ОСНОВИ ПРОГРАМУВАННЯ

Частина 3

2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

ОСНОВИ ПРОГРАМУВАННЯ

Частина 3

Навчальний посібник до виконання
курсової роботи

Харків «ХАІ» 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

ОСНОВИ ПРОГРАМУВАННЯ

Частина 3

Навчальний посібник до виконання
курсової роботи

Харків «ХАІ» 2018

УДК 004.4
О-72

Колектив авторів:
М. Г. Мокляк, Є. В. Соколова, П. О. Лучшев, Т. Г. Дегтярьова,
О. В. Лучшева, В. А. Постернакова

Рецензенти: канд. техн. наук, доц. М. О. Волк,
канд. техн. наук, доц. О. С. Ляшенко

Основи програмування [Електронний ресурс] : навч. посіб. до
О-72 виконання курсової роботи. В 3 ч. Ч. 3 / М. Г. Мокляк [та ін.]. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2018. – 50 с.

Викладено короткі теоретичні відомості про теорію ґрафів, виконано підбір завдань, призначених для набуття навичок з відпрацювання основних задач з введення, зберігання, оброблення і візуалізації ґрафів за допомогою сучасних засобів програмування й розвитку навичок з об'єктно-орієнтованого програмування.

Для студентів спеціальності «Інженерія програмного забезпечення» при виконанні розрахунково-графічних і курсових робіт з дисципліни «Основи програмування», а також може бути корисним для підготовки до олімпіад з програмування.

Іл. 20. Табл. 1. Бібліогр.: 5 назв

УДК 004.4

«Харківський авіаційний інститут», 2018

УМОВНІ ПОЗНАЧЕННЯ І СКОРОЧЕННЯ

$G := (V, E)$	–	ґраф, загальне позначення
$\bar{G} := (V, \bar{E})$	–	неорієнтований ґраф
$\vec{G} := (V, \vec{E})$	–	орієнтований ґраф
$\{елемент: умова\}$	–	множина елементів, які задовольняють умову
$V(G)$	–	множина вершин ґрафа G
$E(G)$	–	множина зв'язків ґрафа G
$\|G\|, \|V(G)\|$	–	порядок ґрафа (кількість вершин)
$\|\|G\|\|, \|E(G)\|$	–	розмір ґрафа (кількість зв'язків)
$N(v)$ / $N[v]$	–	окіл/замкнутий окіл вершини ґрафа
$deg(v)$	–	степінь вершини
$d(v, u)$	–	відстань між вершинами
$R(G)$	–	радіус ґрафа
$D(G)$	–	діаметр ґрафа
$\min\{...\}$ / $\max\{...\}$	–	мінімальне/максимальне значення із множини

ПЕРЕДМОВА

Написання курсової роботи є завершальним етапом вивчення дисципліни «Основи програмування». Курсова робота – це самостійно виконана закінчена робота в області програмування, що містить сукупність результатів і положень, висунутих студентом для публічного захисту, що має внутрішню єдність, яка свідчить про можливість автора вирішувати конкретні прикладні завдання у процесі розроблення програмних засобів. Навчальний посібник призначено для студентів спеціальності 121 «Інженерія програмного забезпечення».

Студент отримує індивідуальне завдання (варіант). Самостійно вивчає методи і математичні моделі для опису досліджуваного процесу або об'єкта, способів і структур подання даних в ЕОМ за допомогою науково-технічної та спеціальної літератури предметної області (теорії ґрафів). Після вивчення теоретичних аспектів вирішення завдання необхідно розробити зовнішні специфікації (форми) для розв'язуваної задачі, побудувати алгоритми розв’язання і реалізувати їх за допомогою мови програмування. Потім виконується тестування та налагодження розробленого додатка.

Пояснювальну записку до курсової роботи виконують на окремих аркушах форматом А4 обсягом 30 – 40 сторінок і супроводжують необхідними формулами, графіками, рисунками, таблицями, блок-схемами алгоритмів [1].

1. МЕТА І ЗАДАЧІ КУРСОВОЇ РОБОТИ

Курсове проектування має такі цілі:

– закріпити, поглибити і систематизувати теоретичні знання й практичні вміння, отримані при вивченні дисципліни «Основи програмування»;

– прищепити навички самостійного розроблення програм на мовах високого рівня;

– закріпити навички розроблення зручного інтерфейсу користувача, системи меню, навігації, системи допомоги;

– навчити студентів методам автоматизованого вирішення складних науково-технічних завдань;

– розвинути здібності самостійної роботи;

– освоїти сучасні методи і технології програмування;

– набути практичних навичок в оформленні результатів проектування відповідно до стандартів.

Для виконання курсової роботи студенту необхідно:

– навчитися працювати з науково-технічною й спеціальною літературою;

– засвоїти методи і математичні моделі для опису досліджуваного процесу або об'єкта;

– вивчити способи і структури подання даних в ЕОМ;

– розробити зовнішні специфікації для розв'язуваної задачі з урахуванням принципів структурного програмування;

– побудувати на основі вибраних методів і математичних моделей алгоритми вирішення;

– реалізувати алгоритми за допомогою мови програмування;

– виконати налагодження і тестування програмного забезпечення;

– застосувати нормативно-довідкову літературу для подання результатів роботи;

– виробити аналітичний підхід до оцінювання результатів у вигляді заключних висновків і рекомендацій.

2. ПОРЯДОК ВИКОНАННЯ РОБОТИ

2.1. Система оцінювання і терміни здачі КП/РГР

Для виконання курсового проектування (КП) або розрахунково-графічної роботи (РГР) кожного студента закріплюють за викладачем, призначеним завідувачем кафедри для керівництва проектуванням. Керівник курсової роботи зобов'язаний у тижневий термін видати їм індивідуальні завдання, які складаються з урахуванням успішності студента, обсягу часу, відведеного на виконання робіт, а також за

результатами індивідуальної попередньої співбесіди. У період курсового проектування студенту необхідно відвідувати консультації, де обговорюються важкі випадки вирішення завдань, а також оцінюється дотримання плану-ґрафіка виконання курсової роботи. У кінці семестру в індивідуальному порядку проводиться захист курсових робіт. До захисту студент подає пояснювальну записку з курсового проектування, оформлену відповідно до вимог, і програмне забезпечення (здійснюваний файл, вихідні файли, файли з даними для тестів) на електронному носії.

Під час виконання курсової роботи студенти значну частину від запланованого обсягу робіт виконують самостійно. Для підвищення ефективності проектування необхідно, щоб кожен студент працював за індивідуальним планом, складеним на підставі типового переліку, що містить такі етапи:

– підбір та аналіз літературних джерел за темою курсового проектування;

– математичну постановку завдань, визначених за темою курсової роботи;

– огляд існуючих математичних методів для вирішення ідентичних завдань, вибір і обґрунтування раціонального методу;

– побудову структурної і функціональної схем програмної системи з виділенням основних функціональних груп;

– складання зовнішніх специфікацій для всіх елементів структурної схеми;

– розроблення алгоритмів роботи функціонально повних частин розв'язуваної задачі;

– вибір алгоритмічної мови програмування і складання програми для розроблених алгоритмів;

– підготовку програм на електронних носіях інформації та налагодження окремих програмних модулів;

– створення методики комплексного налагодження програмного забезпечення;

– проведення контрольних розрахунків і аналіз отриманих результатів;

– оформлення технічної документації на програмну систему і пояснювальної записки до роботи;

– підготовку доповіді до захисту курсової роботи та її публічний захист.

На підставі індивідуального плану курсової роботи формується ґрафік, згідно з яким визначаються терміни і обсяги робіт для вибраних часових інтервалів. Хід виконання курсового проектування перевіряється на контрольних точках викладачем і, можливо, представниками кафедри і деканату. Відставання або зрив ґрафіка розцінюється як невиконання

студентом навчального плану. Як типовий план робіт можна рекомендувати ґрафік, поданий у таблиці 2.1.

Таблиця 2.1 – Ґрафік типового плану робіт

№ п/п		Задачі		Бали				Терміни (тижні)
				КП		РГР		
1	РПЗ	Постановка задачі		62	3	60	4	1
2		Теорія за умовою задачі			4		6	2
3		Структурна схема декомпозиції задачі			4		3	3
4		Інтерфейс користувача	вхідні дані/форми		3		3	4
5			вихідні дані/форми		3		3	4
6			формат вхідного файла		3		2	4
7		Аномалії			5		5	5
8		Метод вирішення			5		5	6
9		Функціональні тести			5		5	7-12
10		Низхідне проектування			25		25	7-10
11		Логічна модель програми			2		2	13-14
12	ПО	Уведення даних з файла		28	4	40	5	7
13		Уведення даних з клавіатури			3		4	8
14		Реакція на аномалії			4		5	9-12
15		Відображення матриці суміжності			3		3	10
16		Ґрафічна візуалізація ґрафа			3		4	10
17		Обчислення характеристики			6		10	11
18		Порівняння ґрафів			1		1	11-14
19		Оригінальність (ООП, ґрафіка тощо)			4		8	7-14
20	Захист	Презентація		10	5	0	0	13-14
21		Доклад			5		0	14-15
		Всього		**100**	**100**	**100**	**100**	

Якщо завдання не будуть виконані у зазначені терміни, то за кожний прострочений тиждень буде відніматися 1 бал. Таким чином, мінімальна оцінка для кожної задачі становить 1 бал.

Бали за пп. 12 – 21 нараховуються тільки після виконання пп.1 – 8.

2.2. Структура пояснювальної записки до КП/РГР

Зовнішня специфікація містить опис зовнішніх функцій проекту і очікуваної поведінки продукту, що розробляється з точки зору зовнішнього стосовно нього спостерігача-користувача.

Зовнішня специфікація містить документацію опису лише зовнішніх аспектів програмного виробу (що являє собою вибір) і не пов’язана з його внутрішньою структурою (як програмний вибір організовано). Таким чином структура пояснювальної записки до роботи має такий вигляд:

1. Зовнішня специфікація
    1.1. Типове завдання

1.2. Теорія (в межах типового завдання)
1.3. Постановка завдань курсової роботи, вхідна/вихідна інформація
2. Проектування (внутрішня специфікація)
2.1. Вхідні дані, обмеження (деталізація п. 1.3)
2.2. Вихідні дані
2.3. Формат вхідного файла
2.4. Низхідне детальне проектування
2.4.1. Завдання/підзадача «…» (вхід-вихід)
2.4.2. Завдання/підзадача «…» (вхід-вихід)
2.4.3. Повна схема декомпозиції
2.5. Аномалії
3. Розроблення програмного забезпечення
3.1. Інтерфейс користувача (пп. 2.1 – 2.3)
3.1.1. Вхідні форми
3.1.2. Вихідні форми
3.2. Клас «…» (пп. 2.4 – …)
3.2.1. Метод «…», алгоритм
3.2.2. Метод «…», алгоритм
3.3. Логічна модель/діаграма класів (п. 2.4.3)
3.4. Функціональні тести (п. 2.5)
4. Результати тестування
4.1. Тестування аномалій
4.2. Тестування еквівалентних ґрафів
4.3. Тестування нееквівалентних ґрафів
Висновки

3. ВАРІАНТИ РОЗРАХУНКОВИХ ЗАВДАНЬ

3.1. Типове завдання

Як розрахункове завдання студент отримує завдання порівняння двох ґрафів з числом вершин $n \le 20$ і числом зв'язків $m \le 50$, тобто визначення еквівалентності або нееквівалентності ґрафів після попередньої обробки на основі заданої характеристики. У результаті необхідно розробити і налагодити програму, оформлену у вигляді виконуваного файла (* .exe), для вирішення завдання порівняння двох ґрафів. Кожному студенту видається варіант характеристики ґрафа, спосіб попередньої обробки, типова форма введення графу.

У типовому завданні (другий аркуш звіту) вказується номер варіанта і деталізація завдання:

1. Тип ґрафа:
   - орієнтований;
   - простий орієнтований;
   - неорієнтований.

2. Попереднє оброблення порівнюваних ґрафів.
3. Характеристика, за якою порівнюються два ґрафа.
4. Спосіб зберігання ґрафа в пам'яті ЕОМ (матриця суміжностей, інцидентностей тощо. Структури даних).
5. Формат зберігання ґрафа у файлі (FO, FI, MFO, MFI, EL).
6. Обмеження на порядок і розмір графа.

3.2. Варіанти характеристик ґрафа для розрахунку

1. Радіус ґрафа.
2. Діаметр ґрафа.
3. Двозв’язність ґрафа (відсутність шарнірів).
4. Дводольність (біхроматичність) ґрафа.
5. Кількість вершин у ґрафі для кожного з можливих ексцентриситетів.
6. Кількість периферійних вершин ґрафа.
7. Середня відстань ґрафа.
8. Цикломатичне число (кількість компонент зі слабкою зв'язністю).
9. Обхват ґрафа.
10. Кількість рівнів порядкової функції орґрафа.
11. Кількість висячих вершин ґрафа і сума попарно найкоротших відстаней між ними.
12. Найбільша кількість зв'язків, видалення яких залишає ґраф зв'язаним.
13. Хроматичне число ґрафа.
14. Хроматичний клас ґрафа (розфарбування ребер).
15. Кількість компонент сильної зв'язності.
16. Топологічний індекс Вінера ґрафа.
17. Число внутрішньої стійкості ґрафа.
18. Число зовнішньої стійкості ґрафа.
19. Кількість мостів у ґрафі.
20. Кількість центральних вершин ґрафа.
21. Топологічний індекс Харарі ґрафа.
22. Максимальна щільність серед околів кожної трійки вершин.
23. Максимальна щільність серед замкнутих околів кожної пари вершин.
24. Середній коефіцієнт кластеризації ґрафа.
25. Топологічний індекс зв'язності (Рандича) остова ґрафа.
26. Кількість остовів ґрафа.
27. Максимальна щільність серед підґрафів, які одержують почерговим видаленням пар вершин.
28. Максимальний топологічний індекс зв'язності (Рандича) серед підґрафів, одержуваних почерговим видаленням трійок вершин.

29. Максимальна щільність серед замкнутих околів кожної трійки вершин.
30. Максимальна щільність серед околів кожної пари вершин.

4. ТЕОРЕТИЧНА ІНФОРМАЦІЯ ПРО ҐРАФИ

Однією з проблем теорії ґрафів є відсутність постійної термінології. Найчастіше в сучасних публікаціях під одними і тими ж термінами розуміють різні речі. Нижче наведено визначення, які будуть використані при вивченні теорії ґрафів у межах курсу «Основи програмування», вони базуються на теорії множин з курсу «Дискретна математика».

4.1. Базові поняття теорії ґрафів

Взагалі під терміном **ґраф** (*graph*) розуміють зв'язану пару множин:

$$G := (V, E), \tag{4.1}$$

де V – множина із n **вершин** (*vertices*), які іноді називають **вузлами** (*nodes)* або **точками** (*points*):

$$V := V(G) = \{v_1, v_2, v_3, \dots, v_n\}; \tag{4.2}$$

$E \subseteq V \times V$ – множина із m **зв'язків** (*edges*):

$$E := E(G) = \{e_1, e_2, e_3, \dots, e_m\}. \tag{4.3}$$

При цьому потужність множини вершин визначає **порядок ґрафа** (*graph order*):

$$\left| G \right| = \left| V \right| = n, \tag{4.4}$$

а потужність множини зв'язків – **розмір ґрафа** (*graph size*):

$$\|G\| = \left| E \right| = m. \tag{4.5}$$

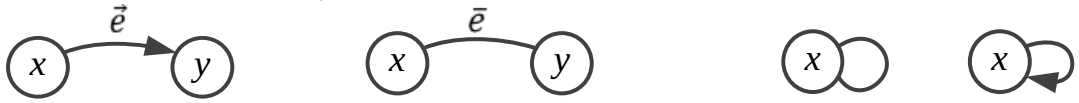
Якщо $\|G\| = 0$, то при $|G| < 1$ ґраф називається **нульовим** (*null graph*), при $|G| = 1$ – **тривіальним** (*trivial graph*), а при $|G| > 1$ – **порожнім** (*empty graph*).

Поняття зв'язку e є двоелементною підмножиною (*2-subset*) **суміжних** (*adjacent*) вершин ґрафа:

$$\begin{gathered} e := \{x, y\} = xy; \quad x, y \in V; \\ E \subseteq \{\, \{x, y\} : \forall\, x, y \in V \,\}. \end{gathered} \tag{4.6}$$

Кожний зв'язок $e = xy$ $(x \neq y)$ **інцидентний** (*incidence*) **граничним вершинам** (*endpoints*) x та y (див. рисунок 4.1). І взагалі, суміжність –

відношення між двома подібними елементами (вершинами або зв'язками), а інцидентність є відношення між двома різнорідними елементами (вершинами і зв'язками).



а б в

Рисунок 4.1 – Види зв’язку між вершинами ґрафа:
а – дуга; б – ребро; в – петля

Якщо у зв'язку зазначено напрямок $\vec{e} := (e, x, y)$ від вершини x до y, то такий зв'язок називається **дугою** (*arc, directed edge,* рисунок 4.1, а) і їх множина буде позначатися $\vec{E}$:

$$\vec{E} := \{\, (e, x, y) : e = xy \neq yx;\ \ e \in E;\ \ x, y \in V \,\}. \tag{4.7}$$

У цьому випадку сам ґраф буде називатися **орієнтованим**, або **орґрафом** (*directed graph, digraph*):

$$\vec{G} := \left(V, \vec{E}\right). \tag{4.8}$$

Зв'язок, у якого граничні вершини збігаються $e = xx$, називається **петлею** (*loop,* рисунок 4.1, в). Треба звернути увагу на те, що в орієнтованому ґрафі петля може мати тільки один напрямок, тому позначається такою трійкою (e, x, x). Приклад орієнтованого ґрафа наведено на рисунку 4.2.

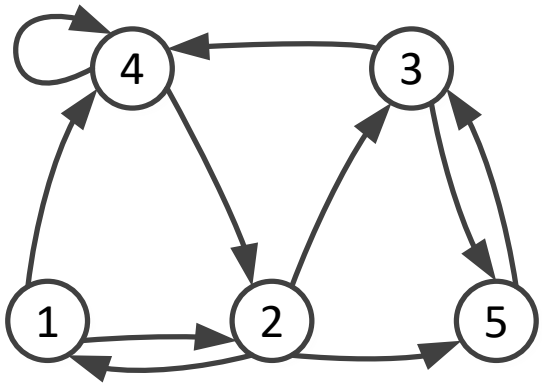


Рисунок 4.2 – Орієнтований ґраф і його множина вершин і зв’язків

Текстовий опис вказаного ґрафа має такий вигляд:

$$\vec{G} := \left(V, \vec{E}\right);$$
$$V = \{1, 2, 3, 4, 5\};$$
$$\vec{E} = \{\, \{1,2\}, \{2,1\}, \{2,3\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,1\}, \{4,2\}, \{4,4\}, \{5,3\} \,\}.$$

Якщо **орієнтований ґраф** (*directed graph*) не має петель і містить не більше одного зв'язку між будь-якою парою вершин, то він буде **простим орґрафом** (*oriented graph*):

$$\vec{G} := (V, \vec{E});$$
$$\forall\, x, y \in V : (x \neq y) \wedge \left(xy \in \vec{E}\right) \wedge \left(yx \notin \vec{E}\right). \tag{4.9}$$

Таким чином, на рисунку 4.2 Рисунок зображено орґраф, а на рисунку 4.3 – простий орґраф.

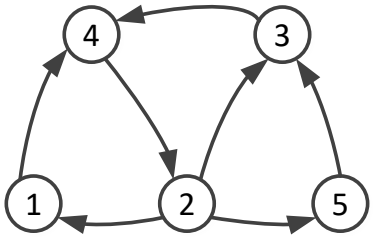


Рисунок 4.3 – Простий орієнтований ґраф

Текстовий опис вказаного ґрафа має такий вигляд:

$$\vec{G} := \left(V, \vec{E}\right); V = \{1, 2, 3, 4, 5\};$$
$$\vec{E} = \{\, \{1,4\}, \{2,1\}, \{2,3\}, \{2,5\}, \{3,4\}, \{4,2\}, \{5,3\} \,\}.$$

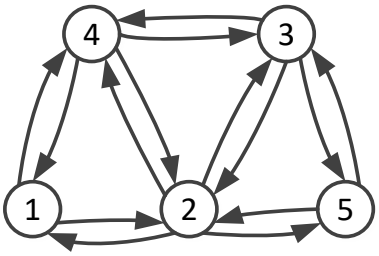
Якщо в ґрафі всі зв'язки між вершинами формуються двома різноспрямованими дугами $\vec{e} = (e, x, y)$ і $\overleftarrow{e} = (e, y, x)$ (рисунок 4.4, а), то такий зв'язок називається **ребром** (*edge, undirected edge,* рисунок 4.1,б), множина яких буде позначатися $\bar{E}$:

$$\bar{E} := \{\, (e, x, y) : e = xy = yx;\ \ e \in E;\ \ x, y \in V \,\}, \tag{4.10}$$

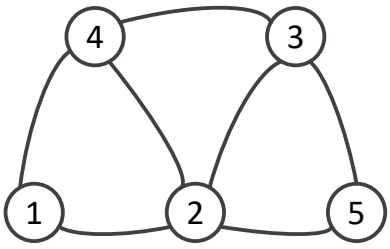
а сам граф буде **неорієнтованим,** або **неорґрафом** (*undirected graph*):

$$\bar{G} := (V, \bar{E}). \tag{4.11}$$

Для спрощення всі зв’язки зображуються у вигляді однієї лінії і без стрілок (див. рисунок 4.4, б).



а



б

Рисунок 4.4 – Еквівалентні подання неорієнтованого ґрафа:
а – $\vec{G} := \left(V, \vec{E}\right)$; б – $\bar{G} := (V, \bar{E})$

Текстовий опис вказаного ґрафа має такий вигляд:

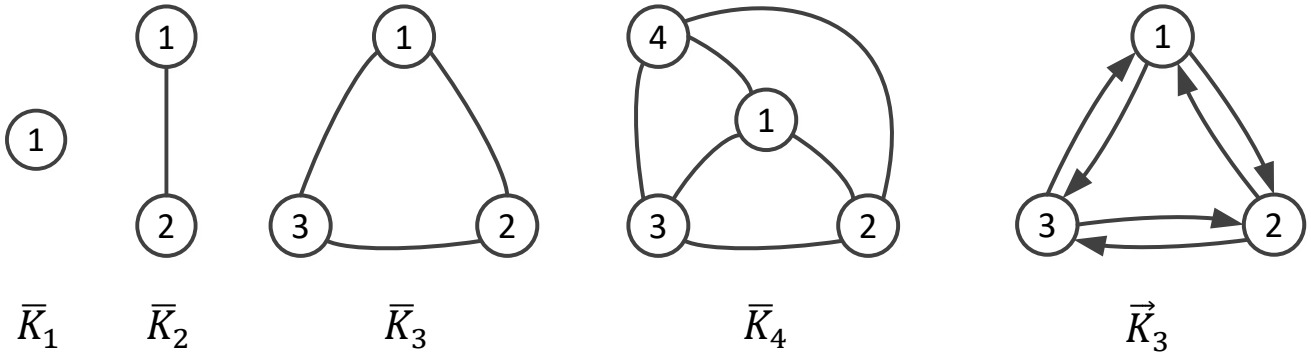
$V = \{1, 2, 3, 4, 5\}$;

$\vec{E} = \{\{1,2\},\{1,4\},\{2,1\},\{2,3\},\{2,4\},\{2,5\},\{3,2\},\{3,4\},\{3,5\},$
$\{4,1\},\{4,2\},\{4,3\},\{5,2\},\{5,3\}\}$;

$\bar{E} = \{\{1,2\},\{1,4\},\sout{\{2,1\}},\{2,3\},\{2,4\},\{2,5\},\sout{\{3,2\}},\{3,4\},\{3,5\},$
$\sout{\{4,1\}},\sout{\{4,2\}},\sout{\{4,3\}},\sout{\{5,2\}},\sout{\{5,3\}}\}$.

У випадках, коли характеристика спрямованості може бути будь-якою, для позначення ґрафа буде використовуватися вираз (4.1), а коли важливо зазначити контекстну прив'язку до орґрафів або неорґрафів, то (4.8) і (4.11) відповідно. Крім цього, у загальній теорії ґрафів кожен зв'язок може мати «вагу» (зважений ґраф) і/або може існувати кілька зв'язків між вершинами з однаковим напрямком (мультиґраф). У цьому посібнику розглядаються ґрафи, «вага» яких дорівнює 1.

Якщо в ґрафі кожна вершина пов'язана з усіма іншими, то такий ґраф називається **повним** і позначається K_N, де *N* – кількість вершин (див. рисунок 4.5).



$\bar{K}_1$ $\bar{K}_2$ $\bar{K}_3$ $\bar{K}_4$ $\vec{K}_3$

Рисунок 4.5 – Повні неорґрафи з 1,2,3 і 4 вершин і повний орґраф з 3 вершин

Кількість зв'язків у такому ґрафі обчислюють за формулою для неорґрафа і орґрафа відповідно:

$$\begin{aligned} \|\bar{K}_N\| &= \frac{N \cdot (N-1)}{2}; \\ \left\|\vec{K}_N\right\| &= N \cdot (N-1). \end{aligned} \tag{4.12}$$

Ґраф $G = (V, E)$ є **дводольним** (*bipartite*), якщо його множина вершин формується з двох підмножин таким чином:

$$\begin{aligned} &V = V_1 \cup V_2; \\ &V_1 \cap V_2 = \emptyset; \\ &E \subseteq V_1 \times V_2, \end{aligned} \tag{4.13}$$

тобто зв’язки у такому ґрафі існують тільки між вершинами з різних підмножин (рисунок 4.6).

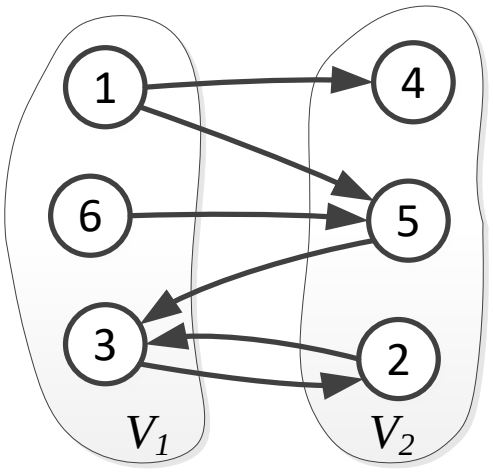


Рисунок 4.6 – Дводольний орґраф

Текстовий опис вказаного ґрафа має такий вигляд:

$$\vec{G} := \left(V, \vec{E}\right);$$
$$V = V_1 \cup V_2 = \{1,2,3,4,5,6\};$$
$$V_1 \cap V_2 = \emptyset;$$
$$\vec{E} \subseteq V_1 \times V_2 = \{\,\{1,4\},\{1,5\},\{2,3\},\{3,2\},\{5,3\},\{6,5\}\,\}.$$

Частковим графом, або **суґрафом** (*spanning subgraph*), ґрафа G називається ґраф G', який утворюється з вихідного ґрафа видаленням деякої кількості зв'язків із збереженням вершин:

$$G' := (V, E');$$
$$E' \subset E. \tag{4.14}$$

Наприклад, ґраф на рисунку 4.3 може бути суґрафом як для ґрафа на рисунку 4.2Рисунок , так і для ґрафа на рисунку 4.4Рисунок , а.

Підґрафом (*subgraph*) ґрафа G називається граф G', який утворюється шляхом видалення з вихідного ґрафа деякої кількості вершин разом з усіма ребрами, інцидентними видаленим вершинам:

$$G' := (V', E');$$
$$E' = \{\,\{x,y\} : \forall\, x,y \in V';\ V' \subset V\,\}. \tag{4.15}$$

Відповідно, **суперґраф** (*supergraph*) – це будь-який ґраф, що містить вихідний ґраф (тобто для якого вихідний ґраф є підґрафом, або суґрафом).

Підґраф, який складається тільки з вершин, які суміжні заданій вершині $v \in V,$ і зв'язків між ними називається **околом** (*neighborhood,* рисунок 4.7):

$$N(v) := \{\, u : uv \in E;\ \forall\, u \in V\,\} \tag{4.16}$$

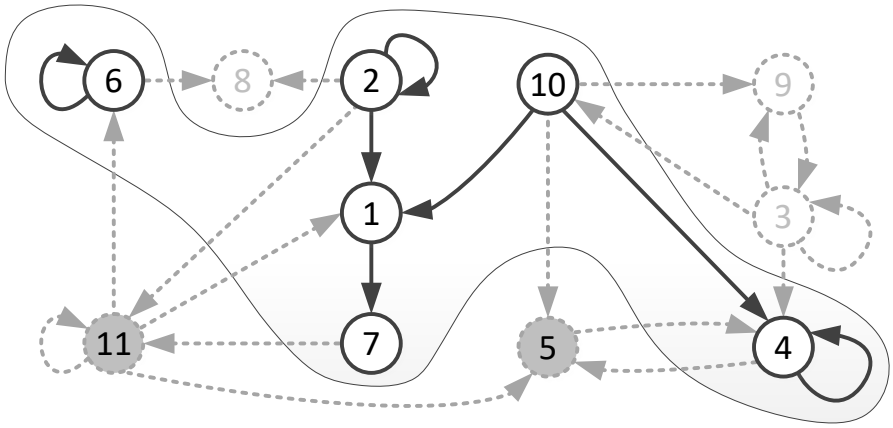


Рисунок 4.7 – Ґраф околу N пари вершин $\{ 5, 11 \}$

До **замкнутого околу** (*closed neighborhood*) входить і сама вершина v:

$$N[v] := \{ v \} \cup N(v) . \tag{4.17}$$

Крім цього, в орієнтованому ґрафі окіл може бути поданий множиною суміжних вершин $N^+(v)$, що виходять із v ребер, і множиною вершин $N^-(v)$, що входять до v:

$$\begin{aligned} N(v) &:= N^+(v) \cup N^-(v); \\ N^+(v) &:= \{ u : vu \in \vec{E} \}; \\ N^-(v) &:= \{ u : uv \in \vec{E} \}. \end{aligned} \tag{4.18}$$

У загальному випадку поняття околу (4.16) – (4.18) можуть бути використані для довільної кількості вершин графа, а під *k*-околом розуміють безліч вершин на відстані *k* від заданої v (рисунок 4.8).

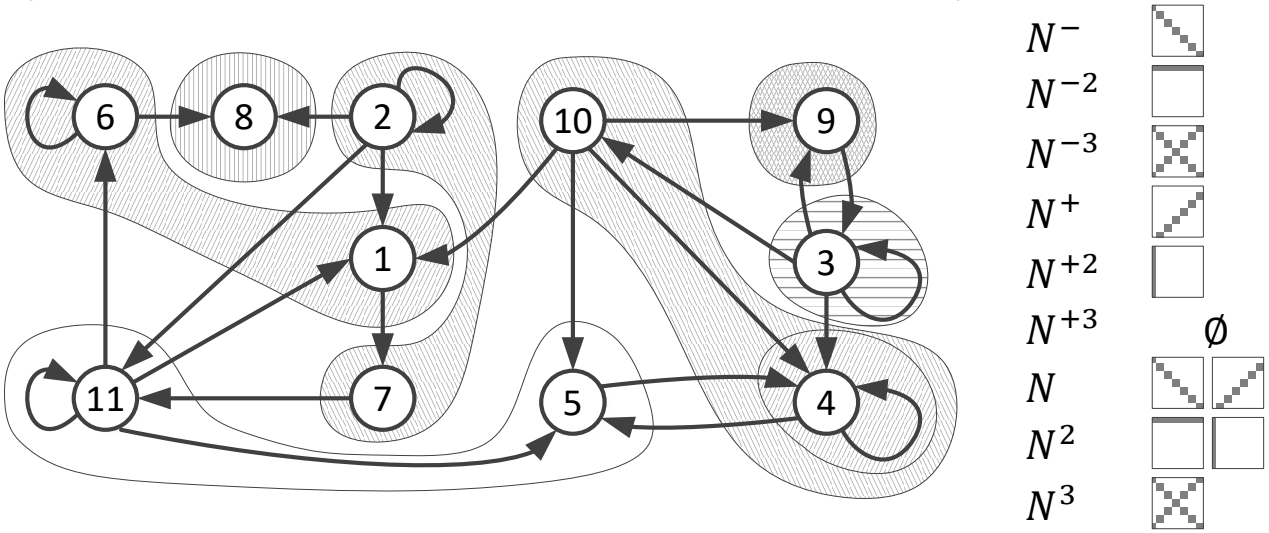


Рисунок 4.8 – Околи $N^{\pm k}$ для пари вершин 5 и 11

4.2. Базові характеристики ґрафа

Степінь вершини (*degree*), або **валентність** (*valency*), – кількість зв'язків ґрафа G, інцидентних вершині v (або суміжних з нею вершин):

$$deg(v) := |N(v)|. \tag{4.19}$$

Для орієнтованого ґрафа його можна виразити через півстепені входу і виходу:

$$deg(v) := deg^{+}(v) + deg^{-}(v). \tag{4.20}$$

Півстепінь входу (*in-degree*) – кількість дуг в орієнтованому ґрафі, що входять у вершину v з її околу:

$$deg^{+}(v) = | N^{-}(v) |. \tag{4.21}$$

Відповідно, **півстепінь виходу** (*out-degree*) – кількість дуг в орієнтованому ґрафі, що виходять з вершини v в її околі:

$$deg^{-}(v) = | N^{+}(v) |. \tag{4.22}$$

При підрахунку степеня петлю враховують двічі. Максимальний і мінімальний степені вершин ґрафа G позначаються відповідно $\Delta(G)$ і $\delta(G)$:

$$\Delta(G) := \max \{ deg(v) : \forall v \in V \}; \tag{4.23}$$

$$\delta(G) := \min \{ deg(v) : \forall v \in V \}; \tag{4.24}$$

$$\overline{deg}(G) := \frac{1}{| G |} \sum_{\forall v \in V} deg(v); \tag{4.25}$$

$$\delta(G) \leq \overline{deg}(G) \leq \Delta(G). \tag{4.26}$$

У загальному випадку ґраф є **k-регулярним** (*k-regular*) або просто регулярним, якщо всі його вершини мають однаковий степінь:

$$deg(v) = k, \forall v \in V. \tag{4.27}$$

Вершина є **ізольованою** (*isolated*), якщо її степінь дорівнює нулю, множину таких вершин визначають таким чином:

$$\{ v : deg(v) = 0, \forall v \in V \}. \tag{4.28}$$

Вершина ґрафа є **листом** (*leaf*) або **висячею (***pendant***),** якщо її степінь дорівнює одиниці, відповідно, множина таких вершин

$$\{ v : deg(v) = 1, \forall v \in V \}. \tag{4.29}$$

4.3. Маршрут, шлях і цикл у ґрафі

Маршрутом у ґрафі G називається підґраф

$$\begin{gathered} P = (V, E); \\ V = \{x_0, x_1, \dots x_{k-1}, x_k\}; \\ E = \{x_0x_1,\ x_1x_2,\ \dots,\ x_{k-1}x_k\}. \end{gathered} \tag{4.30}$$

Для стислості шлях може бути заданий перерахуванням вершин у порядку їх обходу $P = x_0, x_1, \dots x_{k-1}, x_k$. Шлях є **простим** (*trail*), якщо в ньому жодна дуга не зустрічається двічі, або **складним** (в іншому випадку). Шлях є **елементарним** (*path*), якщо жодна вершина не зустрічається в ньому двічі, або **неелементарним** (в іншому випадку). Шлях, для якого початкова вершина збігається з кінцевою, називається **контуром** (*circuit*). Якщо у контурі немає збіжних вершин, окрім першої й останньої, то це є **цикл** (*cycle*). Шлях або цикл, у якого всі зв'язки зустрічаються лише один раз, називається **ейлеровим шляхом**, або **ейлеровим циклом**. Взаємозв'язок цих понять наведено на рисунку 4.9.

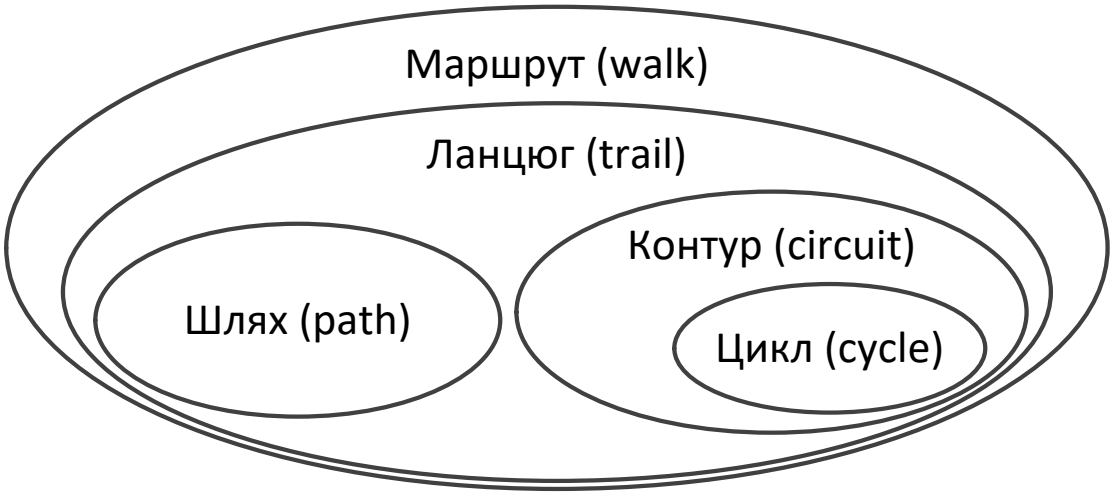


Рисунок 4.9 – Ієрархія понять «маршрут», «ланцюг», «шлях» «цикл» в теорії ґрафів

Елементарний шлях, в якому кількість дуг на одиницю менше кількості вершин ґрафа, називається **гамільтоновим шляхом** (*Hamiltonian path*). Іншими словами, такий шлях проходить через усі вершини один раз. Аналогічно визначається **гамильтонів цикл** (*Hamiltonian cycle*) – це цикл, що проходить лише один раз через всі вершини ґрафа, виключаючи довільно вибраний початок.

Відстань (*distance*) між двома вершинами – число ребер у **найкоротшому шляху** (*geodesic path*):

$$d(v,u) \coloneqq min\ \{\ \|P\| : V(P) \cap \{v,u\} = \{x_0, x_k\}\ \}. \quad (4.31)$$

Може існувати кілька найкоротших шляхів між двома вершинами, а в орієнтованих ґрафах відстань $d(v,u)$ може не збігатися з $d(u,v)$ або зворотний шлях може зовсім бути відсутнім. Якщо немає шляху між двома вершинами, то відстань між ними вважають нескінченною.

Ексцентриситет (*eccentricity*) вершини – відстань від вершини v до максимально віддаленої від неї вершини:

$$\varepsilon(v) := max\ \{\ d(v,u) :\ \forall u\ \in\ V(G)\ \}. \quad (4.32)$$

Радіусом (*radius*) ґрафа називається мінімальний ексцентриситет серед усіх вершин ґрафа:

$$R(G) := min\ \{\ \varepsilon(v) :\ \forall v\ \in\ V(G)\ \}. \quad (4.33)$$

Діаметром (*diameter*) ґрафа називається максимальний ексцентриситет серед усіх вершин ґрафа. Таким чином, діаметр – це найбільша відстань між усіма парами вершин ґрафа:

$$D(G) := max\ \{\ \varepsilon(v) :\ \forall v\ \in\ V(G)\}. \quad (4.34)$$

Центральною вершиною (*center vertex*) ґрафа називається вершина, ексцентриситет якої дорівнює радіусу цього ґрафа:

$$\varepsilon(v) = R(G). \quad (4.35)$$

Периферійною вершиною (*peripheral vertex*) ґрафа називається вершина, ексцентриситет якої дорівнює діаметру цього ґрафа:

$$\varepsilon(v) = D(G). \quad (4.36)$$

Вершина v є **псевдопериферійною** (*pseudo-peripheral*), якщо існує вершина u і виконується умова $d(u,v) = \varepsilon(v) = \varepsilon(u)$.

Вершинна зв'язність (*vertex connectivity*) ґрафа – це найменша кількість вершин, видалення яких приводить до незв'язного або тривіального ґрафа, тобто ґрафа із одних вершин.

Реберна (дугова) зв'язність (*edge connectivity*) визначається як найменша кількість ребер (дуг), видалення яких приведе до незв'язного або тривіального ґрафа.

Дві вершини **сильно зв'язані** (*strong connected*), якщо існує шлях з першої в другу і з другої в першу. Підґраф з таких вершин утворює **сильно зв'язану компоненту** (*strong connected component*), яких в ґрафі може бути декілька (див. рисунок 4.1).

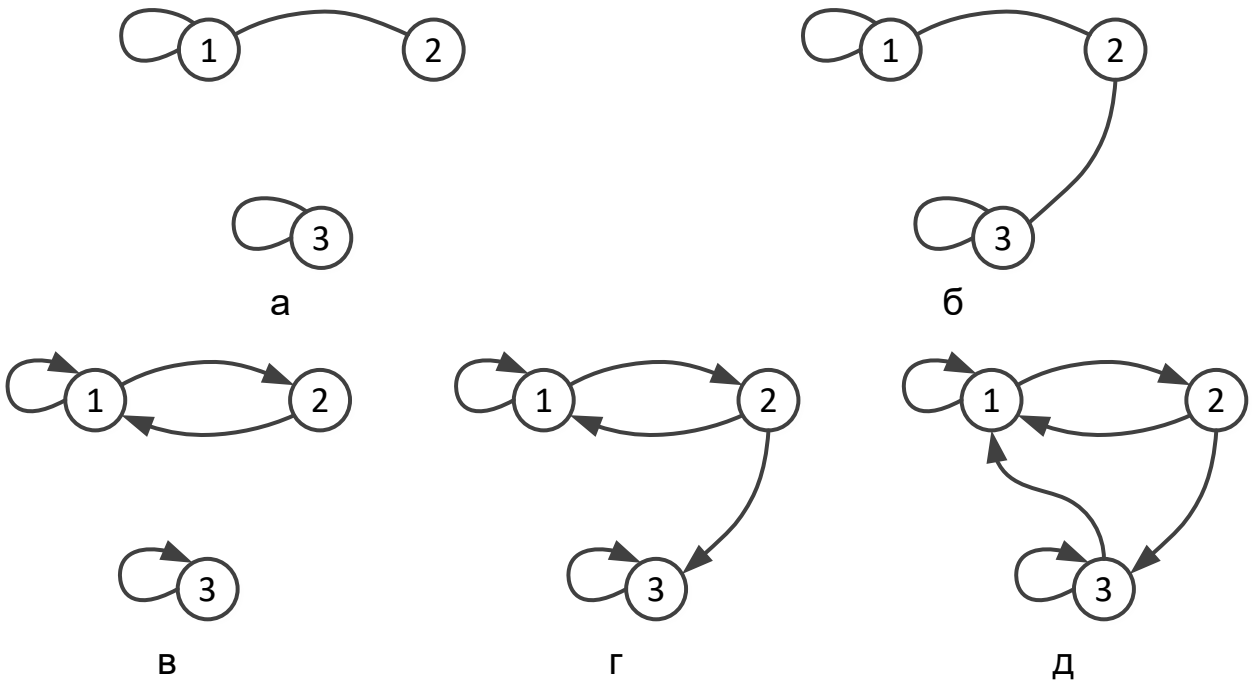


Рисунок 4.10 – Рівень зв’язності в ґрафах: а, в – незв’язані; б – зв’язаний; г – слабко зв’язаний; д – сильно зв’язаний

Формування ґрафа G^* із сильно зв’язаних компонент ґрафа G називається **конденсацією** (*condensation,* рисунок 4.11):

$$SCC(G) := (V^*, E^*) = G^*;$$
$$V^* := \{ C_i : C_i \subset V,\ C_i \cap C_j = \emptyset \};$$
$$E^* := \{ \{C_i, C_j\} : i \neq j, \{u, v\} \in E, u \in C_i, v \in C_j \}. \quad (4.37)$$

Таким чином, кількість компонент сильної зв’язності ґрафа G визначається кількістю вершин ґрафа G^*:

$$p_0(G) = |G^*|. \quad (4.38)$$

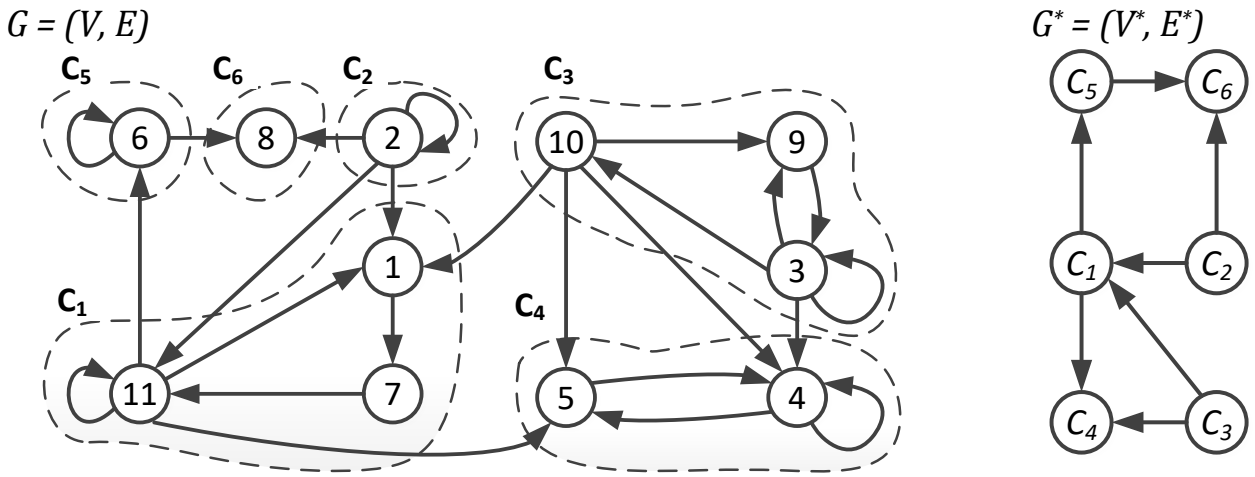


Рисунок 4.11 – Сильно зв’язані компоненти C_i ґрафа G та результат їх конденсації в ґраф G^*

Цикломатичне число (*cyclomatic number*) p_1 ґрафа без урахування петель, який складається з p_0 компонент зв'язності, обчислюється таким чином:

$$p_1(G) = p_0(G) - \|G\| + |G|. \tag{4.39}$$

Воно визначає найменшу кількість ребер, які необхідно видалити з даного ґрафа, щоб отримати ґраф без циклів, тому цикломатичне число відомо також як **контурний ранг** (circuit/cycle rank).

4.4. Додаткові властивості й метрики ґрафа

Ґраф називають *k*-хроматичним, якщо його вершини можна розфарбувати *k* різними кольорами так, що ніякі суміжні вершини не будуть пофарбовані однаково.

Найменше число $\gamma(G)$, для якого ґраф є *k*-хроматичним, називається хроматичним числом неорієнтованого ґрафа і позначається через $\gamma(G)$.

Аналогічно визначають і хроматичний клас ґрафа, тільки замість вершин виконують розфарбування зв'язків.

Вершини, пофарбовані однаково, утворюють **внутрішньо стійку множину**, і хроматичне число можна визначити як мінімальне число внутрішньо стійких множин, які покривають у сукупності всі вершини ґрафа.

Двочастковий (біхроматичний) ґраф – це ґраф, що не містить циклів (контурів) непарної довжини. У такого ґрафа множину вершин можна розбити на дві частини таким чином, що кінці будь-якого зв'язку виявляться в різних частинах.

Індекс Вінера (*Wiener*) – сума довжин найкоротших шляхів d (u, v), що існують в ґрафі між його вершинами:

$$W(G) \coloneqq \sum_{u,v \in V} d(u,v). \tag{4.40}$$

Середню відстань (*average distance*), яка дозволяє оцінити компактність, визначають як відношення суми довжин найкоротших шляхів до максимальної кількості зв'зків, які можливі в даному ґрафі:

$$\mu(G) \coloneqq \frac{W(G)}{|G|\cdot(|G|-1)}. \tag{4.41}$$

Індекс зв'язності (*connectivity index*), запропонований Рандичем (*Randić*), має вигляд

$$r(G) \coloneqq \sum_{\forall(v,u)\in E(G)} \frac{1}{\sqrt{\deg(v)\deg(u)}}. \tag{4.42}$$

Ґраф називається **зв’язним** (*connected*), якщо для кожної пари його вершин знайдеться шлях, який їх зв'язує. В іншому випадку ґраф називають незв'язним. Усякий незв'язний ґраф складається з двох або більше компонент зв'язності.

Обчислення **індексу Харарі** (*Harari index*):

$$H(G) \coloneqq \sum_{\substack{\forall\, v,u \in V(G) \\ v \neq u}} \frac{1}{d(v,u)}. \tag{4.43}$$

Щільним (*dense*) ґрафом називається ґраф, в якому кількість ребер близька до максимальної ($|E| \cong |V|^2$). Ґраф з малою кількістю ребер ($|E| \ll |V|^2$) називається **розрідженим** (*sparse*). Різниця між розрідженим і щільним ґрафом досить невелика і залежить від контексту, а для обчислення щільності ґрафа використовують відношення кількості зв’язків до максимально можливої кількості зв’зків:

$$\rho(G) \coloneqq \frac{\|G\|}{|G| \cdot (|G| - 1)} = \frac{|E(G)|}{|V(G)| \cdot (|V(G)| - 1)}. \tag{4.44}$$

Локальний коефіцієнт кластеризації (*Local Clustering Coefficient*, *LCC*) показує, скільки найближчих сусідів даного вузла є також найближчими сусідами один для одного і визначається відношенням кількості існуючих зв'язків в околі вершини до максимально можливої кількості зв'язків в цьому околі:

$$C(v) \coloneqq \frac{\|N(v)\|}{|N(v)| \cdot (|N(v)| - 1)}. \tag{4.45}$$

Середній коэфіцієнт кластеризації (*average clustering coefficient*) ґрафа

$$\bar{C}(G) \coloneqq \frac{1}{|G|} \sum_{v \in V(G)} C(v). \tag{4.46}$$

Посередництво вершини (*betweeness centrality*) – величина, яка відображає важливість вузла як посередника на шляху між всіма парами вершин у ґрафі:

$$C_B(v) \coloneqq \sum_{\substack{\forall s,v,t \in V(G) \\ s \neq v \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{4.47}$$

де σ_{st} – кількість найкоротших шляхів з вершини s в t, а $\sigma_{st}(v)$ – кількість тих з них, які проходять через v. Для нормалізації значення індексу необхідно $C_B(v)$ розділити на $(|G|-1)\cdot(|G|-2)$, тоді при максимальному рівні посередництва значення дорівнює одиниці, а при мінімальному – нулю.

4.5. Гомеоморфне стиснення ґрафа

Операція **гомеоморфного стиснення** (*homeomorphic/series reduction*) полягає у заміні послідовності з двух зв'язків $\{u,x\}$ и $\{x,v\}$ на один $\{u,v\}$ з видаленням вершини другого степеня (півстепені виходу і входу, що дорівнюють одиниці).

На рисунку 4.12 наведено**Error! Reference source not found.** приклад, де вершини 1 і 5 є вершинами другого степеня, але гомеоморфне стиснення допускається тільки для вершини 1 и тільки для орієнтованого ґрафа.

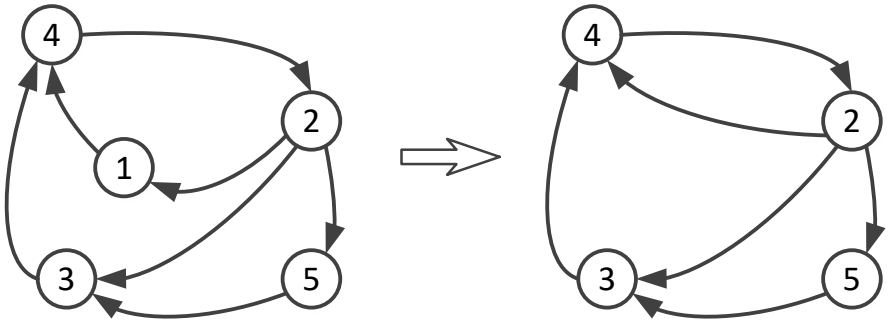


Рисунок 4.12 – Приклад гомеоморфного стиснення для орієнтованого ґрафа

Для простого ґрафа стиснення для вершини 1 порушує умову (4.9), а видалення вершини 5 допускається тільки для мультиґрафів.

5. ПРАКТИЧНІ РЕКОМЕНДАЦІЇ

5.1. Зовнішні формати подання ґрафів

5.1.1. Формат вихідних зв'язків (*Format Output*)

Подання ґрафа одновимірним масивом *FO* формується у такий спосіб: спочатку записуються номери вершин (у будь-якому порядку), в які заходять дуги з першої вершини, потім ставиться роздільник 0, далі – номери вершин, в які заходять дуги з другої вершини, після чого ставиться роздільник 0 і т. д. Таким чином, для неорґрафа і мультиґрафа будується масив *FO* довжиною 2(q+N), оскільки кожне ребро враховується двічі. Для орґрафа і псевдоґрафа масив *FO* має довжину q+N.

Приклад *FO*-подання для ґрафа на рисунку 4.2:

FO: 2 4 0 1 3 5 0 4 5 0 2 4 0 3 0

5.1.2. Формат вхідних зв’язків (*Format Input*)

Масив *FI* формується так: спочатку записуються номери вершин (у будь-якому порядку), з яких виходять дуги в першу вершину, потім ставиться роздільник 0, далі записуються номери вершин, з яких виходять дуги в другу вершину, і ставиться роздільник 0 і т. д. Таким чином, для неорґрафа будується масив *FI* довжиною 2(q+N), оскільки кожне ребро враховується двічі. Для орґрафа і псевдоґрафа масив *FI* має довжину q+N. Зауважимо, що для неорґрафів і мультиґрафів *FO* і *FI* подання збігаються.

Приклад *FI*- подання для ґрафа на рисунку 4.2:

FI: **2 0 1 4 0 2 5 0 1 3 4 0 2 3 0**

5.1.3. Модифікований формат вихідних зв’язків (*Modify Format Output*)

У MFO-поданні ґрафа замість роздільників 0 використовується додатковий масив P довжиною N, в якому зазначаються верхні межі закінчення списків номерів вершин у масиві G, суміжних із заданою вершиною по вихідних дугах. Для неорґрафа довжина масиву G дорівнює 2q, а масиву P – N. Для орґрафа і псевдоґрафа довжина масиву G дорівнює q, а масиву P – N.

Приклад *MFO*-подання для ґрафа на рисунку 4.1:

G: **2 4 1 3 5 4 5 2 4 3**

P: **2 5 7 9 10**

5.1.4. Модифікований формат вхідних зв’язків (*Modify Format Input*)

На відміну від MFO-подання ґрафа в MFI-поданні в масиві P зазначаються верхні межі закінчення списків вершин, суміжних із заданою вершиною по вхідних в неї дугах.

Приклад MFO-подання для ґрафа на рисунку 4.2:

G: **2 1 4 2 5 1 3 4 2 3**

P: **1 3 5 8 10**

5.1.5. Список зв’язків (Edge List)

Для подання ґрафа у вигляді списку зв'язків необхідно використовувати кілька різних роздільників, які дозволять зобразити кожний зв'язок у вигляді логічно зв'язаної пари чисел.

Наприклад, EL-подання ґрафа з рисунку 4.2 з використанням пробілу для відокремлення номерів вершин і крапки з комою для розділення самих зв'язків має такий вигляд:

1 4; **1 2**; **2 1**; **2 3**; **2 5**; **3 4**; **3 5**; **4 4**; **4 2**; **5 1**;

а з використанням пари фігурних дужок –

{ {1 4} {1 2} {2 1} {2 3} {2 5} {3 4} {3 5} {4 4} {4 2} {5 1} }.

5.2. Способи подання ґрафів у пам’яті ЕОМ

Для подання ґрафа $G = (V, E)$ з $V = \{v_1, \dots v_n\}$ і $E = \{e_1, \dots e_m\}$) в оперативній пам’яті ЕОМ можна використовувати матриці суміжності або інцидентності, кожна з яких має свої достоїнства й недоліки.

5.2.1. Матриця суміжності (adjacency)

Одним з найбільш популярних подань ґрафа є матриця **суміжності** (*adjacency*). Це квадратна матриця $A(G) = (a_{i,j})_{n \times n}$, розмір якої визначається кількістю вершин у ґрафі, а її елементи для будь-якого типу ґрафів (орієнтованих і неорієнтованих) визначаються за суміжністю вершин, тобто наявністю зв'язку між ними:

$$a_{i,j} = \begin{cases} 1, & v_i v_j \in E; \\ 0, & v_i v_j \notin E. \end{cases} \quad (5.1)$$

Якщо ґраф не має петель, то на діагоналі матриці розташовані нулі. За визначенням (4.10) матриця суміжності неорґрафа має бути симетричною відносно головної діагоналі, а для простого орґрафа відповідно до (4.9) – навпаки: $a_{i,j} \neq a_{j,i}$ і $a_{i,i} = 0$.

Для орґрафа сума елементів рядка дорівнюватиме півстепеню виходу (4.22), а сума елементів стовпця – півстепеню входу (4.21) відповідної вершини, а їх загальна сума дорівнює степеню вершини як для орґрафа, так і неорґрафа. Сума всіх елементів матриці дорівнює кількості зв’язків в орґрафі й подвоєній їх кількості в неорґрафі.

5.2.2. Матриця інцидентності (incedncy)

Матриця **інцидентності** (*incedncy*) – прямокутна матриця $B(G) = \left(b_{i,j}\right)_{n \times m}$, рядки якої відповідають вершинам, а стовпці – ребрам (дугам) ґрафа. Для неорґрафа елементи матриці визначають таким чином:

$$b_{i,j} = \begin{cases} 2, & v_i \in e_j, e_j = (v_i v_i); \\ 1, & v_i \in e_j, e_j \neq (v_i v_i); \\ 0, & v_i \notin e_j. \end{cases} \quad (5.2)$$

Сума елементів i-го рядка дорівнює степеню вершини v_i ґрафа, а сума елементів будь-якого стовпця – 2.

Для орієнтованого ґрафа елементи матриці інцидентності визначають дещо в інший спосіб. Оскільки кожна дуга інцидентна двом різним вершинам (за винятком того випадку, коли дуга утворює петлю), то кожен стовпець містить один елемент, який дорівнює (–1) у рядку вихідної вершини, і один, який дорівнює (+1) у рядку вихідної вершини, або всі елементи стовпця дорівнюють 0 (у випадку петлі):

$$b_{i,j} = \begin{cases} -1, & \vec{e}_j = (v_i u); \\ 0, & v_i \notin \vec{e}_j \vee \vec{e}_j = (v_i v_i); \\ +1, & \vec{e}_j = (u\, v_i). \end{cases} \quad (5.3)$$

Відповідно, модуль суми всіх від'ємних елементів рядка дорівнює півстепеню виходу, сума всіх додатних елементів – півстепеню входу, а сума елементів будь-якого стовпця дорівнює нулю.

У деяких випадках ситуація, коли петлю в орієнтованому ґрафі подано стовпцем з нульових елементів, спричиняє незручність, оскільки не дозволяє порядковим перебиранням цих елементів визначити вершину з петлею і потребує додаткової функції для визначення номера вершини за номером зв'язку. У цій ситуації діють за аналогією з неорґрафом, присвоюючи елементу значення, яке дорівнює двом.

5.3. Відображення матриць у DataGridView

Для відображення матриці суміжностей можна використати стандартний компонент *DataGridView*. Згідно зі стандартним режимом використання *DataGridView* кожному елементу таблиці присвоюється відповідне значення матриці суміжності, що призводить до дублювання інформації. Уникнути цього можна, якщо використовувати віртуальний режим роботи даного компонента. У цьому випадку застосовують подієву модель роботи. Таблиця *DataGridView* за допомогою обробника події запитує у джерела даних тільки ті комірки, які у певний момент відображаються на екрані:

```
int[,] G; // Матриця суміжності
int nVertecies = 4;
DataGridView dgv;

private void MainForm_Load(object sender, EventArgs e)
{
      G = new int[nVertecies, nVertecies];  dgv.VirtualMode = true;
      dgv.CellValueNeeded += ValueNeeded;
      dgv.RowCount = dgv.ColumnCount = nVertecies;
}

private void ValueNeeded(object sender, DataGridViewCellValueEventArgs e)
{
      e.Value = G[e.RowIndex, e.ColumnIndex];
}
```

5.4. Візуалізація ґрафів

Для візуалізації ґрафів можна використовувати як власні напрацювання, так і існуючі інструментальні засоби. Одним з таких інструментів є пакет ***Microsoft Automatic Graph Layout (MS AGL)*** для платформи ***.NET***, який дозволяє сформувати розміщення вершин і зв'язків

ґрафа і їх зображення в автоматичному режимі. Пакет складається з трьох модулів:

1. ***Microsoft.Msagl.dll*** – модуль формування і компонування взаєморозташування вершин і зв'язків ґрафа.
2. ***Microsoft.Msagl.Drawing.dll*** – модуль з визначенням класів вершин, зв'язків і самого ґрафа. Крім цього, в модулі визначено різні атрибути рисунка такі, як кольори, стилі ліній тощо.
3. ***Microsoft.Msagl.GraphViewerGdi.dll*** – модуль, який містить елемент керування для інтерактивної взаємодії з користувачем і відображення ґрафа.

За допомогою відкритого пакета ***Microsoft Automatic Graph Layout (MS AGL)*** можна достатньо просто додати в проект на мові ***C#*** візуалізацію графа [4]:

1) створити проект ***Windows Forms Application*** (наприклад, ***GraphDemo***);
2) усередині каталогу з проектом створити окремий підкаталог для файлів бібліотеки ***MS AGL***;
3) додати в проект посилання на три бібліотеки, зазначені вище (рисунок 5.1);

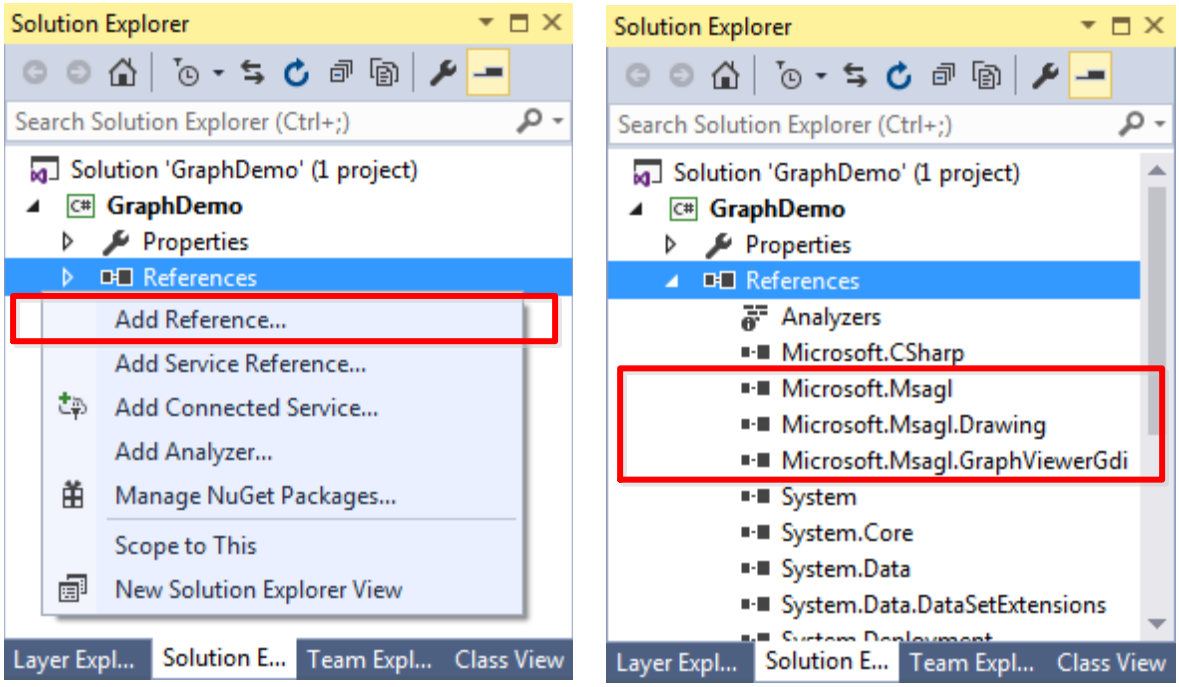


Рисунок 5.1 – Додавання файлів бібліотеки ***MS AGL***

4) у панелі інструментів (за допомогою контекстного меню) додати елемент керування ***GViewer***, який знаходиться в бібліотеці ***Microsoft.Msagl.GraphViewerGdi.dll*** (рисунок 5.2);

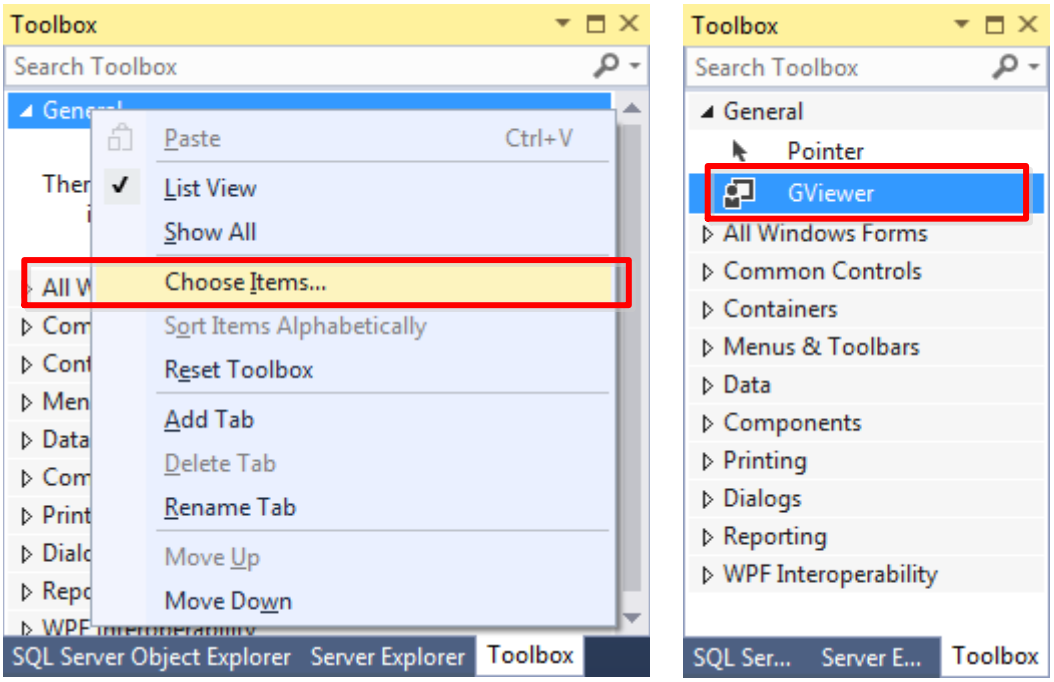


Рисунок 5.2 – Додавання елемента керування ***GViewer***

5) розташувати элемент керування ***GViewer*** на формі (рисунок 5.3);

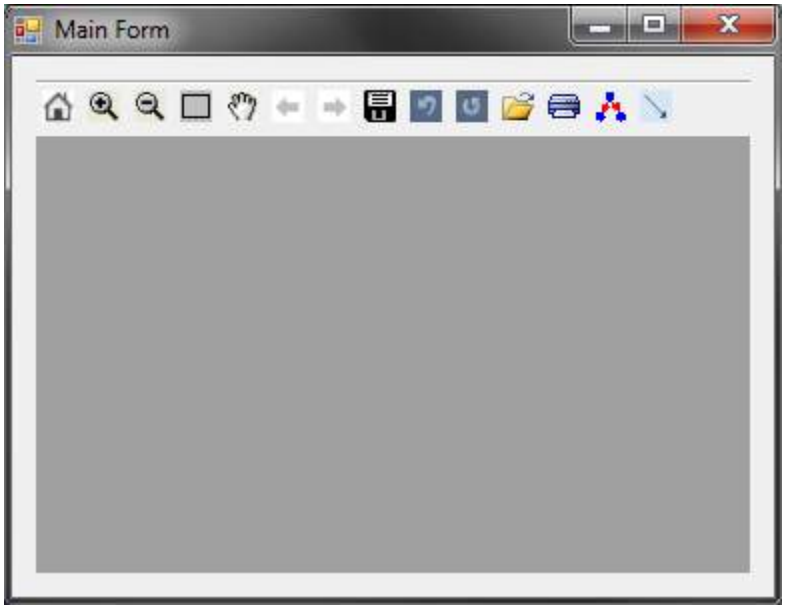


Рисунок 5.3 – Вигляд елемента керування ***GViewer*** на формі

6) додати в текст програми посилання на бібліотеку

```
using Microsoft.Msagl.Drawing;
```

і код для створення об'єкта, що являє ґраф, в який за допомогою методу `AddEdge` додаються ребра:

```
private void ShowGraphClick(object sender, EventArgs e)
{
  Graph graph = new Graph();
  graph.AddEdge("A", "B");
```

```
  graph.AddEdge("A", "C");
  graph.AddEdge("A", "A");
  gViewer1.Graph = graph; ///!!! Присвоєння обов’язково
}
```

Кожна вершина має ім'я, яке задається рядковим значенням (у даному прикладі «A», «B», «C»). Для відображення ґрафа на екрані обов'язково необхідно виконати операцію присвоення ґрафа відповідному полю компонента ***GViewer*** (рисунок 5.4):

```
gViewer1.Graph = graph
```

Якщо цього не зробити, то зображення ґрафа НЕ буде виведено на екран.

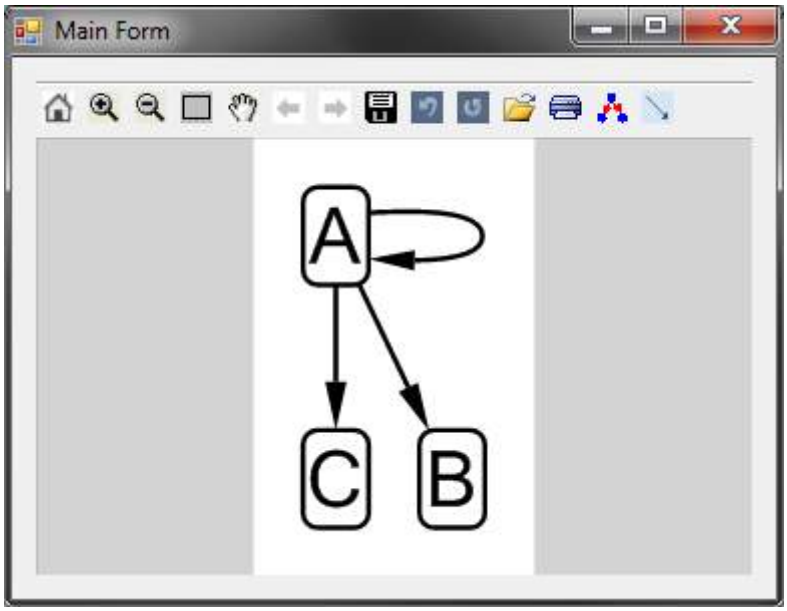


Рисунок 5.4 – Відображення ґрафа на формі

ДОДАТОК А
Приклад виконання роботи

Типове завдання

Варіант № __

1	Тип ґрафа.....	*Орієнтований*
2	Попереднє оброблення порівнюваних ґрафів	*Видалення ізольованих вершин*
3	Характеристика, за якою порівнюються два ґрафи.....	*Коефіцієнт посередництва (betweeness centrality) для кожної вершини*
4	Спосіб зберігання ґрафа у пам’яті ЕОМ	*Матриця інцидентності*
5	Формат зберігання ґрафа у файлі	*TGF (Trivial Graph Format)*
6	Обмеження на порядок і розмір ґрафа.....	*Кількість вершин – не більше 20, кількість зв’язків – не більше 50*

1. Постановка задачи

Використовуючи технологію низхідного проектування розробити і налагодити програму для порівняння двох орієнтованих ґрафів на еквівалентність за значенням коефіцієнта посередництва для кожної вершини. Перед обчисленням характеристики кожного ґрафа виконати попереднє оброблення, видаливши в ньому всі ізольовані вершини.

Для кожного ґрафа зробити можливим введення даних з використанням клавіатури і текстового файла. Дані мають бути подані у форматі ***TGF*** (*Trivial Graph Format*). У процесі введення інформації врахувати, що кількість вершин $n \le 20$ і кількість зв'язків $m \le 50$ для кожного ґрафа.

Розроблена програма має бути налагоджена і протестована на різних комбінаціях коректних і некоректних вхідних даних, які виявляються у процесі розроблення програми, а також аналізу можливих аномалій, пов'язаних з форматом вхідних даних. Виконати попереднє оброблення і обчислити характеристику ґрафа.

2. Теорія за умовою задачі. Визначення індексу посередництва вершини ґрафа

Індекс посередництва (*betweenness centrality*) характеризує, наскільки важливу роль даний вузол відіграє на шляху між іншими вузлами, і показує, скільки найкоротших шляхів між усіма вузлами мережі проходить через певний вузол. Індекс посередництва – це міра контролю. Якщо у будь-якого вузла високий показник індексу посередництва, можна припустити, що він – єдиний зв'язок між різними частинами мережі.

Наприклад, при аналізі футбольного матчу індекс посередництва дозволяє зробити висновок про те, наскільки робота з м'ячем між двома гравцями залежить від третього. Гравці з високим рівнем індексу посередництва відіграють ключові ролі в підтримці темпу гри.

У соціальних мережах високо цінується можливість швидко поширювати інформацію. Тому одним з ключових питань контролю над поширюваною інформацією в соціальній мережі є визначення важливості того чи іншого вузла мережі як передавальної ланки. Це роблять за ступенем впливу вузла на передачу інформації – індексом посередництва (*betweenness centrality*), який оцінює учасника мережі саме в контексті його ступеня контролю за передачею інформації і можливістю контролювати зв'язки між іншими її учасниками.

Коли мова йде про контроль за інформаційними потоками в соціальній мережі і ступенем впливу на інших, то для цього учасник повинен бути посередником між іншими вузлами, оскільки це дає йому можливість перервати контакт між ними. Саме індекс посередництва є найбільш відповідною мірою для визначення ступеня здатності індивіда контролювати взаємодію людей у своєму соціальному оточенні.

Взаємодія двох несуміжних індивідів може знаходитися під контролем можливих посередників. При пошуках роботи, наприклад, важливо не те, скільки знайомих у претендента, а скільки знайомих у цих знайомих. Метод оцінювання індексу посередництва для вершини, запропонований Л. Фріманом у 1977 р. [5], полягає у знаходженні частки найкоротших шляхів, що з'єднують усі пари вершин, які проходять через дану вершину. Це сума ймовірностей того, що інші учасники у своїх взаємодіях будуть вдаватися до посередництва даного індивіда.

Показник нормалізується до інтервалу [0, 1] розподілом на максимально можливу величину – $(n-1) \cdot (n-2)$. Індекс посередництва є глобальною характеристикою вершини і має більш цікаву інтерпретацію, ніж інші індекси посередництва. Показник визначено на зв'язкових ґрафах, хоча індекси для експансивності й престижу невідомі. Показник враховує лише найкоротші шляхи від вершини до вершини і оснований на припущенні, що за наявності між двома вершинами кількох коротких шляхів рівної довжини кожен з них використовується з однаковою ймовірністю. За індексом посередництва оцінюють розподіл інформації в

усіх ланцюгах ґрафа, зважуючи на те, що ланцюг є величиною, зворотною їх довжині.

Для обчислення індексу посередництва даного вузла беруться всі можливі пари інших вузлів і для кожної пари обчислюється частка найкоротших шляхів між цими вузлами, які проходили б через цей вузол. Сума цих часток і становить індекс посередництва для даного вузла:

$$C_B(v) \coloneqq \frac{1}{(|G|-1)\cdot(|G|-2)} \cdot \sum_{\substack{\forall s,v,t \in V(G) \\ s \neq v \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

де σ_{st} – кількість найкоротших шляхів з вершини s до t;

$\sigma_{st}(v)$ – кількість шляхів, що проходять через v.

3. Опис вхідних даних. Приклад візуалізації ґрафа

Trivial Graph Format (*TGF*) являє собою простий текстовий формат для опису ґрафів [6]. Він складається з двох списків, які розділені символом «#». У першому списку зіставляють ідентифікатори *id* вузлів з вершинами, а потім після роздільника «#» розташовують другий список з описом ребер. У другому списку ребра визначаються парою ідентифікаторів вершин і необов'язковою позначкою ребра. Ідентифікатори вузлів – це додатні цілі числа, а позначки для вузлів і для ребер – прості рядки:

```
id₁␣[позначка 1-ї вершини]↵
id₂␣[позначка 2-ї вершини]↵
...
idₙ␣[позначка n-ї вершини]↵
#
id₁,ᵢ␣id₁,ⱼ␣[позначка 1-го ребра/дуги з вершини i у j]↵
id₂,ₚ␣id₂,ᵩ␣[позначка 2-го ребра/дуги з вершини p у q]↵
...
idₘ,ₛ␣idₘ,ₜ␣[позначка m-го ребра/дуги з вершини s у t]↵
<eof>
```

де n – кількість вершин ґрафа;

m – кількість зв'язків у ґрафі;

$1 \leq i,j,p,q,s,t \leq n$;

␣ – роздільник інформації у рядку даних;

– роздільник між списками вершин і зв'язків;

↵ – ознака кінця рядка;

<eof> – ознака кінця файла.

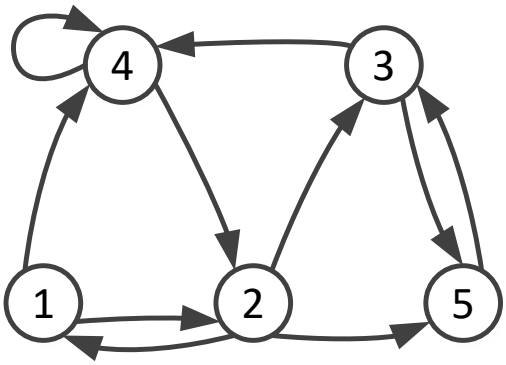
У загальному випадку ґраф, який задано у форматі TGF, може бути інтерпретований як орієнтований (рисунок А.1) або неорієнтований ґраф. Для орієнтованих ґрафів для позначення напряму у ребрі першу вершину у списку ребер будемо вважати вхідною, а другу – вихідною. З

урахуванням специфіки наданого у типовому завданні варіанта необов'язкові позначки вершин і ребер будуть ігноруватися.

$\vec{G} := (V, \vec{E})$

$V = \{1, 2, 3, 4, 5\};$

$\vec{E} = \{ \{1,2\}, \{2,1\}, \{2,3\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,1\}, \{4,2\}, \{4,4\}, \{5,3\} \};$



TGF:
```
1
2
3
4
5
#
1 2
1 4
2 1
2 5
2 3
3 4
3 5
4 4
4 2
5 3
```

Рисунок А.1 – Орієнтований ґраф і його подання у TGF-форматі

Приклади екранних форм уведення ґрафів у форматі *TGF* та їх візуалізація наведено на рисунках А.2 – А.4.

Логічну структуру програми визначення індексу посередництва вершини ґрафа наведено на рисунку А.5.

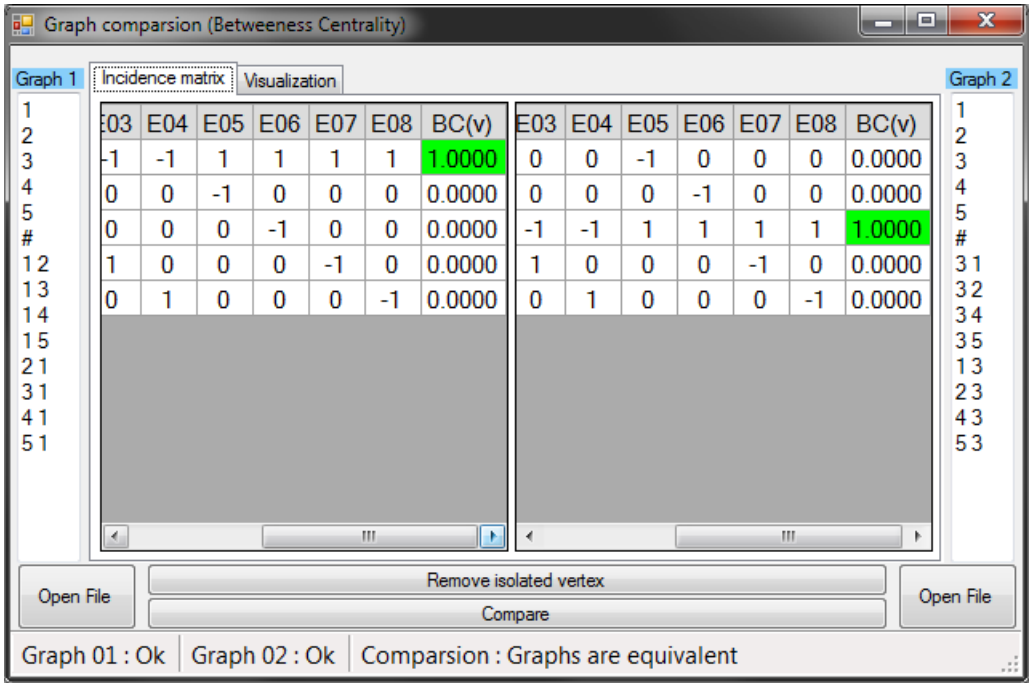


Рисунок А.2 – Результат перетворення вхідних даних у форматі *TGF* на матрицю інцидентностей

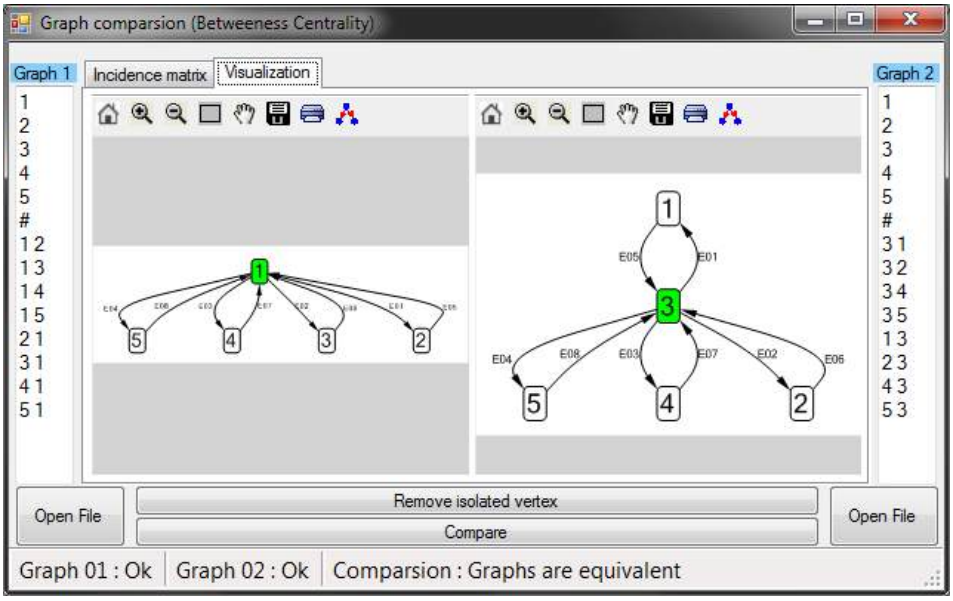


Рисунок А.3 – Візуальне подання двох еквівалентних ґрафів, які задано у форматі *TGF*

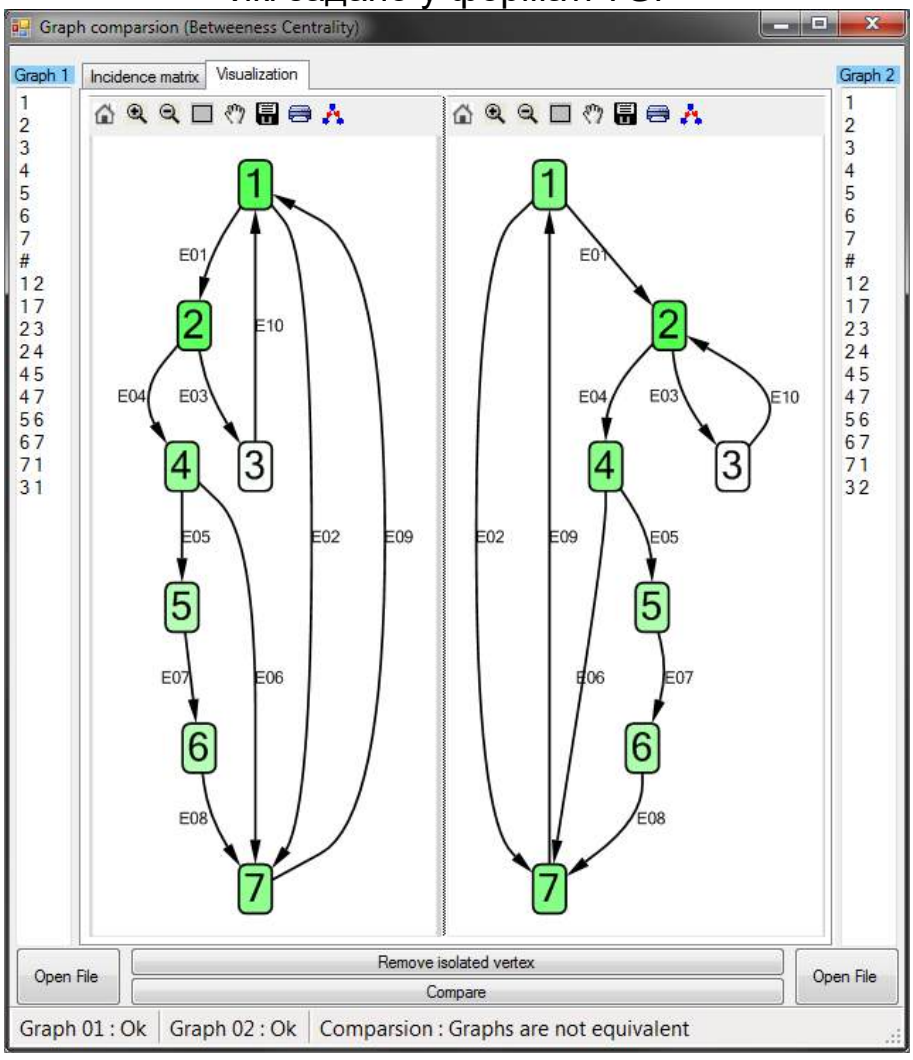


Рисунок А.4 – Візуальне подання двох нееквівалентних ґрафів, які задано у форматі TGF

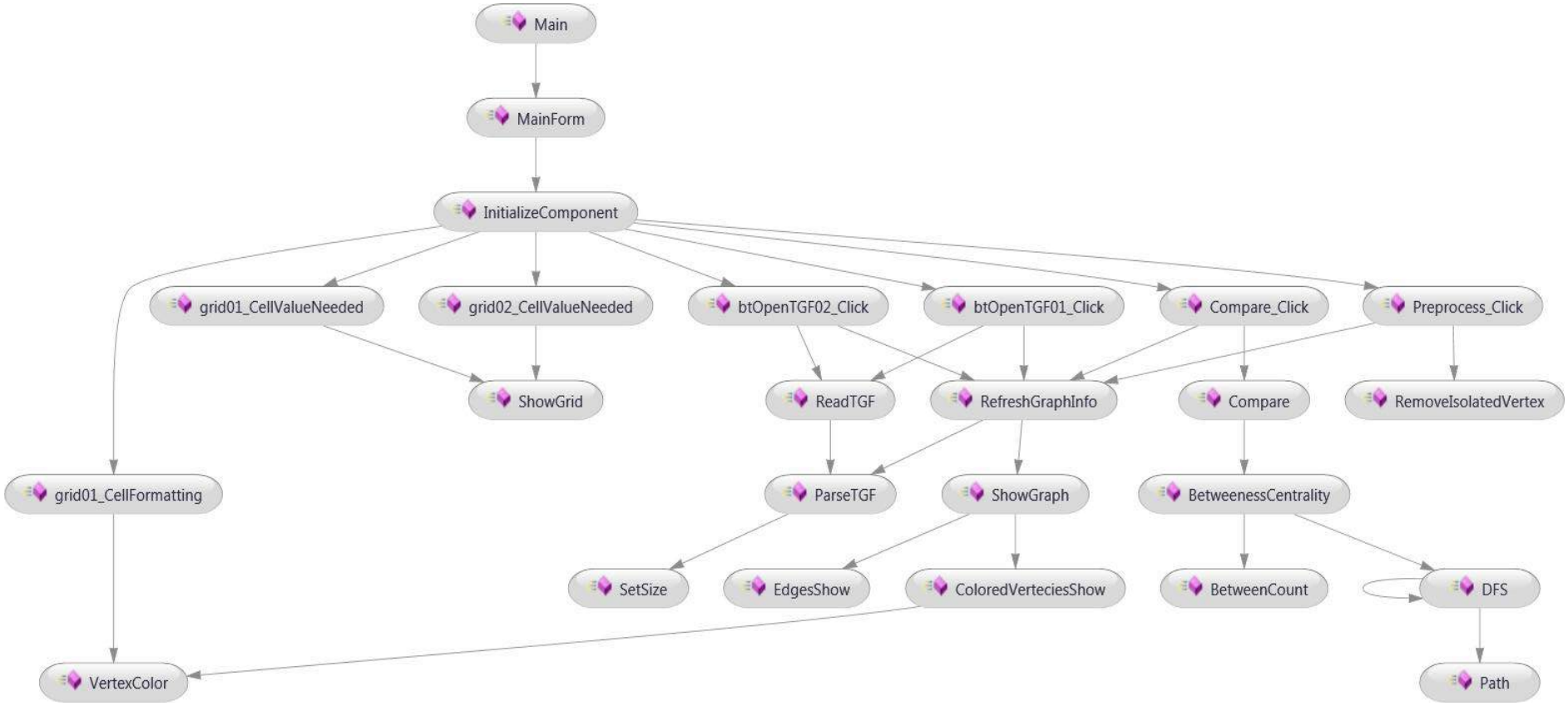


Рисунок А.5 – Логічна модель програми порівняння ґрафів за індексом посередництва

4. Лістинг програми

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;

namespace GraphDemo
{
    /// <summary>
    /// Клас для подання шляху у ґрафі
    /// як списка номерів вершин
    /// </summary>
    public class Path : List<int>
    {
        public Path() : base() { }
        public Path(Path path) : base(path) { }
    };

    /// <summary>
    /// Клас для подання внутрішньої структури ґрафа,
    /// методів його оброблення і визначення характеристик.
    /// </summary>
    public class Graph
    {
        private readonly char[] CRLF = { '\r', '\n' };
        private readonly string
            msgException01 = "Error ('{0}' is not connected)";
        private readonly string
            msgException02 = "Graph size too big";
        private readonly string
            msgException03 = "Graph order too big";
        private readonly string
            msgException04 = "Edge definition incorrect";
        private readonly string
            msgException05 = "Incorrect reference to the vertex";
        private readonly string
            msgException06 = "Multigraph found";

        int[,] a = new int[0, 0];
        int[,] mi = new int[0, 0];

        bool[] visited = new bool[0];
        Path path = new Path();
        List<Path> paths = new List<Path>();

        public string Name { get; set; }

        public void SetSize(int vCount, int eCount)
        {
            Array.Resize( ref visited, vCount +1);
```

```
        mi = new int[vCount + 1, eCount + 1];
    }

    public int VertexCount
    {
        get
        {
            return visited.Count() - 1;
        }
        set
        {
            a = new int[value + 1, value + 1];
            Array.Resize( ref visited, value +1);
        }
    }

    /// <summary>
    /// Модифікований пошук у глибину.
    /// У процесі пошуку формується маршрут, який
    /// зберігається у списку шляхів після досягнення
    /// кінцевої вершини
    /// </summary>
    /// <param name="v">номер початкової вершини</param>
    /// <param name="x">номер кінцевої вершини</param>
    public void DFS(int v, int x)
    {
        // перевіряємо, чи досягли кінцевої вершини
        if (v == x)
        {
            path.Add(v);
            paths.Add(new Path( path ));
            path.Remove(v);
            /// виходимо з процедури, оскільки шлях знайдено
            /// і далі переглядати вершину не потребується
        }
        else
        {
            /// позначаємо поточну вершину
            /// як відвідану і таку, що
            /// належить до с-ї компоненти зв'язності
            visited[v] = true;
            path.Add(v);

            /// перебираємо усі вершини
            for (int i = 1; i <= VertexCount; i++)
                ///суміжні з v, які ще не були відвідані
                if ((a[v, i] == 1) && (!visited[i]))
                {
                    /// повторюємо рекурсивно алгоритм
                    /// для i-ї вершини
                    DFS(i, x);
                }
```

```
            /// відміняємо позначку про відвідування вершини,
            /// щоб можна було до неї повернутися
            /// іншим шляхом
            visited[v] = false;
            path.Remove(v);
        }
    }

    /// <summary>
    /// Визначення характеристики (індексу посередництва)
    /// для зазначеної вершини
    /// </summary>
    /// <param name="v">номер вершини</param>
    /// <returns>
    /// Значення характеристики у діапазоні від 0 до 1
    /// </returns>
    double BetweenessCentrality(int v)
    {
        double result = 0;
        for (int s = 1; s <= VertexCount; s++)
        {
            if (s == v) continue;
            for (int t = 1; t <= VertexCount; t++)
            {
                if ((t == s) || (t == v)) continue;

                paths.Clear();
                DFS(s, t);
                if (paths.Count == 0)
                    throw new Exception(
                        string.Format(msgException01, Name)
                        );
                int bc = BetweenCount(v);
                result += (double)bc / paths.Count;
            }
        }

        result /= ((VertexCount - 1)*(VertexCount - 2));

        return result;
    }

    /// <summary>
    /// Визначення кількості шляхів, які мають
    /// зазначену вершину ґрафа. Вона не має бути
    /// початковою або кінцевою
    /// </summary>
    /// <param name="v">номер вершини</param>
    /// <returns>
    /// кількість шляхів із зазначеною вершиною
    /// </returns>
```

```
private int BetweenCount(int v)
{
    int result = 0;
    for (int i = 0; i < paths.Count; i++)
        if (paths[i].Contains(v))
            result += 1;

    return result;
}

/// <summary>
/// Перетворення ґрафа, який задано у форматі
/// Trivial Graph Format, на внутрішнє матричне
/// подання.
/// </summary>
/// <param name="str">
/// рядок з TGF-поданням ґрафа
/// </param>
public void ParseTGF(string str)
{
    string[] temp = str.Split('#');
    string[] nodes = temp[0].Split(CRLF,
        StringSplitOptions.RemoveEmptyEntries );
    string[] edges = temp[1].Split(CRLF,
        StringSplitOptions.RemoveEmptyEntries );

    if (nodes.Count() > 20)
        throw new Exception(msgException02);
    if (edges.Count() > 50)
        throw new Exception(msgException03);

    VertexCount = nodes.Count();
    SetSize(nodes.Count(), edges.Count());

    for (int i = 0; i < edges.Count(); i++)
    {
        string[] vv = edges[i].Split(' ');
        if (vv.Count() != 2)
            throw new Exception(msgException04);
        if (!nodes.Contains(vv[0]) ||
            !nodes.Contains(vv[1]))
            throw new Exception(msgException05);
        int from = int.Parse(vv[0]);
        int to   = int.Parse(vv[1]);

        if (a[from, to] != 0)
            throw new Exception(msgException06);

        a[from, to] = 1;
        if (from == to)
            mi[from, i + 1] = 2;
        else
```

```
            {
                mi[from, i + 1] = -1;
                mi[to, i + 1] = 1;
            }

        }

    }

    /// <summary>
    /// Подання об’єкта ґрафа у вигляді
    /// матриці інцидентності
    /// </summary>
    /// <param name="nVertex">номер вершини</param>
    /// <param name="nEdge">номер зв’язку </param>
    /// <returns>
    /// -1 - вихідна вершина;
    /// +1 - вхідна вершина;
    /// +2 - петля;
    ///  0 - у всіх інших випадках
    /// </returns>
    public int this[int nVertex, int nEdge]
    {
        get { return mi[nVertex, nEdge]; }
    }

    /// <summary>
    /// Визначення характеристики посередництва
    /// для заданої вершини
    /// </summary>
    /// <param name="nVertex">номер заданої вершини</param>
    /// <returns>
    /// Значення характеристики у діапазоні від 0 до 1
    /// </returns>
    public double this[int nVertex]
    {
        get { return BetweenessCentrality(nVertex); }
    }

    /// <summary>
    /// Читання даних з файла, який містить
    /// опис ґрафа у TGF-форматі
    /// </summary>
    /// <param name="FileName">ім’я файла</param>
    /// <returns>рядок з TGF-поданням ґрафа</returns>
    public string ReadTGF(string FileName)
    {
        string data = File.ReadAllText(FileName);
        ParseTGF(data);
        return data;
    }
```

```
        /// <summary>
        /// Порівняння двох ґрафів на еквівалентність
        /// за індексом посередництва для кожної вершини
        /// </summary>
        /// <param name="g">
        /// ґраф, з яким виконується порівняння
        /// </param>
        /// <returns>
        /// true  - ґрафи еквівалентні
        /// false - ґрафи нееквівалентні
        /// </returns>
        public bool Compare(Graph g)
        {
            List<double> b2 = new List<double>();
            for (int i = 1; i <= VertexCount; i++)
                b2.Add(this.BetweenessCentrality(i));

            List<double> b1 = new List<double>();
            for (int i = 1; i <= g.VertexCount; i++)
                b1.Add(g.BetweenessCentrality(i));

            b1.Sort();
            b2.Sort();

            return b1.SequenceEqual(b2);
        }

        /// <summary>
        /// кількість зв'язків у ґрафі
        /// </summary>
        public int EdgeCount
        {
            get { return mi.GetLength(1); }
        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Agl = Microsoft.Msagl.Drawing;
using Microsoft.Msagl.GraphViewerGdi;

namespace GraphDemo
{
    /// <summary>
    /// Клас для створення интерфейсу з користувачем
```

```
/// і виконання таких дій з ґрафами:
/// - введення даних з клавіатури;
/// - виведення даних з файла;
/// - відображения ґрафів у вигляді матриці інцидентності;
/// - візуалізація ґрафів у вигляді зображення;
/// - організація порівняння двох ґрафів
/// </summary>
public partial class MainForm : Form
{
    readonly private string
        Message01 = "Comparsion : Graphs are equivalent";
    readonly private string
        Message02 = "Comparsion : Graphs are not equivalent";
    readonly private string
        Message03 = "Comparsion : ";
    readonly private string
        Message04 = " : Ok ";
    readonly private string
        Message05 = "Comparsion : Unknown";

    readonly private string
        vertexFormat = "V {0:D2}";
    readonly private string
        edgeFormat   = "E{0:D2}";
    readonly private string
        bcFormat     = "0.0000";
    readonly private string
        bcHeader     = "BC(v)";
    readonly private string
        bcUnknown    = "-";

    Graph g01 = new Graph() { Name = "Graph 01" };
    Graph g02 = new Graph() { Name = "Graph 02" };

    public MainForm()
    {
        InitializeComponent();
    }

    /// <summary>
    /// Метод оброблення натисканням на кнопку "Compare".
    /// Виконується перетворення ґрафів, які задано у
    /// вигляді рядків (TGF-формат) компонент TextBox.
    /// Якщо обидва ґрафи
    /// задано коректно, то виконується їх порівняння.
    /// </summary>
    private void Compare_Click(object sender, EventArgs e)
    {
        try
        {
            RefreshGraphInfo(tbGraph01.Text, g01,
                grid01, gViewer1, status1);
```

```
            RefreshGraphInfo(tbGraph02.Text, g02,
                grid02, gViewer2, status2);

            if (g01.Compare(g02))
                status3.Text = Message01;
            else
                status3.Text = Message02;
        }
        catch (Exception x)
        {
            status3.Text =  Message03 + x.Message;
        }

    }

    /// <summary>
    /// Відображення одного ґрафа у вигляді таблиці, зображення
    /// та видавання повідомлень у випадку некоректних даних
    /// у TGF-форматі
    /// </summary>
    /// <param name="tgf">
    /// вхідний опис ґрафа у TGF-форматі
    /// </param>
    /// <param name="g">ґраф</param>
    /// <param name="dgv">
    /// таблиця для відображення матриці інцидентності
    /// </param>
    /// <param name="viewer">
    /// панель для візуалізації ґрафа
    /// </param>
    /// <param name="status">
    /// статус результату перетворення з TGF-формату
    /// </param>
    private void RefreshGraphInfo(string tgf, Graph g,
        DataGridView dgv, GViewer viewer,
        ToolStripStatusLabel status)
    {
        try
        {
            dgv.RowCount = 0;
            dgv.ColumnCount = 0;
            g.ParseTGF(tgf);
            ShowGraph(g, viewer);
            dgv.RowCount = g.VertexCount + 1;
            dgv.ColumnCount = g.EdgeCount + 1;
            dgv.AutoSizeColumnsMode =
                DataGridViewAutoSizeColumnsMode.AllCells;
            dgv.Invalidate();
            status.Text = g.Name + Message04;
        }
        catch(Exception x)
        {
```

```
            status.Text = g.Name + " : " + x.Message;
        }
    }

    /// <summary>
    /// Вибір графа № 1 з файла за допомогою стандартного
    /// діалогу "OpenFileDlg" і його відображення у вигляді
    /// матриці інцидентності й зображення
    /// </summary>
    private void btOpenTGF01_Click(object sender, EventArgs e)
    {
        openFileDialog.FileName = string.Empty;
        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                status3.Text = Message05;
                tbGraph01.Text = g01.ReadTGF(
                    openFileDialog.FileName);
                RefreshGraphInfo(tbGraph01.Text, g01,
                    grid01, gViewer1, status1);
            }
            catch (Exception x)
            {
                status3.Text = x.Message;
            }
        }
    }

    /// <summary>
    /// Вибір графа № 2 з файла за допомогою стандартного
    /// діалогу "OpenFileDlg" і його відображення у вигляді
    /// матриці інцидентності й зображення
    /// </summary>
    private void btOpenTGF02_Click(object sender, EventArgs e)
    {
        openFileDialog.FileName = string.Empty;
        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                status3.Text = Message05; //string.Empty;
                tbGraph02.Text = g02.ReadTGF(
                    openFileDialog.FileName);
                RefreshGraphInfo(tbGraph02.Text, g02,
                    grid02, gViewer2, status2);
            }
            catch (Exception x)
            {
                status3.Text = x.Message;
            }
        }
```

```
}

/// <summary>
/// Формування зображення для зазначеного ґрафа
/// на заданій панелі
/// </summary>
/// <param name="g">заданий ґраф</param>
/// <param name="view">
/// панель для відображення ґрафа
/// </param>
private void ShowGraph(Graph g, GViewer view)
{
    Agl.Graph vg = new Agl.Graph();

    ColoredVerteciesShow(g, vg);
    EdgesShow(g, vg);

    view.Graph = vg;
}

/// <summary>
/// Формування зв’язків для візуалізації
/// зазначеного ґрафа
/// </summary>
/// <param name="g">заданий ґраф</param>
/// <param name="vg">
/// об’єкт візуального подання ґрафа
/// </param>
private void EdgesShow(Graph g, Agl.Graph vg)
{
    for (int i = 1; i < g.EdgeCount; i++)
    {
        int from = 0, to = 0;
        for (int j = 1; j <= g.VertexCount; j++)
        {
            if (g[j, i] == 2) { from = to = j; break; }
            if (g[j, i] == -1) { from = j; }
            if (g[j, i] == 1) { to = j; }
        }

        var edge = vg.AddEdge(
            from.ToString(),
            string.Format(" E{0:D2}", i), to.ToString());
        // зв'язок без стрілки для неорґрафів
        // edge.Attr.ArrowheadAtTarget =
        //          Agl.ArrowStyle.None;
        edge.Label.FontSize = 5;
    }
}

/// <summary>
/// Формування візуального подання
```

```
/// множини вершин ґрафа з їх розфарбуванням
/// залежно від значення характеристики
/// посередництва

/// </summary>
/// <param name="g">заданий ґраф</param>
/// <param name="vg">
/// об’єкт візуального подання ґрафа
/// </param>
private void ColoredVerteciesShow(Graph g, Agl.Graph vg)
{
    for (int j = 1; j <= g.VertexCount; j++)
    {
        var node = new Agl.Node(j.ToString());
        Color c;
        try
        {
            c = VertexColor(g[j],Color.White, Color.Lime);
        }
        catch
        {
            c = Color.White;
        };
        node.Attr.FillColor =
            new Agl.Color(c.R, c.G, c.B);
        vg.AddNode(node);
    }
}

/// <summary>
/// Інтерполяція кольору залежно від
/// значення fraction
/// </summary>
/// <param name="fraction">
/// значення у діапазоні від 0 до 1
/// </param>
/// <param name="c1">
/// колір, що відповідає значенню fraction = 0
/// </param>
/// <param name="c2">
/// колір, що відповідає значенню fraction = 1
/// </param>
/// <returns>
/// колір, що відповідає вхідному значенню fraction
/// </returns>
Color VertexColor(double fraction, Color c1, Color c2)
{
    // Color c1 for fraction = 0
    // Color c2 for fraction = 1

    int R = (int)((c2.R - c1.R) * fraction + c1.R);
    int G = (int)((c2.G - c1.G) * fraction + c1.G);
```

```
        int B = (int)((c2.B - c1.B) * fraction + c1.B);
        return Color.FromArgb(R, G, B);
    }

    /// <summary>
    /// Прив'язка відображення ґрафа № 1 у вигляді
    /// матриці інцидентності на лівій панелі
    /// </summary>
    private void grid01_CellValueNeeded(object sender,
        DataGridViewCellValueEventArgs e)
    {
        e.Value = ShowGrid(g01, e.RowIndex, e.ColumnIndex);
    }

    /// <summary>
    /// Прив'язка відображення ґрафа № 2 у вигляді
    /// матриці інцидентності на правій панелі
    /// </summary>
    private void grid02_CellValueNeeded(object sender,
        DataGridViewCellValueEventArgs e)
    {
        e.Value = ShowGrid(g02, e.RowIndex, e.ColumnIndex);
    }

    /// <summary>
    /// Формування значення для відображення в
    /// DataGridView для зазначеного ґрафа
    /// в заданих рядку і стовпцю
    /// </summary>
    /// <param name="g">оброблюваний ґраф</param>
    /// <param name="r">номер рядка</param>
    /// <param name="c">номер стовпця</param>
    /// <returns></returns>
    private object ShowGrid(Graph g, int r, int c)
    {
        object res;
        try
        {
            if (c == 0 && r == 0)
                res = g.Name;
            else
                if (c == 0)
                    res = string.Format(vertexFormat, r);
                else if (r == 0)
                    if (c >= g.EdgeCount)
                        res = bcHeader;
                    else
                        res = string.Format(edgeFormat, c);
                else
                    if (c >= g.EdgeCount)
                        res = g[r].ToString(bcFormat);
                    else
```

```
                    res = g[r, c];
                }
                catch
                {
                    res = bcUnknown;
                }
                return res;
            }

            /// <summary>
            /// Формування фонового кольору комірок для виведення
            /// заголовка матриці інцидентності та стовпця з
            /// характеристикою посередництва для кожної
            /// вершини ґрафа
            /// </summary>
            private void grid01_CellFormatting(object sender,
                DataGridViewCellFormattingEventArgs e)
            {
                if (e.RowIndex == 0 || e.ColumnIndex == 0)
                    e.CellStyle.BackColor = Color.Gainsboro;

                else
                {
                    int cc = ((DataGridView)sender).ColumnCount - 1;
                    if (e.ColumnIndex == cc)
                        try
                        {
                            if (sender.Equals(grid01))
                                e.CellStyle.BackColor =
                                    VertexColor(g01[e.RowIndex],
                                                Color.White,
                                                Color.Lime);
                            else
                                e.CellStyle.BackColor =
                                    VertexColor(g02[e.RowIndex],
                                    Color.White,
                                    Color.Lime);
                        }
                        catch
                        {
                            e.CellStyle.BackColor = Color.White;
                        }
                }
            }

        }
    }
```

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Павленко, В. Н. Требования и порядок оформления учебных и научно-исследовательских документов / В. Н. Павленко, А. С. Набатов, И. М. Тараненко. – Харків : ХАІ, 2007. – 66 с.

2. Graph file formats [Электронный ресурс]. – Режим доступа к ресурсу: http://www2.sta.uwi.edu/~mbernard/research_files/fileformats.pdf.

3. Trivial Graph Format [Электронный ресурс]. – Режим доступа к ресурсу: https://ru.wikipedia.org/wiki/Trivial_Graph_Format.

4. Microsoft Automatic Graph Layout [Электронный ресурс]. – Режим доступа к ресурсу: https://github.com/Microsoft/automatic-graph-layout.

5. Betweenness centrality [Электронный ресурс]. – Режим доступа к ресурсу: https://en.wikipedia.org/wiki/Betweenness_centrality.

6. TGF. Input and Output [Электронный ресурс] – Режим доступа к ресурсу: http://docs.yworks.com/yfiles/doc/developers-guide/tgf.html

ЗМІСТ

Навчальне видання

Мокляк Микола Григорович,
Соколова Євгенія Віталіївна,
Лучшев Павло Олександрович
та ін.

ОСНОВИ ПРОГРАМУВАННЯ

Частина 3

Редактор Н. М. Сікульська

Зв. план, 2018
Підписано до видання 30.11.2018
Ум. друк. арк. 2,8. Обл.-вид. арк. 3,13. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001