МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №6**

**з дисципліни «Алгоритми та структури даних»**

**на тему: «Графи та пошук у них»**

Виконав: студент гр. ПЗ1911

Сіньков Г.О.

Прийняла: ас. каф. КІТ

Куроп’ятник О. С.

Дніпро, 2020

**Лабораторна робота №6**

**Тема.** Графи та пошук у них.

**Мета**. Ознайомитися з поняттям графа. Отримати практичні навички реалізації різних представлень та обходу графів.

**Постановка задачі**

Написати програму мовою java, що складається з класів:

-вершини, який містить поле назви;

-графа з представленням матрицею суміжності;

-графа з представленням списком суміжності;

-інтерфейсу користувача.

Взаємодію з користувачем організувати через текстове меню. Для кожного пункту меню передбачити зворотній зв'язок у вигляді виведення результатів виконання дії та/або текстового повідомлення.

Передбачити такі дії над графом: додавання вершин, ребер (дуг), видалення дуг, обхід в глибину та ширину, з можливістю завдання стартової вершини.

Розробити тестові приклади для перевірки працездатності програми.

Варіанти завдань: парні – орієнтований граф, непарні – неорієнтований

**Текст програми**

Vertex.java

package com.company;  
  
  
// Вершинный класс  
public class Vertex {  
 private String Char;  
 public boolean wasVisited; //для матриці суміжності  
  
 public Vertex(String Char) {  
 this.Char = Char;  
 wasVisited = false;  
 }  
  
 public String getChar() {  
 return Char;  
 }  
}

Graph.java

package com.company;  
  
import java.util.\*;  
  
public class Graph { //граф матриці суміжності  
  
 private Vertex[] vertices;  
 public int currAmountOfVer;  
 private int amount\_verteces;  
 private int[][] adjacency\_matrix;  
  
 public Graph(int vertex) {  
 amount\_verteces = vertex;  
 adjacency\_matrix = new int[amount\_verteces + 1][amount\_verteces + 1];  
 for (int i = 0; i < amount\_verteces + 1; i++)  
 for (int j = 0; j < amount\_verteces + 1; j++)  
 adjacency\_matrix[i][j] = 0;  
  
 vertices = new Vertex[amount\_verteces + 1];  
 currAmountOfVer = -1;  
 }  
  
 public void addEdge(String start, String end) {  
 try {  
 int Start = -1, End = -1;  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 if(start.equals(vertices[i].getChar()))  
 Start = i;  
 if(end.equals(vertices[i].getChar()))  
 End = i;  
 }  
 adjacency\_matrix[Start][End] = 1;  
 adjacency\_matrix[End][Start] = 1;  
 }  
 catch (ArrayIndexOutOfBoundsException index) {  
 System.*out*.println("Вершин не існує");  
 }  
 }  
 public void removeEdge(String start, String end) {  
 try {  
 int Start = -1, End = -1;  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 if(start.equals(vertices[i].getChar()))  
 Start = i;  
 if(end.equals(vertices[i].getChar()))  
 End = i;  
 }  
 adjacency\_matrix[Start][End] = 0;  
 adjacency\_matrix[End][Start] = 0;  
 }  
 catch (ArrayIndexOutOfBoundsException index) {  
 System.*out*.println("Вершин не існує");  
 }  
 }  
  
 public void addVertex(Vertex vertex) {  
 if(currAmountOfVer > amount\_verteces)  
 System.*out*.println("Немає місця для додавання Vertex");  
 else {  
 ++currAmountOfVer;  
 vertices[currAmountOfVer] = vertex;  
 }  
 }  
  
 public void deleteVertex(String Char) {  
 int indxOfVertexFound = -1;  
 boolean found = false;  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 if(Char.equals(vertices[i].getChar())) {  
 indxOfVertexFound = i;  
 currAmountOfVer--;  
 found = true;  
 }  
 }  
 if(!found) {  
 System.*out*.println("Немає такої вершини!");  
 return;  
 }  
 for(int i = indxOfVertexFound; i < currAmountOfVer + 1; i++) {  
 vertices[i] = vertices[i + 1];  
 for (int j = 0; j < currAmountOfVer + 1; j++) {  
 adjacency\_matrix[i][j] = adjacency\_matrix[i + 1][j + 1];  
 }  
 }  
 }  
  
 private ArrayList<Vertex> findNeighbours(Vertex node) {  
 int nodeIndex=-1;  
 ArrayList<Vertex> neighbours = new ArrayList<>();  
  
 //індекс пошуку для вузла  
 for (int i = 0; i <currAmountOfVer + 1; i++) {  
 if(vertices[i].equals(node)) {  
 nodeIndex=i;  
 break;  
 }  
 }  
  
 //додавання сусідніх вузлів до списку  
 if(nodeIndex != -1) {  
 for (int i = 0; i < currAmountOfVer + 1; i++)  
 if(adjacency\_matrix[nodeIndex][i] == 1)  
 neighbours.add(vertices[i]);  
 }  
  
 return neighbours;  
 }  
  
 public void BFS(int start){  
 if(start > currAmountOfVer || start < 0 ) //виняток для неправильного номера вузла  
 throw new ArrayIndexOutOfBoundsException("Поза межами");  
  
 Queue<Vertex> queue = new LinkedList<>(); //черга для зберігання сусідніх та невідвіданих вузлів  
 queue.add(vertices[start]); //почніть з пуску стартового вузла  
 vertices[start].wasVisited = true;  
  
 while (!queue.isEmpty())  
 {  
 Vertex element = queue.remove();  
 System.*out*.print(element.getChar() + " -> ");  
  
 ArrayList<Vertex> neighbours = findNeighbours(element); //отримати сусідні вузли  
 for (Vertex node : neighbours) { //штовхання невідвіданих сусідських вузлів  
 if(node != null && !node.wasVisited) {  
 queue.add(node);  
 node.wasVisited = true;  
 }  
 }  
 }  
 System.*out*.println();  
 //набір, не відвідуваний для всіх вузлів  
 for (int i = 0; i < currAmountOfVer + 1; i++)  
 vertices[i].wasVisited = false;  
 }  
  
 public void print() {  
 for (int i = 0; i < currAmountOfVer + 1; i++)  
 System.*out*.print(" " + vertices[i].getChar());  
  
 System.*out*.println();  
 for (int i = 0; i < currAmountOfVer + 1; i++) {  
 System.*out*.print(vertices[i].getChar());  
 for (int j = 0; j < currAmountOfVer + 1; j++)  
 System.*out*.print(" " + adjacency\_matrix[i][j]);  
 System.*out*.println();  
 }  
 }  
  
 public void DFS(int start) throws ArrayIndexOutOfBoundsException {  
 if(start > currAmountOfVer || start < 0 ) //виняток для неправильного номера вузла  
 throw new ArrayIndexOutOfBoundsException("Поза межами");  
  
 Stack<Vertex> stack = new Stack<>(); //стек для зберігання сусідніх та невідвіданих вузлів  
 stack.add(vertices[start]); //почніть з пуску стартового вузла  
  
 while(!stack.isEmpty()) {  
 Vertex element=stack.pop(); //вимкнути вузол зі стеку та надрукувати, якщо його не відвідали  
 if(!element.wasVisited) {  
 System.*out*.print(element.getChar() + " -> ");  
 element.wasVisited = true;  
 }  
  
 ArrayList<Vertex> neighbours = findNeighbours(element); //отримати сусідні вузли  
 Collections.*reverse*(neighbours); //переверніть список, щоб перенести вузли в стек у правильному порядку  
 for(Vertex node : neighbours) //штовхання невідвіданих сусідських вузлів  
 if(node != null && !node.wasVisited)  
 stack.push(node);  
  
 }  
  
 System.*out*.println();  
 //набір, не відвідуваний для всіх вузлів  
 for (int i = 0; i < currAmountOfVer + 1; i++)  
 vertices[i].wasVisited = false;  
 }  
  
 public void SetVisit(boolean res){  
 for (int i = 0; i < currAmountOfVer; i++)  
 vertices[i].wasVisited = res;  
 }  
}

Graph2.java

package com.company;  
  
import java.util.\*;  
  
public class Graph2 { //граф списком суміжності  
  
 private List<Vertex> vertexes;  
 private List<List<Vertex>> adjencyList;  
 private int currAmountVertex;  
  
 public Graph2() {  
 currAmountVertex = -1;  
 vertexes = new ArrayList<>();  
 adjencyList = new ArrayList<>();  
 }  
  
 public void makeEdge(String to, String from) {  
 try {  
 int To = 0, From = 0, IndxFound = -1;  
 for (Vertex v : vertexes) {  
 ++IndxFound;  
 if (to.equals(v.getChar()))  
 To = IndxFound;  
 if (from.equals(v.getChar()))  
 From = IndxFound;  
 }  
 adjencyList.get(To).add(vertexes.get(From));  
 adjencyList.get(From).add(vertexes.get(To));  
 }  
 catch (ArrayIndexOutOfBoundsException index) {  
 System.*out*.println("Вершин не існує");  
 }  
 }  
  
 public void removeEdge(String to, String from){  
 try {  
 int To = -1, From = -1, IndxFound = -1;  
 for (int i = 0; i < currAmountVertex + 1; i++) {  
 ++IndxFound;  
 if (to.equals(vertexes.get(i).getChar()))  
 To = IndxFound;  
 if (from.equals(vertexes.get(i).getChar()))  
 From = IndxFound;  
 }  
 adjencyList.get(To).remove(vertexes.get(From));  
 adjencyList.get(From).remove(vertexes.get(To));  
 }  
 catch (ArrayIndexOutOfBoundsException index) {  
 System.*out*.println("Вершин не існує");  
 }  
 }  
  
 public void addVertex(Vertex vertex){  
 ++currAmountVertex;  
 vertexes.add(vertex);  
 adjencyList.add(new ArrayList<Vertex>());  
 }  
  
 public void deleteVertex(String Char){  
 int indxFoundVertex = -1;  
 boolean found = false;  
 for (int i = 0 ; i < currAmountVertex + 1; i++) {  
 if(Char.equals(vertexes.get(i).getChar())) {  
 ++indxFoundVertex;  
 found = true;  
 break;  
 }  
 }  
 if(!found) {  
 System.*out*.println("Немає такої вершини!");  
 return;  
 }  
 --currAmountVertex;  
 Vertex deleted = vertexes.remove(indxFoundVertex);  
 List<Vertex> deletedV = adjencyList.remove(indxFoundVertex);  
 }  
  
 private ArrayList<Vertex> findNeighbours(Vertex node) {  
 ArrayList<Vertex> neighbours = new ArrayList<>();  
 int nodeIndex = vertexes.indexOf(node); //індекс пошуку для вузла  
  
 if(nodeIndex != -1) //додавання сусідів вузлів до списку  
 for (Vertex i : adjencyList.get(nodeIndex))  
 neighbours.add(i);  
  
 return neighbours;  
 }  
  
 public void DFS(int start) throws ArrayIndexOutOfBoundsException{  
  
 if(start > vertexes.size() || start < 0 ) //виняток для неправильного номера вузла  
 throw new ArrayIndexOutOfBoundsException("Поза межами");  
  
 Stack<Vertex> stack = new Stack<>(); //стек для зберігання сусідніх та невідвіданих вузлів  
 stack.add(vertexes.get(start)); //почніть з пуску стартового вузла  
  
 while (!stack.isEmpty()) {  
 Vertex element = stack.pop(); //вимкнути вузол зі стеку та надрукувати, якщо його не відвідали  
 if(!element.wasVisited) {  
 System.*out*.print(element.getChar() + " -> ");  
 element.wasVisited = true;  
 }  
  
 List<Vertex> neighbours= findNeighbours(element); //отримати сусідні вузли  
 Collections.*reverse*(neighbours); //переверніть список, щоб перенести вузли в стек у правильному порядку  
  
 for (Vertex nodeIter : neighbours) { //штовхання невідвіданих сусідніх вузлів  
 if(nodeIter != null && !nodeIter.wasVisited)  
 stack.add(nodeIter);  
 }  
 }  
 System.*out*.println();  
  
 //набір, не відвідуваний для всіх вузлів  
 for(Vertex nodeIter : vertexes)  
 nodeIter.wasVisited = false;  
 }  
  
 public void BFS(int start) throws ArrayIndexOutOfBoundsException {  
  
 if(start > vertexes.size() || start < 0 ) //виняток для неправильного номера вузла  
 throw new ArrayIndexOutOfBoundsException("Поза межами");  
  
 Queue<Vertex> queue = new LinkedList<>(); //черга для зберігання сусідніх та невідвіданих вузлів  
 vertexes.get(start).wasVisited = true; //почніть з пуску стартового вузла  
 queue.add(vertexes.get(start));  
  
 while (!queue.isEmpty()) {  
 Vertex element = queue.remove();  
 System.*out*.print(element.getChar() + " -> ");  
  
 ArrayList<Vertex> neighbours = findNeighbours(element); //отримати сусідні вузли  
 for (Vertex nodeIter : neighbours) { //штовхання невідвіданих сусідських вузлів  
 if(nodeIter!=null && !nodeIter.wasVisited) {  
 queue.add(nodeIter);  
 nodeIter.wasVisited = true;  
 }  
 }  
 }  
 System.*out*.println();  
 //набір, не відвідуваний для всіх вузлів  
 for(Vertex nodeIter : vertexes)  
 nodeIter.wasVisited = false;  
 }  
  
 public void print(){  
 for (Vertex v : vertexes)  
 System.*out*.print(v.getChar() + " ");  
 System.*out*.println(" ");  
  
 for(int i = 0; i < currAmountVertex + 1; i++) {  
 System.*out*.print(vertexes.get(i).getChar() + " ");  
 for (Vertex v : adjencyList.get(i))  
 System.*out*.print(v.getChar() + " ");  
 System.*out*.println();  
 }  
 }  
  
 public void SetVisit(boolean res){  
 for (Vertex v : vertexes)  
 v.wasVisited = res;  
 }

}

Main.java

package com.company;  
  
import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
 int menu;  
   
 Graph graph = new Graph(6);  
 Graph graph2 = new Graph(6);  
  
 graph2.addVertex(new Vertex("A"));  
 graph2.addVertex(new Vertex("B"));  
 graph2.addVertex(new Vertex("C"));  
 graph2.addVertex(new Vertex("D"));  
 graph2.addVertex(new Vertex("E"));  
 graph2.addVertex(new Vertex("F"));  
  
 graph2.addEdge("A", "B");  
 graph2.addEdge("A", "C");  
 graph2.addEdge("C", "F");  
 graph2.addEdge("B", "E");  
 graph2.addEdge("E", "F");  
 graph2.addEdge("D", "B");  
 graph2.addEdge("F", "B");  
 graph2.addEdge("E", "D");  
  
 graph2.print();  
 System.*out*.println("------------------------------------");  
 graph2.removeEdge("C", "F");  
 graph2.removeEdge("A", "B");  
  
 graph2.print();  
 System.*out*.println("------------------------------------");  
 graph2.deleteVertex("A");  
  
 graph2.print();  
 System.*out*.println("------------------------------------");  
 System.*out*.println("DFS");  
  
 graph2.DFS(3);  
 System.*out*.println("");  
 graph2.SetVisit(false);  
  
 System.*out*.println("------------------------------------");  
 System.*out*.println("BFS");  
  
 graph2.BFS(2);  
 graph2.SetVisit(false);  
  
 do{  
 System.*out*.println("1.Створення графа");  
 System.*out*.println("2.Додавання вершини");  
 System.*out*.println("3.Додавання ребра");  
 System.*out*.println("4.Відалення дуги");  
 System.*out*.println("5.Обхід в глибину");  
 System.*out*.println("6.Обхід в ширину");  
 System.*out*.println("7.Вивід матрииці суміжності");  
 System.*out*.println("8.Вихід");  
 Scanner scanner = new Scanner(System.*in*);  
 menu = scanner.nextInt();  
  
 switch (menu) {  
 case 1 -> {  
 graph.addVertex(new Vertex("A"));  
 graph.addVertex(new Vertex("B"));  
 graph.addVertex(new Vertex("C"));  
 graph.addVertex(new Vertex("D"));  
 graph.addVertex(new Vertex("E"));  
 graph.addVertex(new Vertex("F"));  
  
 graph.addEdge("A", "B");  
 graph.addEdge("A", "C");  
 graph.addEdge("C", "F");  
 graph.addEdge("B", "E");  
 graph.addEdge("E", "F");  
 graph.addEdge("D", "B");  
 graph.addEdge("F", "B");  
 graph.addEdge("E", "D");  
 }  
 case 2 -> {  
 String vertex;  
 Scanner scannerVertex = new Scanner(System.*in*);  
 vertex = scannerVertex.nextLine();  
 graph.addVertex(new Vertex(vertex));  
 }  
 case 3 -> {  
 String start, end;  
 System.*out*.println("Введіть початок: ");  
 Scanner scannerStart = new Scanner(System.*in*);  
 start = scannerStart.nextLine();  
 System.*out*.println("Введіть кінець: ");  
 Scanner scannerEnd = new Scanner(System.*in*);  
 end = scannerEnd.nextLine();  
 graph.addEdge(start, end);  
 }  
 case 4 -> {  
 System.*out*.println("Видаляемо ребро C, F");  
 System.*out*.println("Видаляемо ребро A, B");  
 graph.removeEdge("C", "F");  
 graph.removeEdge("A", "B");  
 graph.print();  
 }  
 case 5 -> {  
 graph.BFS(2);  
 //System.out.println();  
 }  
 case 6 -> {  
 graph.DFS(3);  
 //System.out.println();  
 }  
 case 7 -> graph.print();  
 }  
  
 } while (menu != 8);  
 }  
}

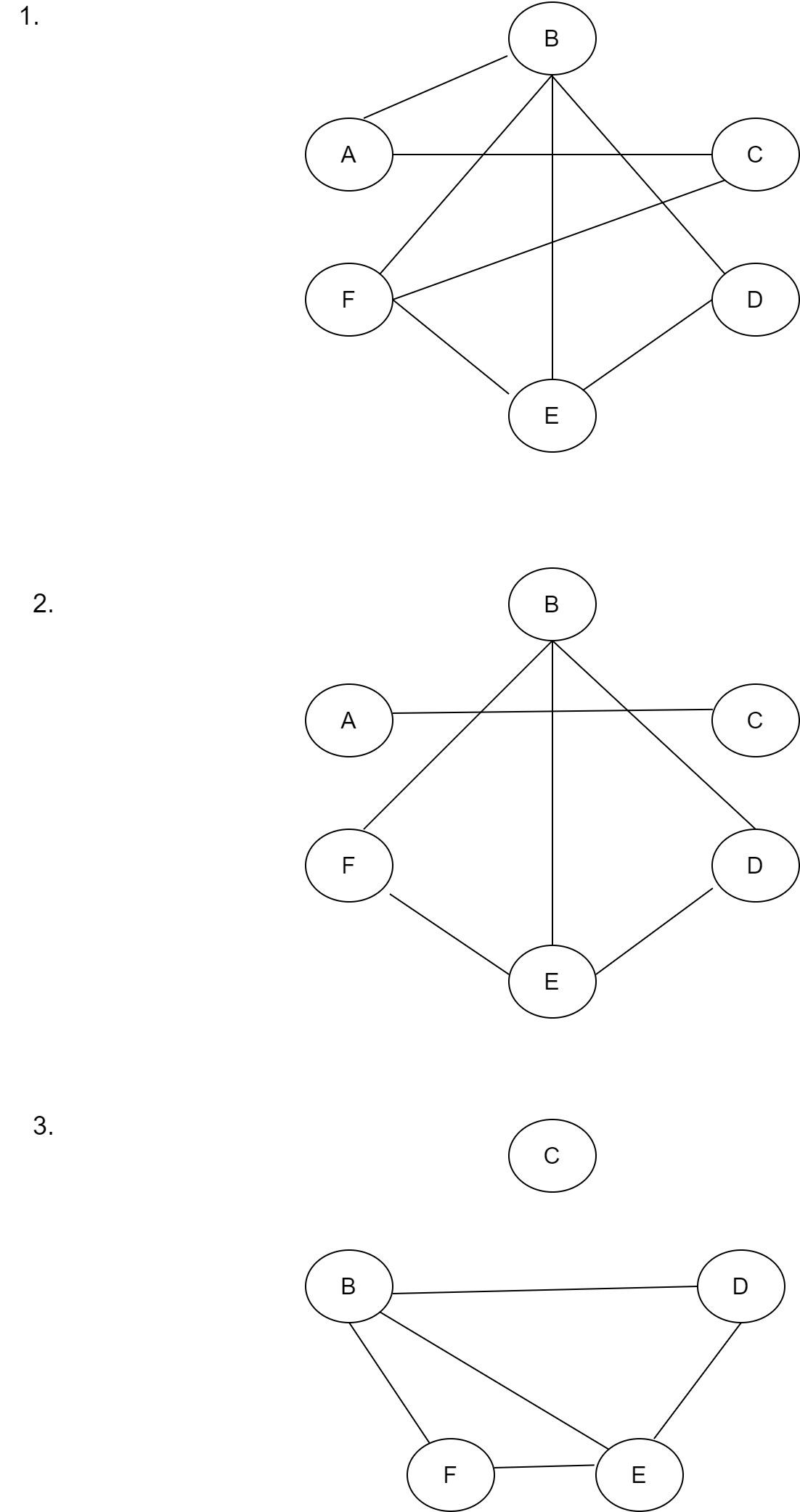


Рисунок 1. Представлення графа графічно.

Обхід в ширину (з вершини D, порядок вершин з право-наліво)

|  |  |
| --- | --- |
| D | B |
| B | E, B |
| E | F, E, B, D |

Обхід в глиб (з вершини E, порядок вершин зліва-направо)

|  |  |
| --- | --- |
| E | F |
| E | D, F |
| F | E, B, D, F |

Висновок: отже, на практиці ми з’ясували, що граф має безліч застосувань у різних сферах життя. Можна використовувати як у реалізації позиційних ігор (шахи, шашки, «хрестики-кулики»), у розробці генеалогічного дерева, у розробці схем авіаліній, метрополітена, хімії (структура формул хімічного з’єднання), а також дослідження лабіринту.

Слід відмітити, що у реалізації графу двома способами, використання списку є більш доцільним й універсальним. У випадку матриці, розрахунок йде на більш статичні розміри тієї ж матриці, проте легше працювати із індексованими даними. Для реалізації загального функціоналу, різниці як такої немає.орієнтований