0; a<col; a++) {

matrix[a].splice(col-1, 0, lastElements[a]);

}

}

else if (isolNode() === 1) {

alert("Graph has insulated node!");

matrix.splice(i, 0, dopMatrix);

col++;

for (var a=0; a<col; a++) {

matrix[a].splice(col-1, 0, lastElements[a]);

}

}

else if (twoFirstNodes() === 1) {

alert("Graph has two or more first nodes!");

matrix.splice(i, 0, dopMatrix);

col++;

for (var a=0; a<col; a++) {

matrix[a].splice(col-1, 0, lastElements[a]);

}

}

else if (twoFirstNodes() === 0) {

alert("Graph doesn't have first node!");

matrix.splice(i, 0, dopMatrix);

col++;

for (var a=0; a<col; a++) {

matrix[a].splice(col-1, 0, lastElements[a]);

}

}

else if (twoLastNodes() === 0) {

alert("Graph doesn't have last node!");

matrix.splice(i, 0, dopMatrix);

col++;

for (var a=0; a<col; a++) {

matrix[a].splice(col-1, 0, lastElements[a]);

}

}

else if (twoLastNodes() === 1) {

alert("Graph has two or more last nodes!");

matrix.splice(i, 0, dopMatrix);

col++;

for (var a=0; a<col; a++) {

matrix[a].splice(col-1, 0, lastElements[a]);

}

}

$('#col').val(col);

drawMatrix();

drawGraph();

$("#build").remove();

});

$(document).on("click", "#deleteEdge", function(){

var i = prompt("Enter a number of node from wich you want to delete edge");

var j = prompt("Enter a number of node to wich you want to delete edge");

i--;

j--;

if (matrix[i][j] != '0') {

matrix[i][j] = '0';

if (isolNode() === 1) {

alert("Graph has insulated node!");

matrix[i][j] = '1';

}

else if (dividedGraph() === 1) {

alert("Divided graph!");

matrix[i][j] = '1';

}

else if (twoFirstNodes() === 1) {

alert("Graph has two or more first nodes!");

matrix[i][j] = '1';

}

else if (twoFirstNodes() === 0) {

alert("Graph doesn't have first node!");

matrix[i][j] = '1';

}

else if (twoLastNodes() === 0) {

alert("Graph doesn't have last node!");

matrix[i][j] = '1';

}

else if (twoLastNodes() === 1) {

alert("Graph has two or more last nodes!");

matrix[i][j] = '1';

}

}

else {

alert("This edge is don't exist!");

}

drawMatrix();

drawGraph();

$("#build").remove();

});

$(document).on("click", "#deleteNodeSave", function(){

var col = parseInt($('#col').val());

var i = prompt("Enter a number of node which you want to delete");

i--;

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

if (matrix[i][j] === '1') {

for (var t=0; t<col; t++) {

if (matrix[t][i] === '1') {

matrix[t][j] = '1';

}

}

}

}

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

matrix[j].splice(i,1);

}

matrix.splice(i,1);

col--;

$('#col').val(col);

drawMatrix();

drawGraph();

$("#build").remove();

console.log(matrix);

});

});

# РОЗДІЛ 5 SCREENSHOTS



Рис. 7. Початкове вікно програми

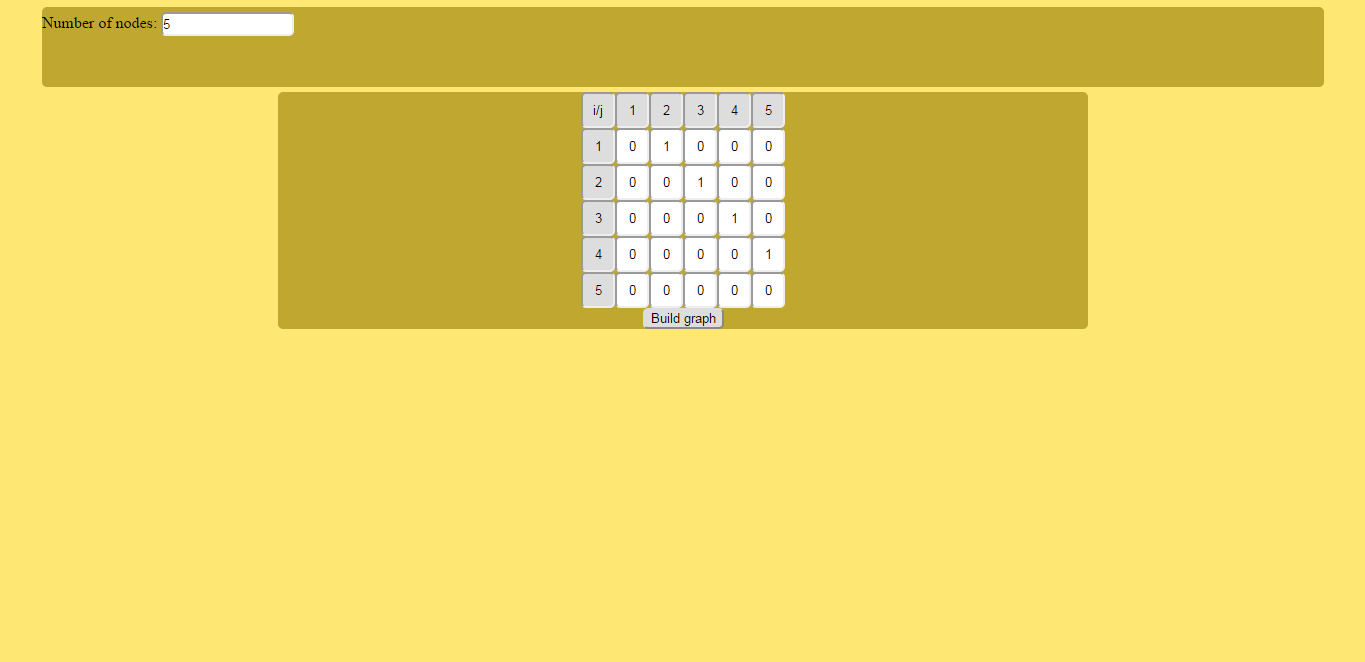


Рис. 8. Введення даних матриці суміжності

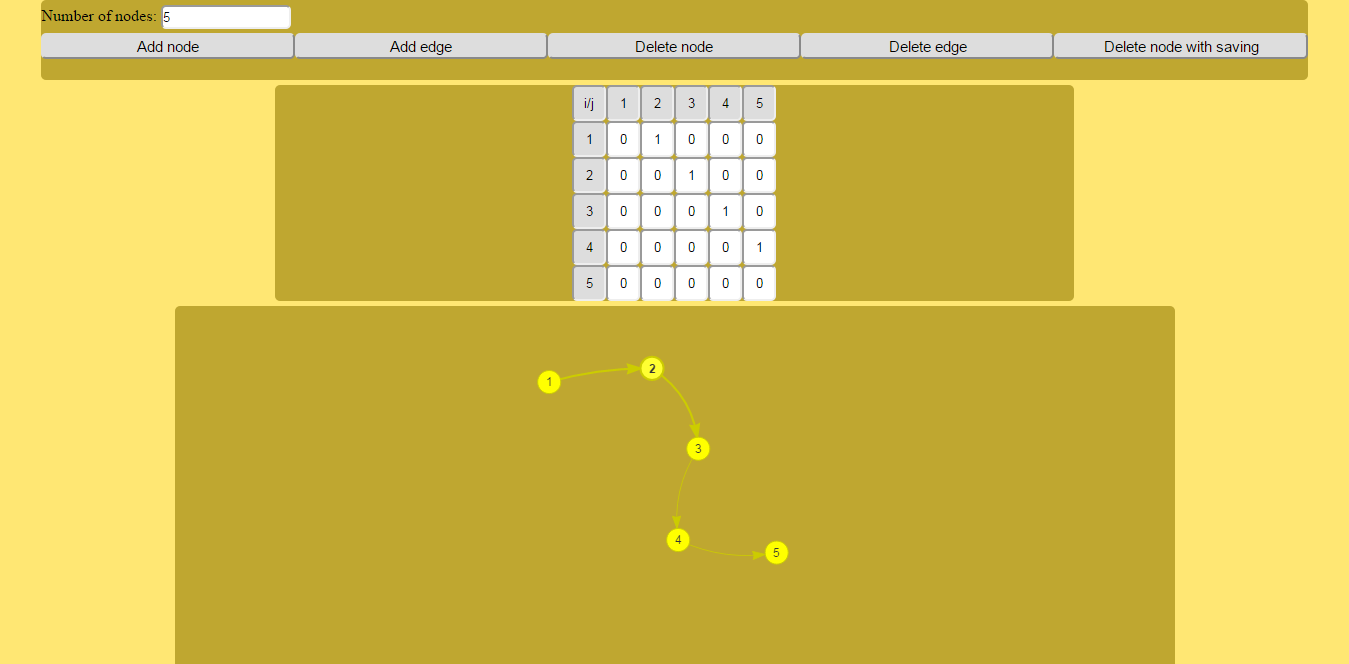


Рис. 9. Побудова графової структури

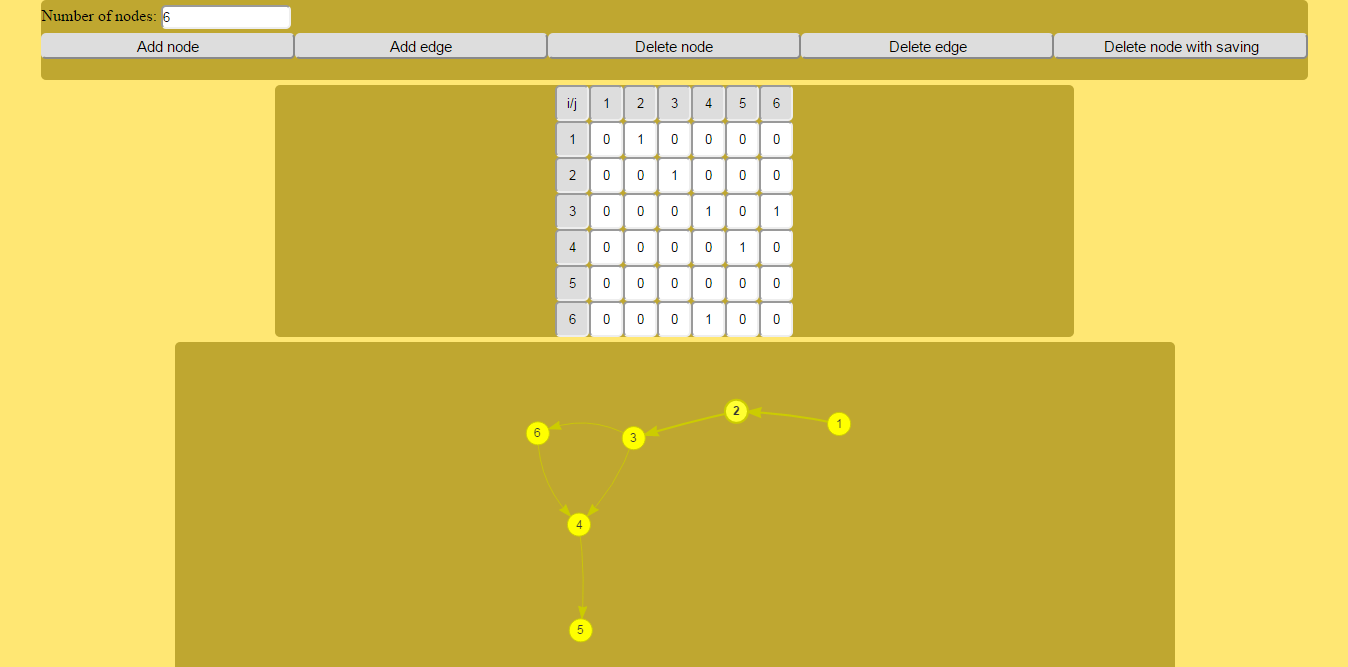


Рис. 10. Додавання вершини з відповідними дугами

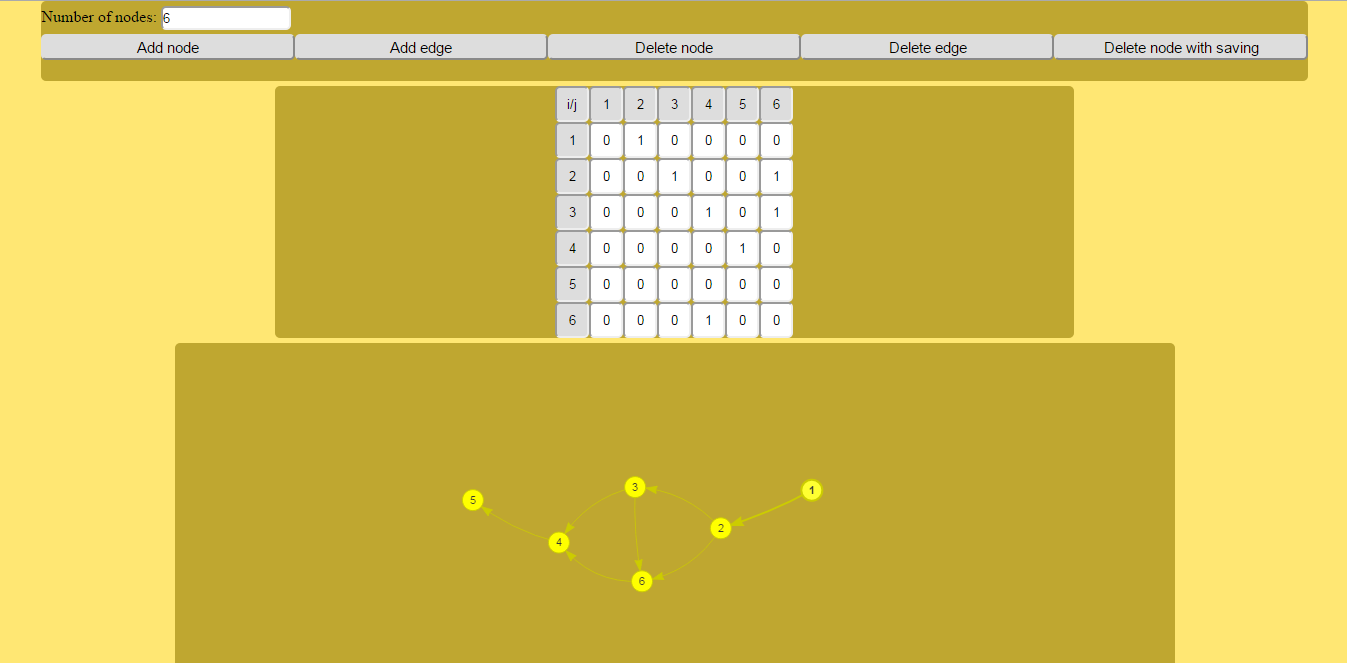


Рис. 11. Додавання дуги

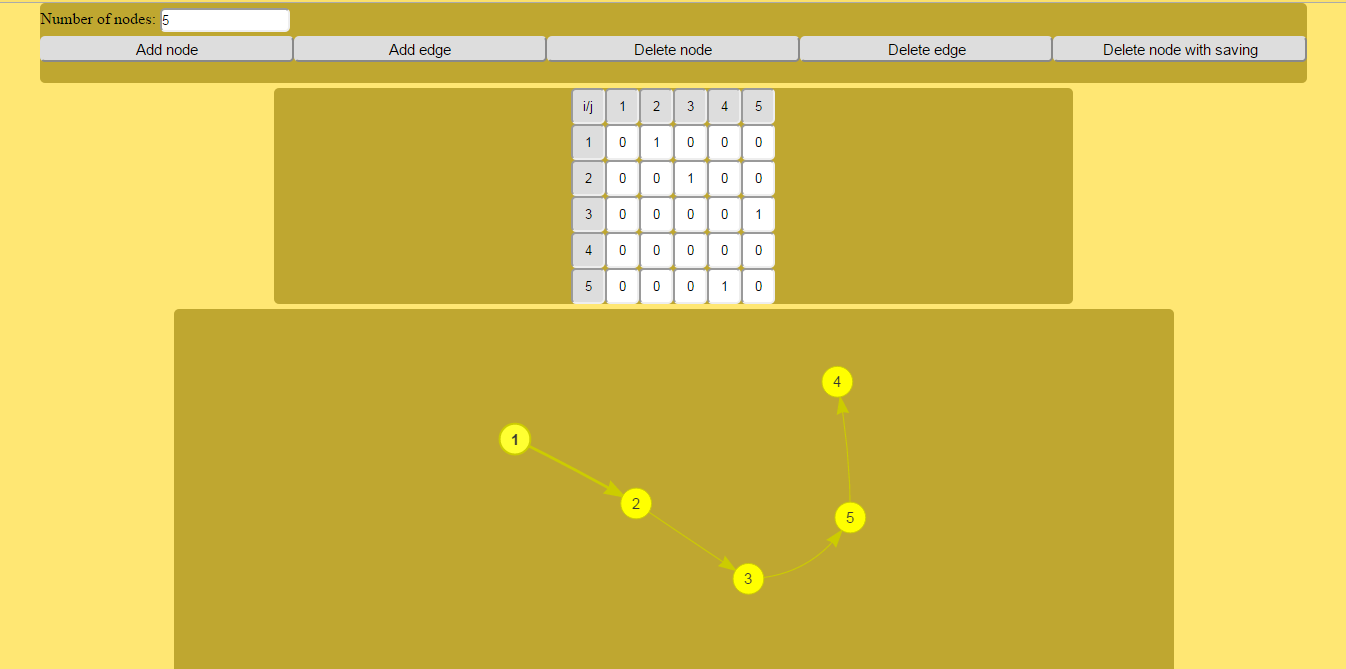


Рис. 12. Видалення вершини

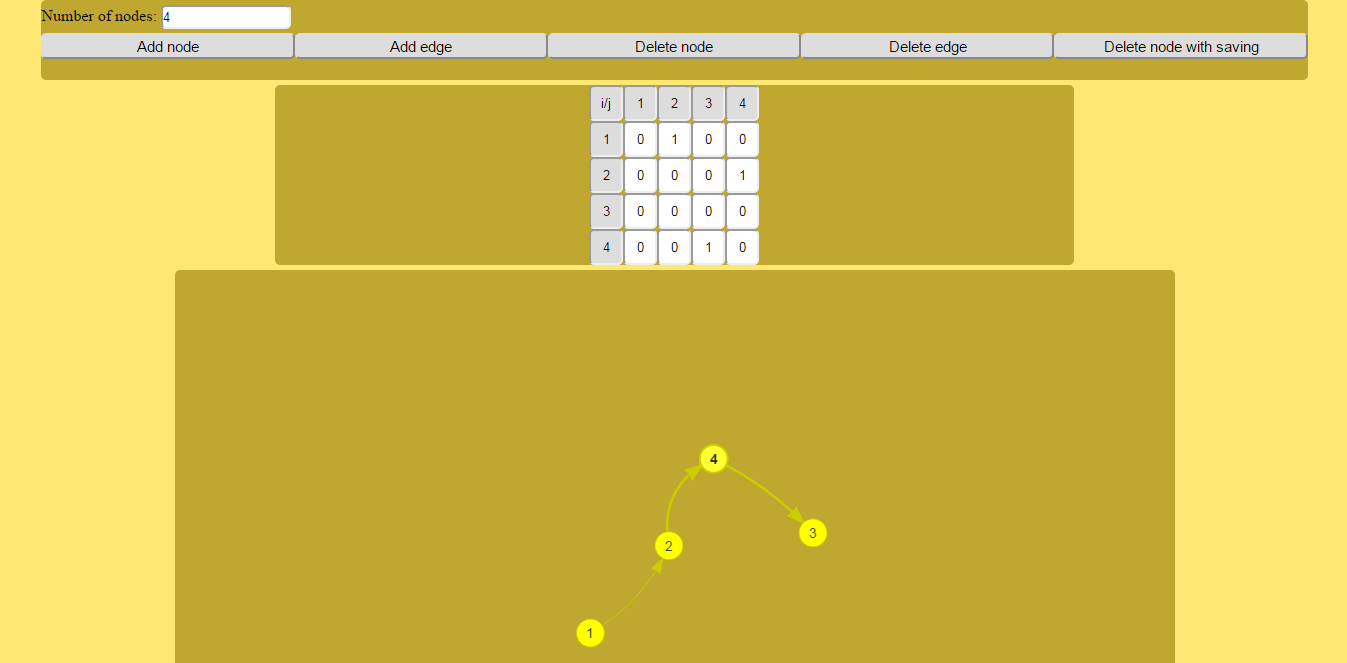


Рис. 13. Видалення вершини зі збереженням

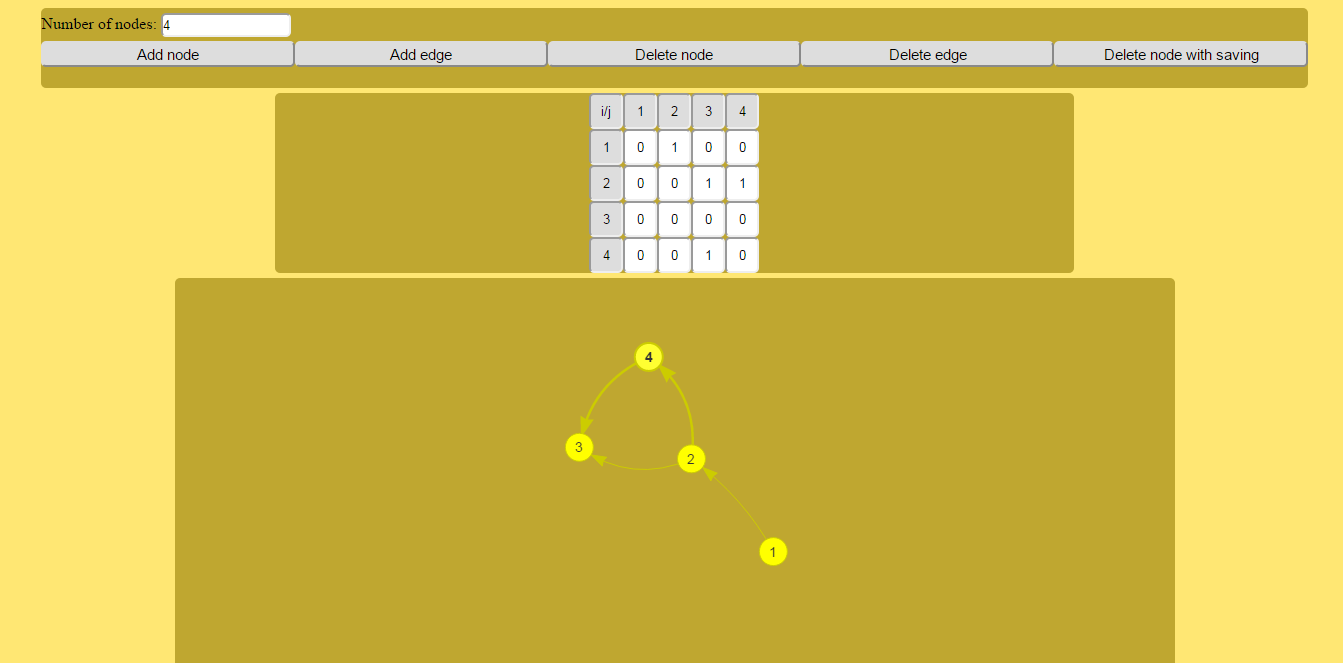


Рис. 14. Додавання нової дуги

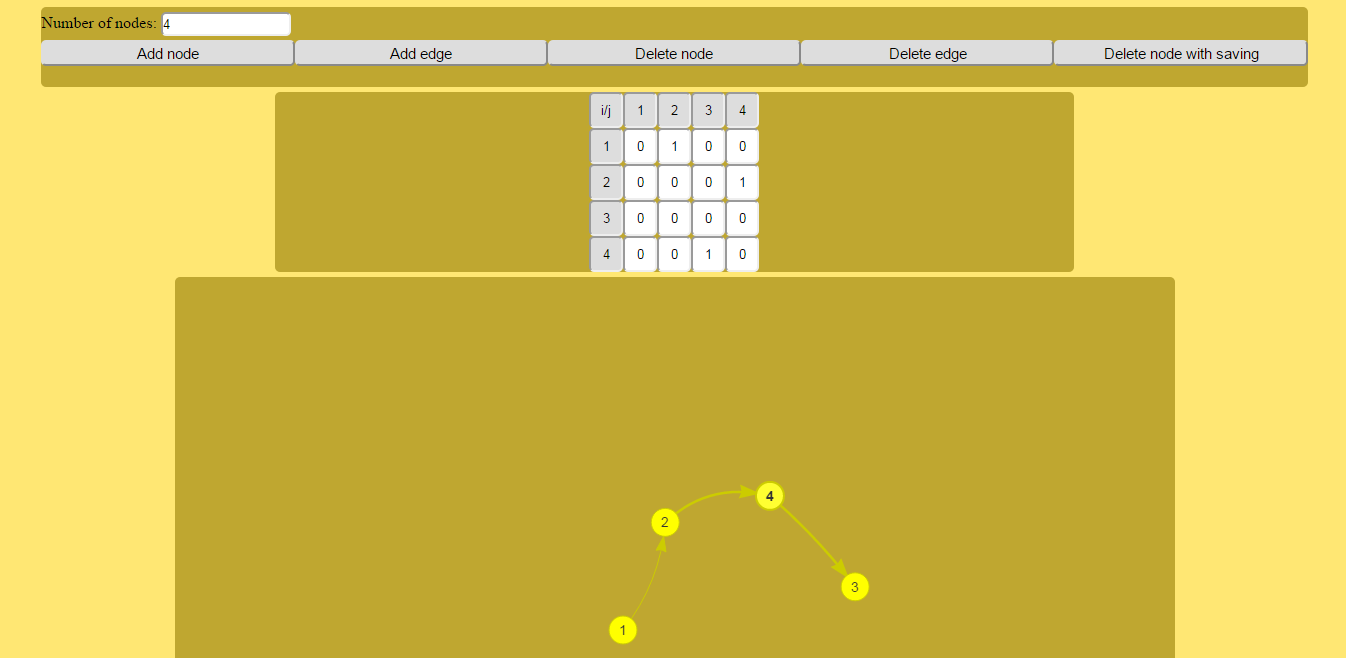


Рис. 15. Видалення дуги