ЗМІСТ

Мета роботи: Вивчення основних властивостей графів, способів подання графів, шляхів та циклів, обходу графі, розфарбовування графів, набуття практичних навичок програмування алгоритмів, що базуються на графах.

ВСТУП

Теорія графів – один з найбільш важливих розділів дискретної математики, що використовується для розв’язання широкого спектру завдань інформатики та кібернетики, наприклад: проектування та оптимізація комп’ютерних систем, дослідження автоматів та складно організованих систем, розв’язання транспортних задач, встановлення взаємозалежності об’єктів предметної області, тощо.

1. ОСНОВНІ ОЗНАЧЕННЯ ТА ВЛАСТИВОСТІ

Графом G = (V, E) називається об’єкт, який заданий парою множин (V, E), де V – множина вершин, $E \subseteq V \times V$ – множина ребер.

Множину вершин графу G позначають V(G), а множину ребер – E(G). Кількість вершин графу n(G) = |V(G)|, а кількість ребер m(G) = |E(G)|.

Граф називається *скінченним*, якщо множини його вершин і ребер є скінченними.

Кількість вершин n(G) графу називають його *порядком*.

Якщо для деякого ребра $e = (v,w) \in E(G)$, то кажуть (див. рис. 1):

- вершини v та w суміжні;
- вершини v та w інцидентні ребру e;
- ребро e інцидентне вершинам v і w;

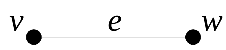


Рис.1.

Неорієнтований граф — це граф, для кожного ребра якого несуттєвий порядок двох його кінцевих вершин.

Орієнтований граф — це граф, для кожного ребра якого істотний порядок двох його кінцевих вершин. Ребра орієнтованого графа називають дугами.

Змішаний граф – це граф, що містить як орієнтовані, так і неорієнтовані ребра. Кожної з перерахованих видів графа може містити одне або кілька ребер, у яких обидва кінці сходяться в одній вершині, такі ребра називаються *петлями*

Основні різновиди кінцевих графів (див. рис. 2):

1) граф без петель та кратних ребер називається *простим* (звичайним);
2) граф без петель, але з кратними ребрами називається *мультографом*;
3) найбільш загальний вид граф з петлями та кратними ребрами називається *псевдографом*;
4) граф з орієнтованими ребрами (дугами) і петлями та без кратних ребер

називається *простим орієнтованим графом*;

5) граф з орієнтованими ребрами (дугами) та кратними ребрами і петлями називається *орієнтованим мультиграфом*;

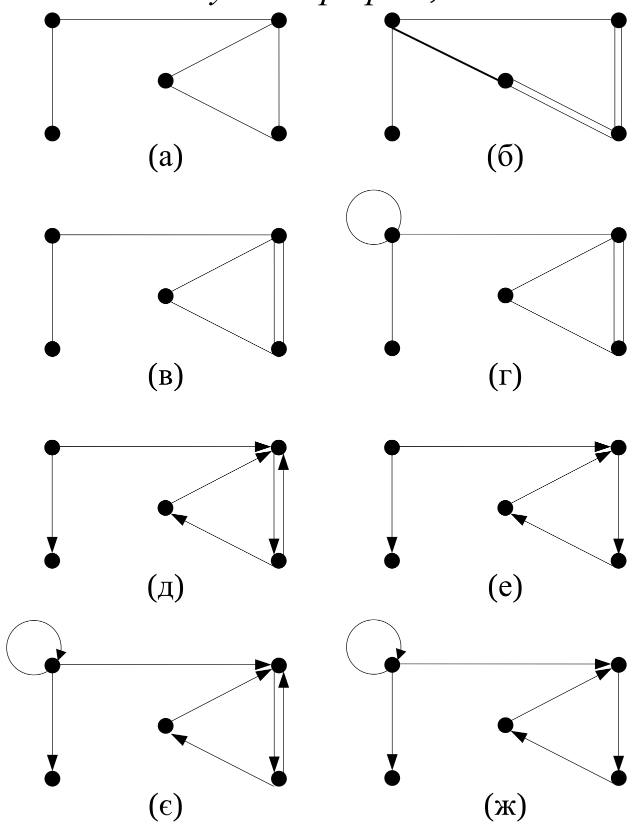


Рис. 2. Типи графів.

а) простий граф; б,в) мультограф; г) псевдограф; е,ж)орієнтований граф; д,є) орієнтований мультиграф

Степінь вершини в неорієнтованому графі – це кількість інцидент них їй ребер. Степінь вершини v позначають $\deg(v)$.

Приклад. У неорієнтованому графі на рис. 3. Степені вершини є наступними: $\deg(a)=1, \deg(b)=4$, $\deg(c)=0$, $\deg(e)=3$, $\deg(d)=2$. Вершина типу $\deg(a)=1$ називається *висячою,* а типу $\deg(c)=0$ – *ізольованою.*

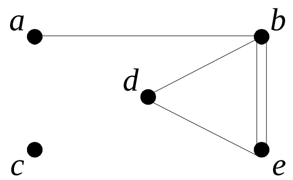


Рис. 3.

1.1. Способи подання графів

Існує декілька основних способи подання графа: графічний, матричний, списком пар (списком ребер) та списками суміжності. Розглянемо три перші способи.

Графічний спосіб є найбільш зрозумілим для людини поданням графа, проте його неможливо використовувати при комп’ютерному розв’язанні задач (див. рис. 2).

Найпопулярнішим способом комп'ютерного подання графа є матричний. Розглядають способи за допомогою матриці суміжності та матриці інцидентності.

1) Матриця суміжності вершин

Для графа $G(V)$ (позначається $M(G) = \{M_{ij}\}$) - це квадратна матриця $\Delta\|\delta ij\|$, в якій стовпцям і рядкам відповідають вершини графу. M_{ij} - кількість ребер, які з'єднують V_i з V_j в графі G.

Якщо граф G неорієнтований, то $M_{ij} = M_{ji}$, тобто матриця M є симетричною, для орієнтованого – цей елемент матриці відповідає кількості ребер з початком в i-й вершині та кінцем у j-й вершині.

Для орієнтованого графа: якщо дуга «входить» присвоюємо значення 0,в іншому випадку – 1.

2) Матриця інцедентності - булева матриця з елементами $v_1, v_2, \ldots, v_n$ – вершини графу G; $e_1, e_2, \ldots, e_m$ – його ребра.

Відношення інцидентності можна означити матрицею $E = \|\varepsilon ij\|$, яка має n рядків - вершин та m стовпців-ребер.

Для неорієнтованого графа якщо ребро e_j є інцидентним вершині v_i і $\varepsilon_{ij} = 1$, в іншому випадку $\varepsilon_{ij} = 0$.

У матриці інцидентності $\|\varepsilon ij\|$ орієнтованого графу, якщо вершина vi – початок дуги e_j , то $\varepsilon_{ij} = -1$, якщо v_i – кінець - $\varepsilon_{ij} = 1$; якщо e_j – петля, а v_i – інцидентна їй вершина, то $\varepsilon_{ij} = 2$.

1.2. Шляхи та цикли

Шляхом з вершини m у вершину n називають таку послідовність ребер, що веде з m до n, у якій кожні дві сусідніх ребра мають спільну вершину і ніяке ребро не зустрічається більш, ніж один раз.

Шлях з вершини m у вершину n називається простим, коли він не проходить через жодну вершину графа більш, ніж один раз. Циклом називається шлях, у якому співпадають його початкова і кінцева вершина.

Простим циклом у графі називається цикл, що не проходить через жодну вершину більше одного разу.

Ейлеревим шляхом у графі називається шлях, що містить всі ребра графа.

Ейлеревим циклом у графі називається цикл, що містить всі ребра графа. Схожим чином, ейлерів цикл — ейлерів шлях, що починається і завершується в одній вершині.

Граф, що має ейлеровий цикл, називається *ейлеревим графом*. Зв'язаний граф або мультиграф G має ейлерів цикл тоді й тільки тоді, коли степені всіх його вершин парні. Якщо мультограф має 2 вершини непарного степеня, то граф має тільки ейлерів шлях.

Гамільтонів шлях — шлях, що містить кожну вершину графа рівно один раз.

Гамільтонів шлях, якщо початкова і кінцева вершини графа збігаються, називається гамільтоновим циклом.

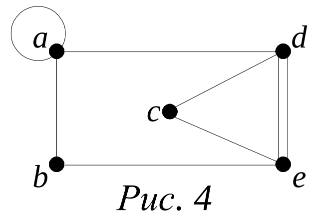


Рис. 4

Приклад. На рис. 4 подано неорієнтований псевдограф *G*. Дано шляхи: $P_1 = a,d,e,c,d,e,b,a,a$; $P_2 = a,a,d,e,c,d,e,b$; $P_3 = a,d,c,e,b,a$; $P_4 = a,b,e,d,c$. Вірними є наступні твердження: P_1 – ейлерів цикл, P_2 – ейлерів шлях, P_3 – гамільтонів цикл, P_4 – гамільтонів шлях.

2. ОБХІД ГРАФІВ

Виділяється два основних алгоритми обходів графів або пошуку у графах – це пошук углиб та пошук вшир.

Пошуку углиб або DFS-методу (Depth First Search).

НехайG=(V, E)– простий зв'язний граф, усі вершини якого позначено попарно різними символами. У процесі пошуку вглиб вершинам графа G надають номери (DFS-номери) та певним чином позначають ребра. У ході роботи алгоритму використовують структуру даних для збереження множин, яку називають стеком. Зі стеку можна вилучити тільки той елемент, який було додано до нього останнім: стек працює за принципом «останнім прийшов – першим вийшов» (last in, first out – скорочено LIFO). Інакше кажучи, додавання й вилучення елементів у стеку відбувається з одного кінця, який називається верхівкою стеку. DFS-номер вершиних позначають DFS(x).

Алгоритм пошуку вглиб у простому зв'язаному графі:

1. Почати з довільної вершини v_s. Виконати DFS(v_s) :=1. Включити цю вершину в стек.
2. Розглянути вершину у верхівці стеку: нехай це вершина x. Якщо всі ребра, інцидентні вершиніх, позначено, то перейти до кроку4, інакше – до кроку 3.
3. Нехай (x,y) – ребро, в якомуDFS(y)не визначено, то позначити ребро(x,y) потовщеною суцільною лінією, визначити DFS(y), як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.
4. Виключити вершину хіз стеку. Якщо стек порожній, то зупинитись, інакше – перейти до кроку 2

Пошуку вшир вершини графа проглядають в іншій послідовності, ніж у методі пошуку вглиб, і їм надають BFS-номери (breadth first search). BFS-номер вершини x позначають відповідно BFS(x). Під час пошуку рухаються вшир, а не вглиб: спочатку проглядають усі сусідні вершини, після цього – сусіди сусідів і так далі.

У ході реалізації алгоритму використовують структуру даних для збереження множин, яку називають чергою. Із черги можна вилучити тільки той елемент, який перебував у ній найдовше: працює принцип «першим прийшов – першим вийшов» (first in, first out – скорочено FIFO). Елемент включається у хвіст черги, а виключається з її голови. Пошук ушир, узагалі кажучи, відрізняється від пошуку вглиб заміною стеку на чергу. Після такої модифікації що раніше відвідується вершина (включається в чергу), то раніше вона використовується (і виключається з черги). Використання вершини полягає в перегляді одразу всіх ще не відвіданих її сусідів. Усю процедуру подано нижче.

Алгоритм пошуку вшир у простому зв'язаному графі:

1. Почати з довільної вершини v_s. Виконати BFS(v_s):=1. Включити вершину v_s у чергу.
2. Розглянути вершину, яка перебуває на початку черги; нехай це буде вершина x. Якщо для всіх вершин, суміжних із вершиною x, вже визначено BFS-номери, то перейти до кроку 4, інакше – до кроку 3.
3. Нехай (x,y)– ребро, у якому номер BFS(y) не визначено. Позначити це ребро потовщеною суцільною лінією, визначити BFS(y), як черговий BFS-номер, включити вершину цю у чергу й перейти до кроку 2.
4. Виключити вершину x із черги. Якщо черга порожня, то зупинитись, інакше – перейти до кроку 2.

2.1. Розфарбовування графа

Розфарбуванням простого графа G називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору (значення). Найменшу можливу кількість кольорів у розфарбуванні графа називають *хроматичним числом* і позначають $\chi(G)$.

Теорема. Якщо найбільший зі степенів вершин графа дорівнює n, то цей граф можливо розфарбувати в $n+1$ колір.

Приклад розфарбування графа G подано на рис. 5. Хроматичне число поданного графа: $\chi(G)=3$.

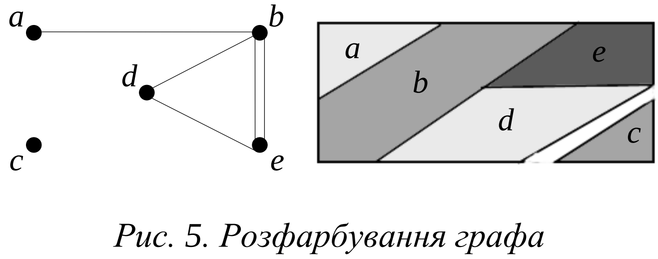


Рис. 5. Розфарбування графа

2.2. Приклад програми побудови матриці суміжності простого графа

```
#include <stdio.h>
#include <conio.h>
#include <iostream>
#include <math.h>
using namespace std;

void main()
{
	int x, y, w, chromatic;
	int summond[6];
	int m[15] = { 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 5 };
	int n[15] = { 2, 3, 4, 5, 6, 3, 4, 5, 6, 4, 5, 6, 5, 6, 6 };
	int contiguity[6][6];
	int incidence[6][11];
	int Rmn[15];
	cout << "Input values:\n";

	for (int i = 0; i <= 14; i++)
	{
		cin >> Rmn[i];
	}

	int k = 0, j = 0, z = -1;
	for (int i = 0; i <= 5; i++)
	{
		z += 1;
		for (j = z; j <= 5; j++)
		{
			if (i == j)
			{
				contiguity[i][j] = 0;
			}
			else
			{
				contiguity[i][j] = Rmn[k];
				contiguity[j][i] = Rmn[k++];
			}
		}
	}
	cout << "\nMatrix of contiguity:\n\n";
	for (int i = 0; i <= 5; i++)
		for (int k = 0, j = 0; j <= 5; j++)
		{
			cout << contiguity[i][j] << " ";
			if (j == 5) cout << "\n";
		}

	for (int i = 0; i <= 5; i++)
		for (int j = 0; j <= 10; j++)
			incidence[i][j] = 0;
	for (int i = 0, k = 0; i <= 14; i++)
	{
		x = Rmn[i];
```

```
			y = m[i] - 1;
			w = n[i] - 1;

			while (x > 0)
			{
				incidence[y][k] = 1;
				incidence[w][k] = 1;
				k++;
				x--;
			}
		}

			_getch(); }
```

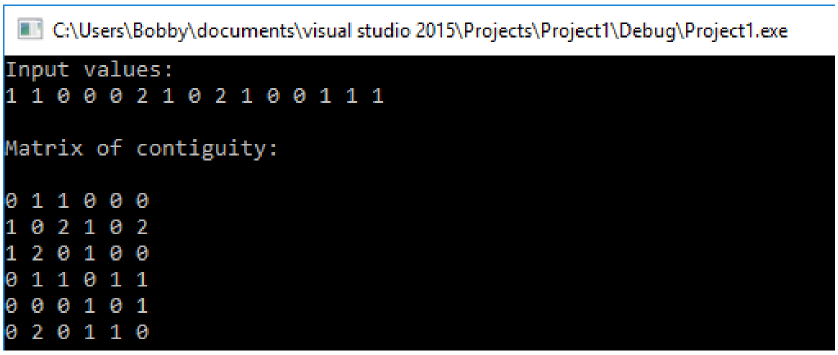


Рис. 6. Результат роботи програми

3. АЛГОРИТМ ДЕЙКСТРИ

Найефективніший алгоритм визначення довжини найкоротшого шляху від фіксованої вершини до будь-якої іншої запропонував 1959 р. датський математик Е. Дейкстра (E. Dijkstra). Цей алгоритм застосовний лише тоді, коли вага кожного ребра (дуги) додатна. Опишемо докладно цей алгоритм для орієнтованого графа.

Нехай $G=(V,E)$ – зважений орієнтований граф, $\omega(v_i,v_j)$ – вага дуги v_i,v_j. Почавши з вершини *a*, знаходимо віддаль від *a* до кожної із суміжних із нею вершин. Вибираємо вершину, віддаль від якої до вершини а найменша; нехай це буде вершина v*. Далі знаходимо віддалі від вершини а до кожної вершини суміжної з v* вздовж шляху, який проходить через вершину v*. Якщо для якоїсь із таких вершин ця віддаль менша від поточної, то замінюємо нею поточну віддаль. Знову вибираємо вершину, найближчу до *a* та не вибрану раніше; повторюємо процес.

Описаний процес зручно виконувати за допомогою присвоювання вершинам міток. Є мітки двох типів: тимчасові та постійні. Вершини з постійними мітками групуються у множину М, яку називають множиною позначених вершин. Решта вершин має тимчасові мітки, і множину таких вершин позначимо як Т, Т= V \ М. Позначатимемо мітку (тимчасову чи постійну) вершини v як $l(v)$. Значення постійної мітки $l(v)$ дорівнює довжині найкоротшого шляху від вершини a до вершини v, тимчасової – довжині найкоротшого шляху, який проходить лише через вершини з постійними мітками. Фіксованою початковою вершиною вважаємо вершину a; довжину найкоротшого шляху шукаємо до вершини z (або до всіх вершин графа).

Тепер формально опишемо ***алгоритм Дейкстри***:

1. Присвоювання початкових значень. Виконати $l(a) = 0$ та вважати цю мітку постійною. Виконати $l(v) = \infty$ для всіх $v \neq a$ й уважати ці мітки тимчасовими. Виконати $x = a$, $M = a$.
2. Оновлення міток. Для кожної вершини $v \in \Gamma(x) \setminus M$ замінити мітки: $l(v) = \min\{l(v); l(x) + \omega(x, v)\}$, тобто оновлювати тимчасові мітки вершин, у які з вершини x іде дуга.
3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину v* з умови $l(v^*) = \min\{l(v)\}$, $v \in T$, де $T = V \setminus M$.
5. Уважати мітку вершини v* постійною й виконати $M = M \cup \{v^*\}$; $x = v^*$ (вершину v* включено в множину M).
6. а) Для пошуку шляху від a до z: якщо $x = z$, то $l(z)$ – довжина найкоротшого шляху від a до z, зупинитись; якщо $x \neq z$, то перейти до кроку 2.
4. б) Для пошуку шляхів від a до всіх вершин: якщо всі вершини отримали постійні мітки (включені в множину M), то ці мітки дорівнюють довжинам найкоротших шляхів, зупинитись; якщо деякі вершини мають тимчасові мітки, то перейти до кроку 2.

5. ЗАВДАННЯ

Порядок виконання роботи: Скласти комп'ютерні програми із зазначеними вхідними даними та результатами для завдання 1.

Завдання 1: Неорієнтований граф на 6 вершинах заданий вектором R_{mn} (табл. 1), де m та n – номера вершин графа, m=$\overline{1,6}$ та n=$\overline{1,6}$. Елементи вектора R_{mn} відповідають кількості ребер між відповідними вершинами m та n. Для заданого графа необхідно:

1. Побудувати матрицю суміжності та матрицю інцидентності для заданого графа. Намалювати граф.

2. Визначити тип графа.

3. Виписати усі ейлерові та гамільтонові ланцюги та цикли (якщо є). Відповідь обґрунтувати.

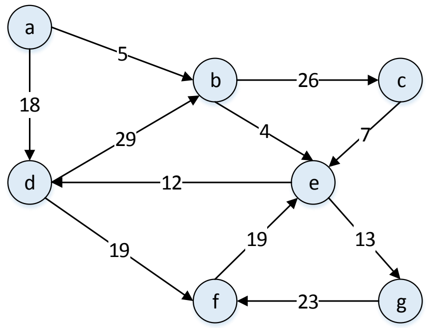
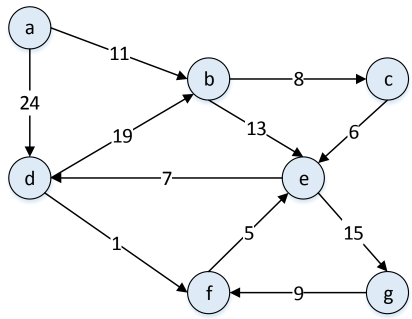
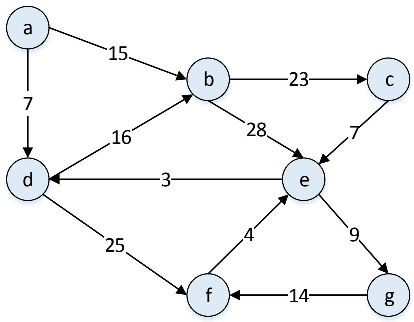
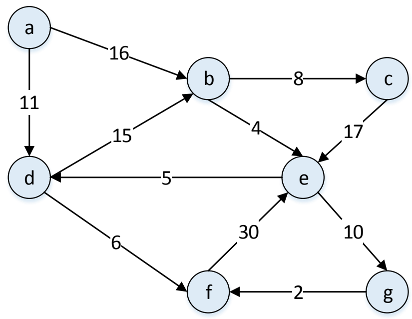
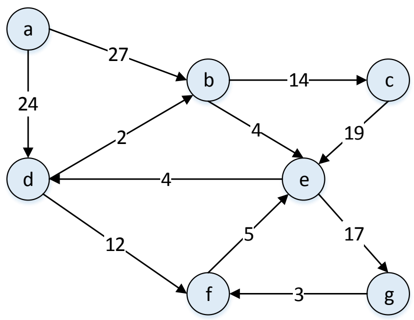
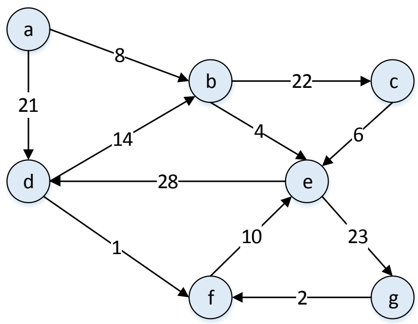
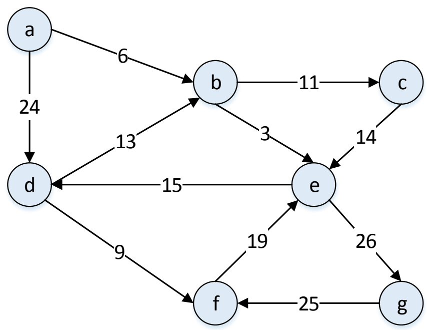
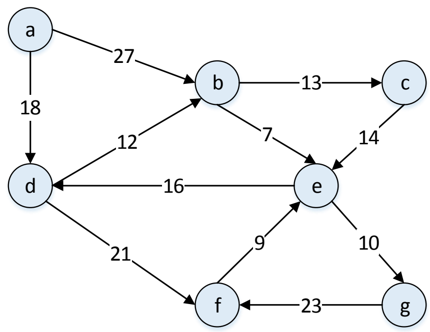
4. Визначити хроматичне число та реберне хроматичне число графа.
5. Розфарбувати вершини та ребра графа.

Таблиця 1. Варіанти задання графа

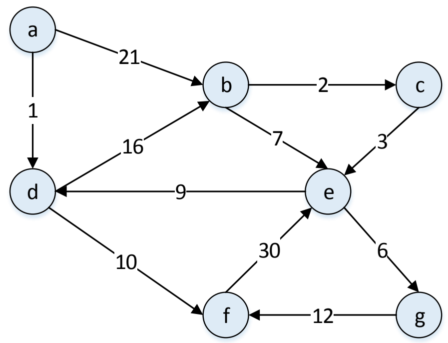
Варіант	m	1	1	1	1	1	2	2	2	2	3	3	3	2	2	5
	n	2	3	4	5	6	3	4	5	6	4	5	6	5	6	6
1	Rmn	1	2	1	1	1	0	0	0	1	1	0	0	1	1	1
2	Rmn	0	0	1	1	1	0	1	0	1	0	1	0	0	0	2
3	Rmn	1	0	0	1	0	0	1	1	1	0	0	0	1	0	1
4	Rmn	1	0	1	0	0	1	1	0	2	0	0	1	0	2	0
5	Rmn	2	1	1	0	0	1	1	0	1	0	0	0	1	1	0
6	Rmn	1	0	0	1	0	1	0	0	0	1	0	0	1	2	2
7	Rmn	1	1	1	0	0	0	2	0	2	0	0	1	0	1	0
8	Rmn	0	2	0	2	0	1	0	0	0	0	1	0	1	1	1
9	Rmn	1	2	1	0	0	0	0	0	1	1	1	0	1	0	0
10	Rmn	1	1	1	1	0	0	0	0	1	0	1	0	1	1	1
11	Rmn	1	2	0	0	1	1	1	0	0	2	1	0	1	0	0
12	Rmn	0	0	0	1	1	1	0	0	2	0	0	1	1	0	2
13	Rmn	0	1	0	1	0	0	1	2	0	1	0	0	0	1	1
14	Rmn	1	1	0	0	0	1	0	1	1	0	1	1	1	0	1
15	Rmn	1	0	2	0	2	0	0	0	1	0	0	0	1	0	1
16	Rmn	1	1	0	0	0	1	1	0	1	0	0	0	1	1	2
17	Rmn	1	0	0	0	0	1	1	2	0	0	0	1	1	0	1
18	Rmn	2	1	0	1	0	1	1	0	0	0	1	0	1	1	1
19	Rmn	1	1	0	0	0	2	1	0	2	1	0	0	1	1	1
20	Rmn	2	0	1	1	0	0	1	0	1	0	0	0	0	2	1
21	Rmn	1	1	0	0	0	1	1	0	1	1	0	1	1	1	0
22	Rmn	2	0	0	0	0	1	1	1	1	0	1	0	1	0	0
23	Rmn	1	1	1	1	0	1	1	0	1	0	0	0	1	1	0
24	Rmn	0	0	1	0	1	2	1	0	1	1	0	1	0	1	0
25	Rmn	2	0	1	0	1	1	0	0	1	0	0	1	1	0	1
26	Rmn	0	2	1	0	1	0	0	1	1	0	0	0	0	1	1
27	Rmn	0	1	0	1	0	1	1	0	0	1	1	0	1	1	1
28	Rmn	2	0	1	1	0	1	1	0	0	0	1	0	1	1	1
29	Rmn	1	0	1	0	2	1	1	1	0	0	0	1	1	1	0
30	Rmn	1	1	0	0	0	2	0	1	2	1	0	0	1	0	0

Завдання 2: За алгоритмом Дейкстри знайти найкоротші віддалі від вершини *a* до вершини *g* графа з табл. 2.

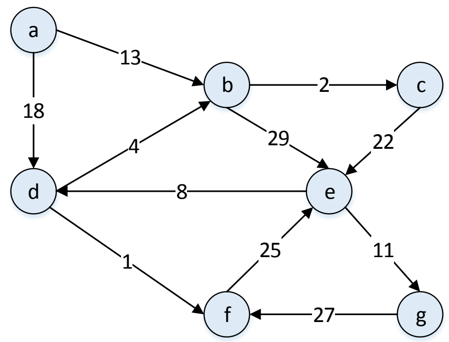
Таблиця 2. Варіанти задання графа

№	Орієнтований граф	№	Орієнтований граф
1.		2.	
3.		4.	
5.		6.	
7.		8.	

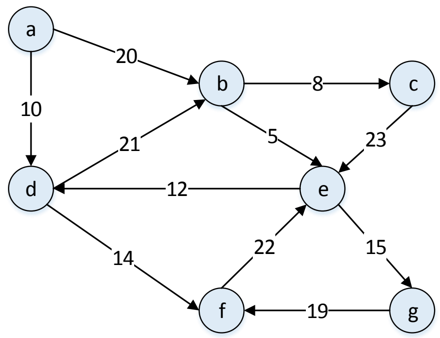
9.



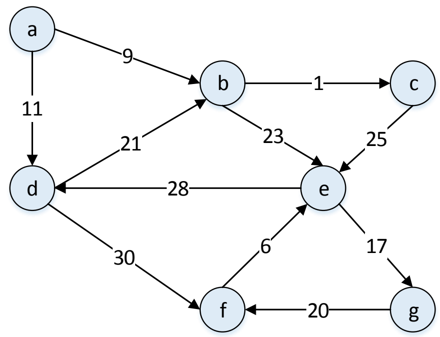
10.



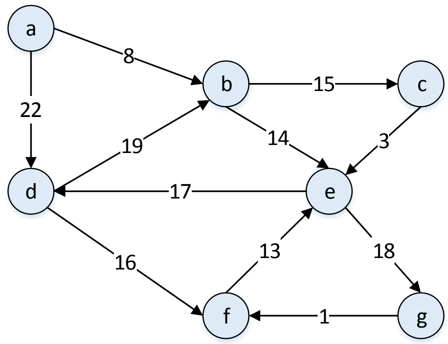
11.



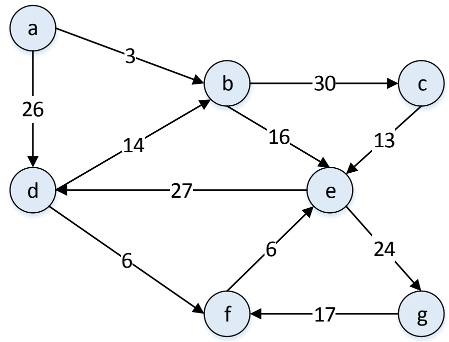
12.



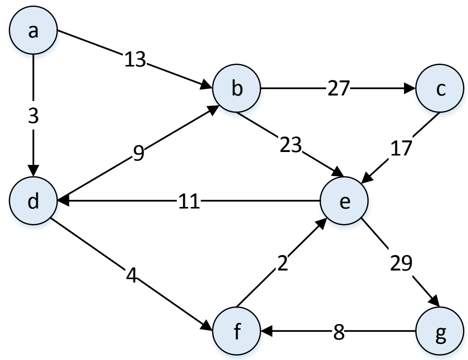
13.



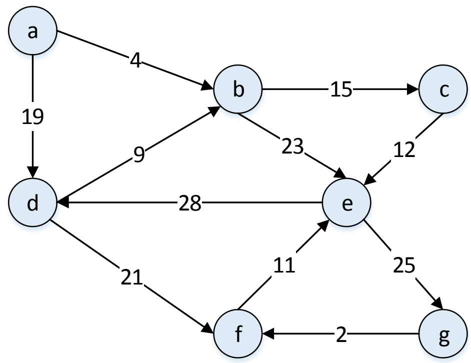
14.



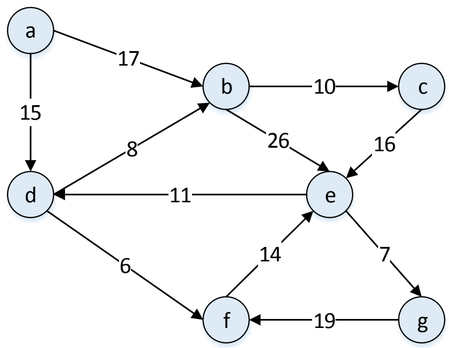
15.



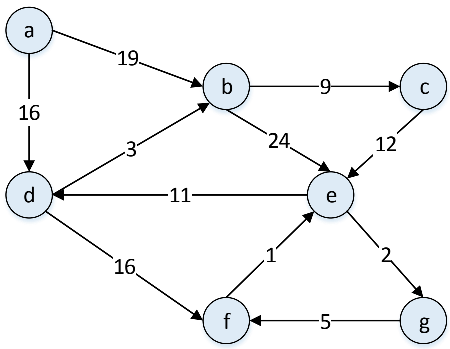
16.



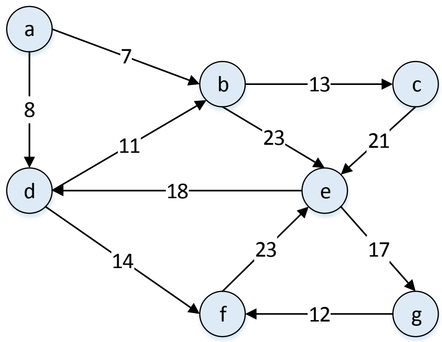
17.



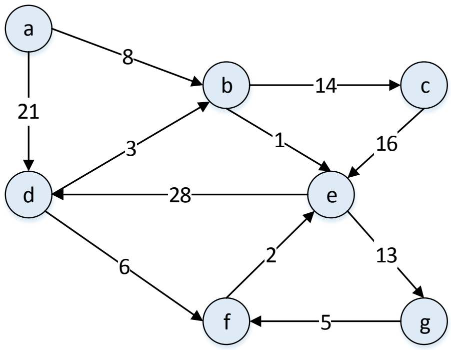
18.



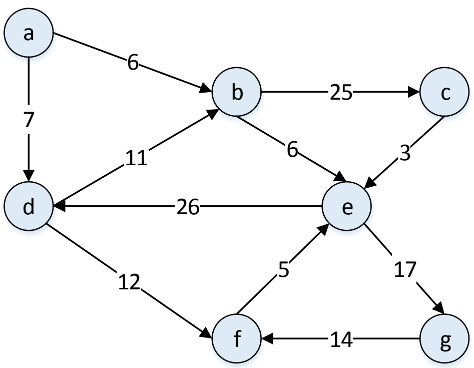
19.



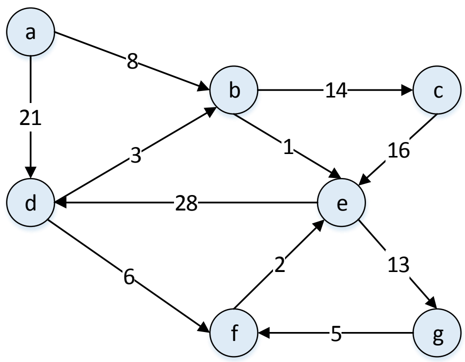
20.



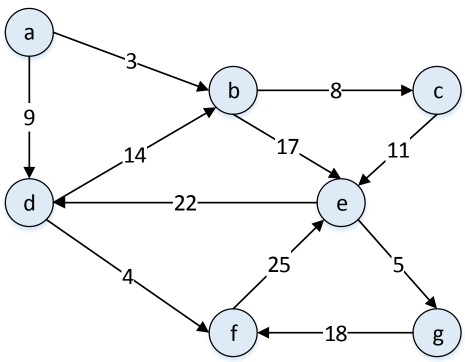
21.



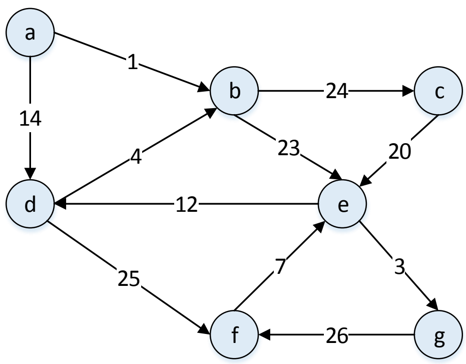
22.

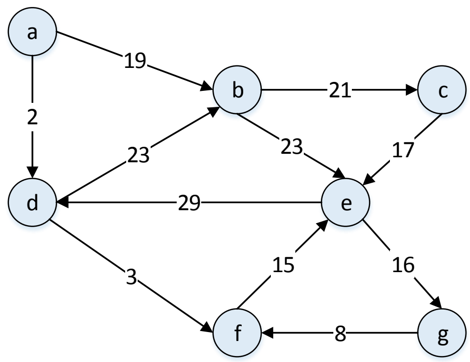
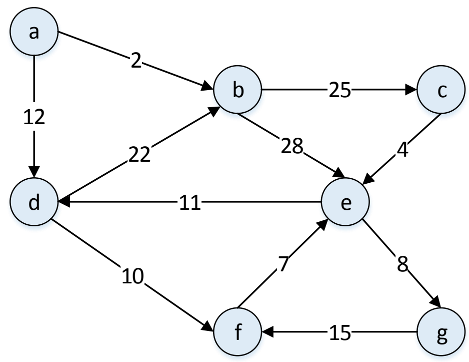
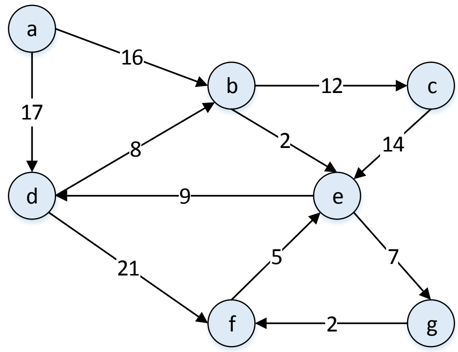
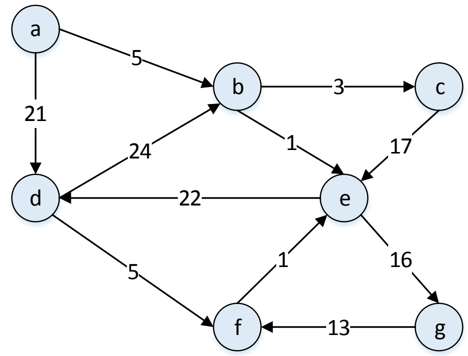
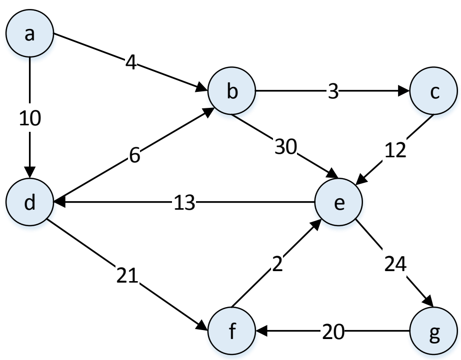
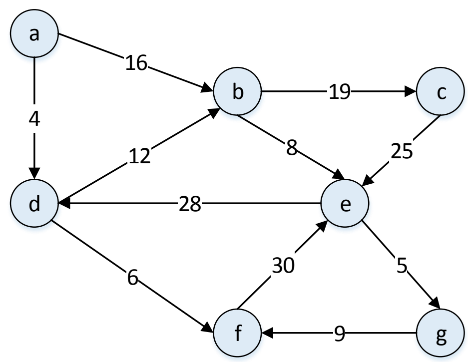


23.



24.



<table>
<tr><td>25.</td><td>
</td><td>26.</td><td>
</td></tr>
<tr><td>27.</td><td>
</td><td>28.</td><td>
</td></tr>
<tr><td>29.</td><td>
</td><td>30.</td><td>
</td></tr>
</table>

6. ВИМОГИ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Кожен студент отримує набір завдань відповідно до свого порядкового номеру у списку групи або відповідно до номеру залікової книжки.
2. Звіт про виконання роботи оформляються у вигляді завдань та розв’язку до них.
3. Звіт акуратно оформляється на аркушах А4 та скріпляються скріпкою.
4. Звіт про виконання лабораторної роботи необхідно захистити у строго визначені терміни.

7. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Основні властивості графів,
2. Способи подання графів,

3. Шляхи та цикли (ейлерів цикл, гамільтонів цикл).

4. Обхід графів, розфарбовування графів.

8. ЗМІСТ ЗВІТУ ПО РОБОТІ

1. Назва роботи.

2. Мета роботи.

Теоретична частина (основні властивості графів, способи подання графів, шляхи та цикли, обхід графів, розфарбовування графів.

3. Опис виконаної роботи та отриманих результатів (запрограмувати завдання 1):

- завдання;
- текст програми;
- результати виконання програми;

4. Висновки.

Додаток А
ТИТУЛЬНА СТОРІНКА ЗВІТУ ДО ЛАБОРАТОРНОЇ РОБОТИ

Міністерство освіти і науки України

Національний університет "Львівська політехніка"

Кафедра інформаційних

систем та мереж

ДИСКРЕТНА МАТЕМАТИКА

Звіт

до лабораторної роботи №__

____

(назва лабораторної роботи)

Виконав:

студент гр.________

(назва групи)

____

(прізвище та ініціали студента)

Прийняв:

____

(прізвище та ініціали викладача)

Львів – 20__