

Тема 16. Зв'язність графів. Ейлерів і гамільтонів цикли

План лекції

Шляхи та цикли. Зв'язність. Термінологія

Шляхи в графах та ізоморфізм

Оцінка кількості ребер простого графа

Критерій двочастковості графа

Обходи графів

Ейлерів цикл у графі

Гамільтонів цикл у графі

Шляхи та цикли. Зв'язність. Термінологія

Шляхом довжиною r із вершини u в вершину v в неорієнтованому графі називають послідовність ребер $e_1=\{x_0, x_1\}$, $e_2=\{x_1, x_2\}$, ..., $e_r=\{x_{r-1}, x_r\}$, де $x_0=u$, $x_r=v$, r – натуральне число. Отже, шлях довжини r має r ребер, причому ребро враховують стільки разів, скільки воно входить у шлях. Вершини u та v називають *крайніми*, а решту вершин шляху – *внутрішніми*.

Циклом у неорієнтованому графі називають шлях, який з'єднує вершину саму із собою, тобто $u=v$.

У простому графі шлях можна задати послідовністю вершин, через які він проходить: $x_0, x_1, x_2, \dots, x_{r-1}, x_r$.

Шлях або цикл називають *простим*, якщо він не містить повторюваних ребер.

Іноді використовують такі означення. Шлях називають *елементарним*, якщо він не містить повторюваних вершин. Цикл називають *елементарним*, якщо він не містить повторюваних вершин, окрім першої та останньої.

Говорять, що шлях із крайніми вершинами u та v з'єднує ці вершини. Шлях, що з'єднує вершини u та v , позначають як $\langle u, v \rangle$ та називають $\langle u, v \rangle$ -шляхом.

Приклад. На рис. 1 зображено простий граф. У ньому a, d, c, f, e – простий шлях довжиною 4, оскільки пари $\{a, d\}$, $\{d, c\}$, $\{c, f\}$ та $\{f, e\}$ – ребра. Однак, d, e, c, b – не шлях, бо пара $\{e, c\}$ – не ребро. Зазначимо, що b, c, f, e, b – цикл довжиною 4, позаяк $\{b, c\}$, $\{c, f\}$, $\{f, e\}$, та $\{e, b\}$ – ребра та цей шлях починається й закінчується в одній і тій самій вершині b . Шлях a, b, e, d, a, b , довжина якого дорівнює 5, не простий, оскільки він двічі проходить через ребро $\{a, b\}$.

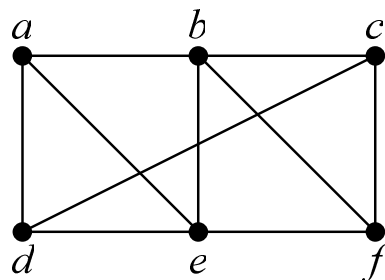


Рис. 1

Неорієнтований граф називають *зв'язним*, якщо будь-які дві його вершини з'єднані шляхом. Граф називають *незв'язним*, якщо він не є зв'язним. Незв'язний граф складається із двох або більше зв'язних підграфів, кожна пара з яких не має спільних вершин. Ці зв'язні підграфи називають *компонентами зв'язності* або просто *компонентами* графа.

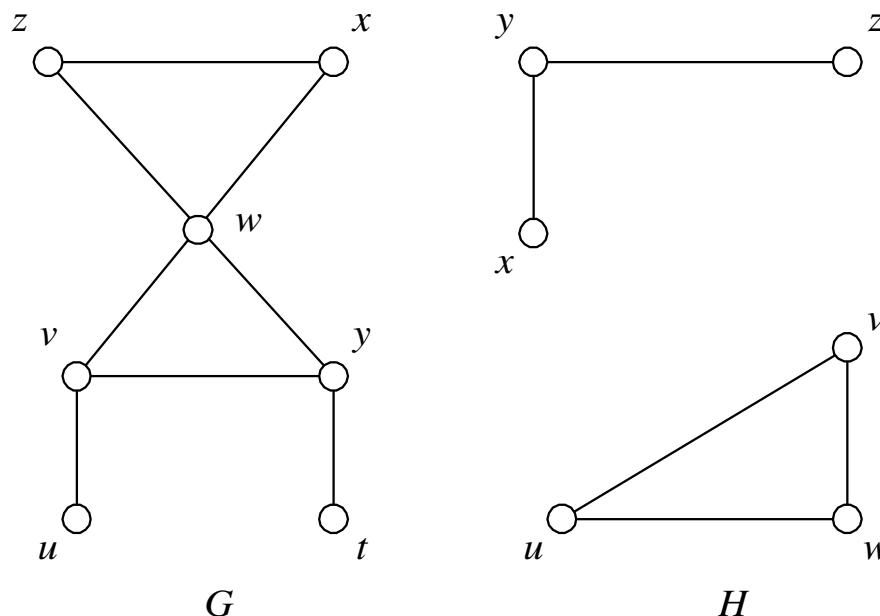


Рис. 2

Приклад. Граф G на рис. 2 зв'язний; граф H – незв'язний, оскільки не існує шляху $\langle y, v \rangle$. Граф H має дві компоненти.

Віддаллю $d(u, v)$ між вершинами u та v у неорієнтованому графі G називають довжину найкоротшого $\langle u, v \rangle$ -шляху, а сам цей шлях називають *геодезичним*.

Теорема. Між кожною парою різних вершин зв'язного неорієнтованого графа існує простий шлях.

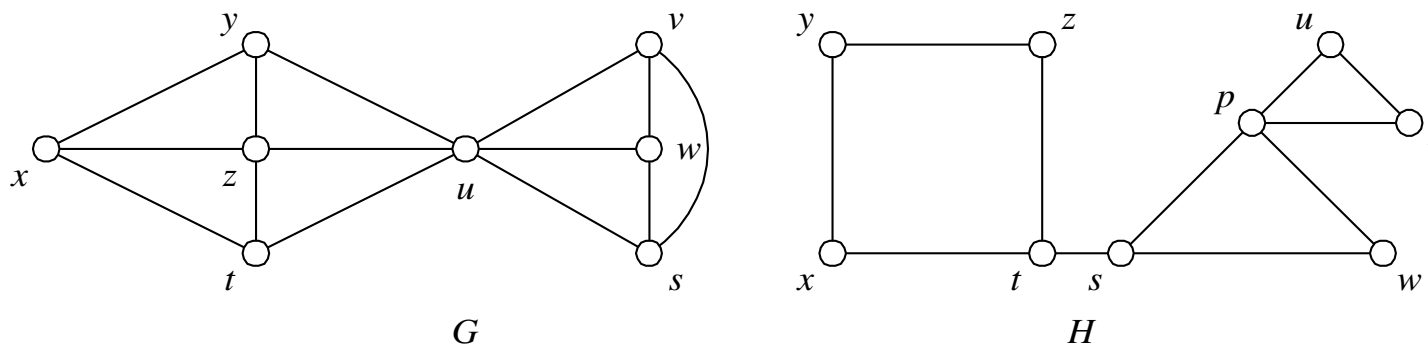


Рис. 3

Вершину u простого графа G називають *точкою з'єднання*, якщо граф G в разі її видалення матиме більше компонент, ніж даний граф G . Зокрема, якщо G – зв'язний граф і u – точка з'єднання, то G без вершини u – незв'язний. Нагадаємо, що вершину u при цьому видаляють разом із інцидентними їй ребрами. Ребро графа G називають *мостом*, якщо його видалення збільшує кількість компонент. Отже, точки з'єднання й мости – це свого роду «вузькі місця» простого графа.

Приклад. Граф G із рис. 3 має точку з'єднання u і не має мостів, а граф H – три точки з'єднання t, s, p та один міст $\{t, s\}$.

Для орієнтованого графа вводять поняття *орієнтованого шляху* (або просто *шляху*) з вершини u у вершину v . Це скінченна послідовність дуг $e_1 = (x_0, x_1)$, $e_2 = (x_1, x_2)$, ..., $e_r = (x_{r-1}, x_r)$, де $x_0 = u$, $x_r = v$. Вершини u та v називають, як і в неорієнтованому графі, *крайніми*, а решту вершин шляху – *внутрішніми*. *Довжиною* шляху називають кількість дуг, з яких він складається.

Орієнтованим *циклом* називають орієнтований шлях, який з'єднує вершину саму із собою, тобто $u=v$. Орієнтований шлях або цикл називають *простим*, якщо жодна дуга не міститься в ньому більше одного разу.

Для орієнтованого графа поняття зв'язності вводять по-різному, залежно від того, чи враховують напрям дуг.

Орієнтований граф називають *сильно зв'язним*, якщо для будь-яких його різних вершин u та v існують орієнтовані шляхи від u до v та від v до u . Отже, для сильної зв'язності орієнтованого графа повинна існувати послідовність дуг з урахуванням орієнтації від будь-якої вершини графа до будь-якої іншої.

Орієнтований граф може не бути сильно зв'язним, але може бути, так би мовити, „в одному цілому”. У зв'язку із цим дамо таке означення. Орієнтований граф називають *слабко зв'язним*, якщо існує шлях між будь-якими двома різними вершинами у відповідному йому неорієнтованому графі (тобто без урахування напрямку дуг).

Зрозуміло, що сильно зв'язний граф водночас і слабо зв'язний.

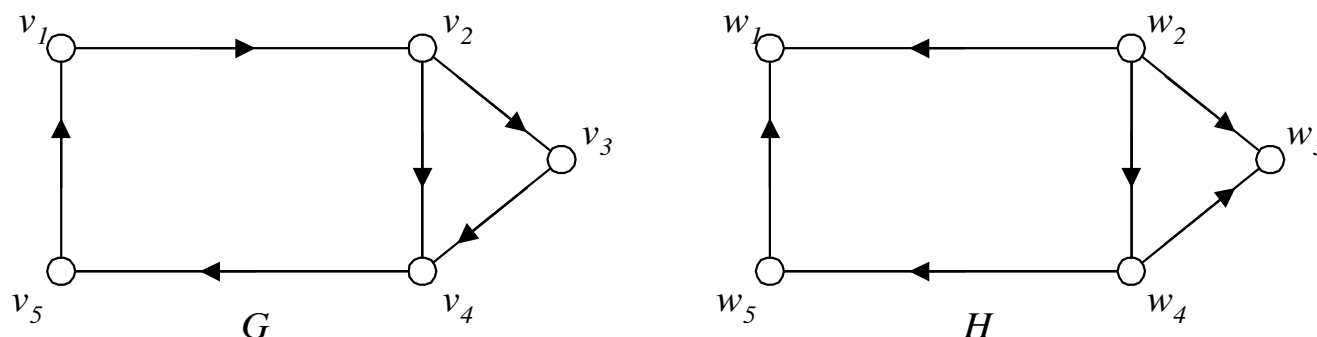


Рис. 4

Приклад. На рис. 4 зображено графи G й H . Граф G – сильно зв'язний. Граф H – слабо зв'язний; він не сильно зв'язний, бо не існує орієнтованого шляху від w_1 до w_2 .

Терміни, які тут уведено, не уніфіковано й вони по-різному означаються різними авторами. Ми дотримуємося термінології з книги [Kenneth H. Rosen. Discrete Mathematics and Its Applications. McGraw-Hill, Inc, 2012]. Уводяться поняття шляху, циклу, простих шляху й циклу в неорієнтованому графі. Аналогічно ці поняття вводяться й для орієнтованих графів. З нашого погляду перевагою цієї термінології є її простота: небагато нових термінів і вони стосуються всіх типів графів, як неорієнтованих, так і орієнтованих. Зазначимо (див. табл.), що в літературі також поширений альтернативний набір термінів [В.Ф. Емеличев и др. Лекции по теории графов. М.: Наука, 1990], [Ю.В. Капитанова та ін. Основи дискретної математики. К.: Наукова думка, 2002]. Очевидно, особливо уважним потрібно бути з термінами „цикл”, „простий цикл” і „шлях”, оскільки їх використовують у різних значеннях. Зокрема, простий шлях і цикл ми визначаємо як такі, що не містять повторюваних ребер, тоді як у іншій системі термінів їх визначають як такі, що не містять повторюваних вершин (ми такий шлях і цикл назвали елементарними). Це має значення для формулювання означень і теорем, у яких використовуються терміни й поняття шляху та циклу.

Таблиця

<i>Властивості шляху</i>	<i>Терміни у лекціях стосуються довільних графів</i>	<i>Неорієнтовані графи [В.Емеличев, Ю.Капітонова]</i>	<i>Орієнтовані графи</i>	
			<i>[В.Емеличев]</i>	<i>[Ю.Капітонова]</i>
Найзагальніший випадок: ребра (дуги) можуть повторюватись	<i>Шлях</i>	<i>Маршрут</i>	<i>Орієнтований маршрут або маршрут</i>	<i>Маршрут або шлях</i>
Усі ребра (дуги) різні	<i>Простий шлях</i>	<i>Ланцюг</i>	<i>Ланцюг</i>	<i>Орланцюг</i>
Усі вершини різні, окрім, можливо, крайніх	<i>Елементарний шлях</i>	<i>Простий ланцюг</i>	<i>Шлях</i>	<i>Простий орланцюг</i>
Перша й остання вершини співпадають	<i>Цикл</i>	<i>Циклічний маршрут</i>	<i>Циклічний маршрут</i>	—
Перша й остання вершини співпадають, а ребра (дуги) не повторюються	<i>Простий цикл</i>	<i>Цикл</i>	—	<i>Замкнутий орланцюг</i>
Перша й остання вершини співпадають, а усі інші вершини різні	<i>Елементарний цикл</i>	<i>Простий цикл</i>	<i>Контур</i>	<i>Цикл</i>

Шляхи в графах та ізоморфізм

Є різні способи використати концепцію шляху та циклу для визначення, чи є два графи ізоморфними. Наприклад, існування простого циклу певної довжини – корисний інваріант, яким можна скористатись для доведення, що два графи не ізоморфні.

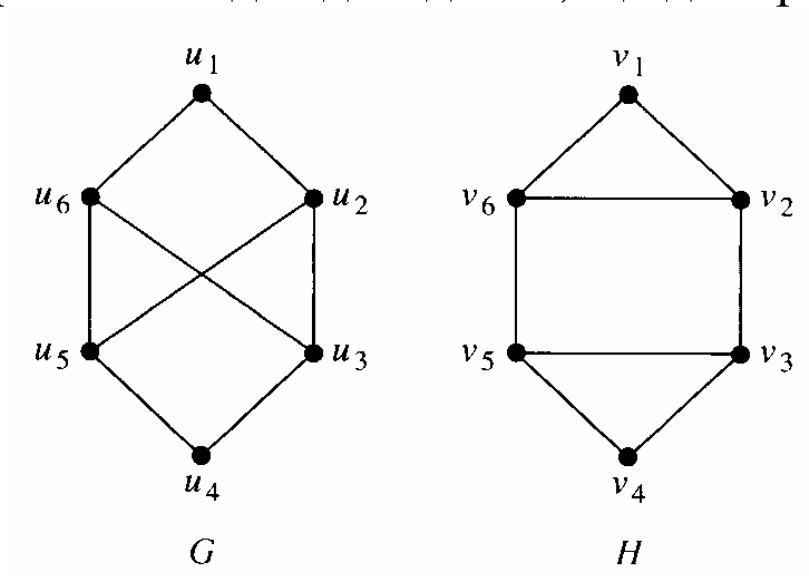


Рис. 5

Приклад. Дослідимо, чи ізоморфні графи G та H на рис. 5. Обидва ці графи мають по шість вершин і по вісім ребер. Кожний з них має чотири вершини степеня три і дві вершини степеня два. Отже, три інваріанти – кількість вершин, кількість ребер і степені вершин – усі узгоджені в цих двох графах. Проте, граф H має простий цикл довжиною три, нехай v_1, v_2, v_6, v_1 , тоді як граф G не має простого циклу довжиною три. Цикл певної довжини є інваріантом. Отже, графи G та H не ізоморфні.

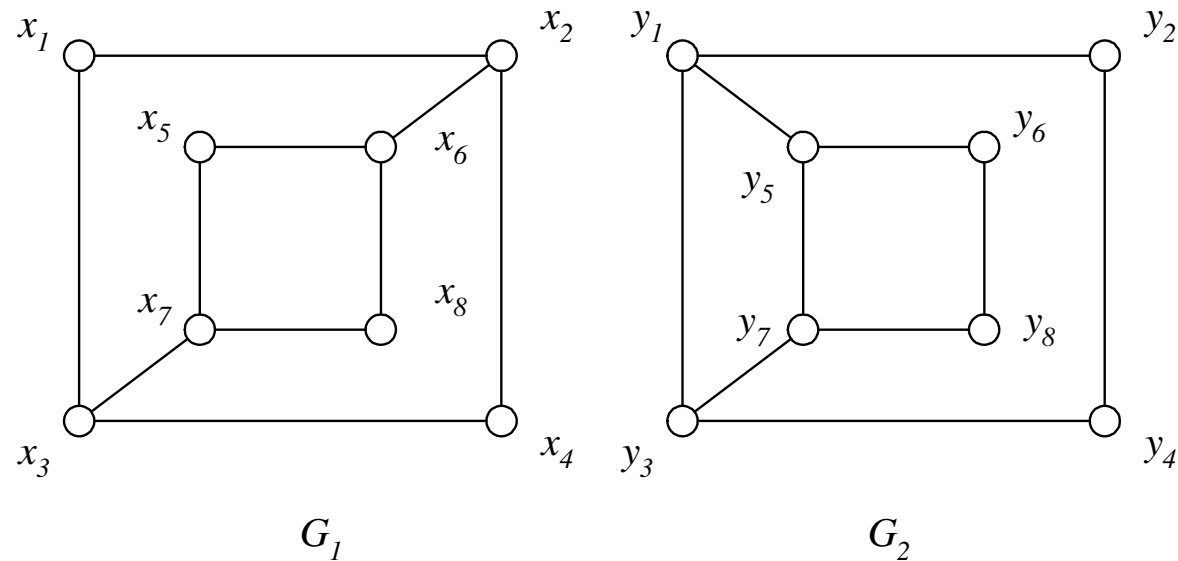


Рис. 6

Існують і інші інваріанти, пов'язані зі шляхами й циклами. Наприклад, інваріантом є простий цикл, який проходить через вершини певного степеня. Наприклад, графи G_1 та G_2 на рис. 6 неізоморфні, бо граф G_2 має простий цикл, який проходить тільки через вершини степеня три, а граф G_1 такого циклу не має.

Оцінка кількості ребер простого графа

Очевидно, що кількість ребер у зв'язному простому графі з n вершинами не перевищує кількості ребер у графі K_n , тобто $n(n-1)/2$. Але скільки може бути ребер у простому графі з n вершинами й фіксованою кількістю k компонент?

Теорема. Якщо простий граф G має n вершин і k компонент, то кількість m його ребер задовольняє нерівності

$$n - k \leq m \leq (1/2)(n - k)(n - k + 1).$$

Доведення. Доведемо спочатку верхню оцінку. Нехай G – простий граф з n вершинами, k компонентами й максимальною для таких графів кількістю ребер m_{\max} . Очевидно, що кожна компонента графа G – повний граф. Нехай K_p, K_q – дві компоненти, $p \geq q > 1$, v – вершина з компоненти K_q . Вилучимо з графа всі ребра, інцидентні вершині v , і з'єднаємо вершину v ребром із кожною вершиною з компоненти K_p . Кількість вершин і компонент при цьому не зміниться, а кількість ребер зросте на величину $p - (q - 1) = p - q + 1 > 1$, що неможливо, бо граф G має максимально можливу кількість ребер. Звідси випливає, лише одна компонента графа G являє собою повний граф із більшою ніж 1 кількістю вершин $n - (k - 1) = n - k + 1$, а решта $(k - 1)$ компонент – ізольовані вершини. Отже,

$$m_{\max} = (1/2)(n - k)(n - k + 1).$$

Доведемо нижню оцінку. Доведення здійснюємо математичною індукцією за кількістю ребер m . Для $m=0$ твердження очевидне, оскільки тоді $k=n$ і, отже, $0 \leq 0$. Нехай тепер $m > 0$, і для графів із кількістю ребер меншою, ніж m , нижня оцінка справджується. Припустимо, що граф G має найменшу можливу кількість ребер m_{\min} серед усіх простих графів з n вершинами й k компонентами. Вилучивши довільне ребро, отримаємо граф з n вершинами, $k+1$ компонентою й $m_{\min} - 1$ ребром. Для цього графа справджується припущення індукції: $n - (k + 1) \leq m_{\min} - 1$, звідки випливає нерівність $n - k \leq m_{\min}$.

Питання. Чи можна стверджувати, що коли простий граф з n вершинами містить більше ніж $(n - 1)(n - 2)/2$ ребер, то він обов'язково зв'язний?

Критерій двочастковості графа

Д. Кьоніг сформулював критерій двочастковості простого графа в термінах довжин простих циклів.

Теорема (Кьоніг, 1936 р.). Для того, щоб граф G був двочастковим, необхідно й достатньо, щоб він не містив простих циклів із непарною довжиною.

Доведення. Необхідність. Нехай G – двочастковий граф, C – один із його простих циклів довжиною k . Пройдемо всі ребра цього циклу, починаючи з вершини v . Зробивши k кроків, повернемось у вершину v . Оскільки кінці кожного ребра містяться в різних підмножинах вершин, то k – парне число.

Достатність. Нехай зв'язний простий граф $G = (V, E)$ з $n > 1$ вершинами не має простих циклів із непарною довжиною та $v \in V$. Побудуємо розбиття $V = A \cup B$ ($A \cap B = \emptyset$) так. Довільну вершину $x \in V$ долучимо до множини A , якщо віддаль $d(x, v)$ – парна, а ні, то до множини B . Залишилося довести, що породжені підграфи $G(A)$ та $G(B)$ порожні (тобто не мають ребер). Припустимо, що це не так, тобто існують дві суміжні вершини u та w , які належать одній множині. Тоді жодна із цих вершин не співпадає з v , бо $v \in A$, а всі вершини, суміжні з v , належать множині B . Нехай $U = \langle u, v \rangle$ та $W = \langle w, v \rangle$ – геодезичні шляхи, v_1 – остання (якщо починати від v) зі спільних вершин цих шляхів (рис. 7). Позначимо як X_U та Y_U відповідно частини шляху U від v до v_1 та від v_1 до u . Аналогічно, як X_W та Y_W позначимо відповідно частини шляху W від v до v_1 та від v_1 до w . Очевидно, що довжини шляхів X_U та X_W однакові (бо шляхи U та W геодезичні). Отже, довжини шляхів Y_U та Y_W мають один тип парності (позаяк за припущенням вершини u та w належать одній множині, то їх віддалі від вершини v мають один тип парності). Але тоді об'єднання шляхів Y_U та Y_W та ребра $\{u, w\}$ являє собою простий цикл із непарною довжиною. Суперечність. Теорему доведено.

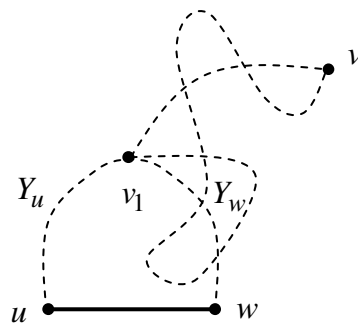


Рис. 7

Доведення теореми Кьоніга підказує простий спосіб розпізнавання двочастковості графа. Цей спосіб ґрунтується на простому алгоритмі, який називають пошуком ушир. Множину вершин, суміжних із вершиною v , будемо називати *оточенням* вершини v . *Пошук ушир* так приписує вершинам графа номери $0, 1, 2, \dots$.

Починають із довільної вершини, приписують їй номер 0 . Кожній вершині з оточення вершини 0 приписують номер 1 . Тепер розглядають по чергово оточення всіх вершин із номером 1 , і всім вершинам, що належать цим оточенням і ще не мають номера, приписують номер 2 . Розглядають оточення всіх вершин із номером 2 та продовжують процес присвоювання номерів, доки це можливо. Якщо даний граф $G=(V, E)$ зв'язний, то пошук ушир занумерує всі його вершини.

Далі розіб'ємо множину вершин V на дві підмножини – A та B . До множини A долучимо всі вершини з парними номерами (та 0), а до множини B – з непарними. Розглянемо породжені підграфи $G(A)$ та $G(B)$. Якщо обидва вони порожні (достатньо перевірити, що всі пари вершин з однаковими номерами не суміжні), то G – двочастковий граф, а ні – то не двочастковий.

Зазначимо, що існує й інший, більш розповсюджений варіант пошуку вшир. Він відрізняється тим, що всі вершини отримують різні номери.

Обходи графів

Існує багато алгоритмів на неорієнтованих графах, які ґрунтуються на систематичному переборі їх вершин або обході вершин, під час якого кожна вершина одержує унікальний порядковий номер. Методи обходу вершин графа називають методами пошуку.

Деревом називають зв'язний граф, який не містить простих циклів. Під час обходу графа ми певним чином позначаємо ребра, які утворять дерево обходу.

Пошук углиб у простому зв'язному графі

Опишемо метод пошуку в простому зв'язному графі. Цей метод називають пошуком вглиб, або DFS-методом (від англ. Depth First Search).

Нехай $G = (V, E)$ – простий зв'язний граф, усі вершини якого позначені 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* (англ. stack – стіг).

Зі стеку можна вилучити тільки той елемент, котрий 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 називають *верхівкою стеку*. DFS-номер вершини x позначають як $DFS(x)$.

Алгоритм пошуку вглиб у простому зв'язному графі

- Крок 1. Почати з довільної вершини v_s . Виконати $\text{DFS}(v_s) := 1$. Включити вершину v_s у стек.
- Крок 2. Розглянути вершину, яка знаходиться у верхівці стеку, нехай це буде вершина x .
Якщо для всіх вершин, суміжних із вершиною x , уже визначені DFS-номери, то перейти до кроку 4, інакше – до кроку 3.
- Крок 3. Нехай $\{x, y\}$ – ребро, у якому номер $\text{DFS}(y)$ не визначений. Позначити це ребро потовщеною суцільною лінією (або внести в список ребер дерева), визначити $\text{DFS}(y)$ як черговий DFS-номер, включити вершину y у стек й перейти до кроку 2.
- Крок 4. Вилучити вершину x зі стеку. Якщо стек порожній, то зупинитись, інакше – перейти до кроку 2.

Щоб вибір номерів був однозначним, доцільно домовитись, що вершини, суміжні з тією, яка вже отримала DFS-номер, аналізують за зростанням їхньої нумерації (або в алфавітному порядку). Динаміку роботи алгоритму зручно відображати за допомогою таблиці з чотирма стовпцями: вершина, DFS-номер, уміст стеку, список ребер дерева. Її називають *протоколом обходу* графа пошуком вглиб.

Приклад. Виконаємо обхід графа на рис. 8 пошуком вглиб, починаючи з вершини b . Розв'язок подано на рис. 9; протокол пошуку вглиб подано в таблиці поруч з рисунками. У цій таблиці в третьому стовпці вважаємо, що верхівка стеку праворуч. У четвертому

стовпці таблиці накопичуються ребра дерева пошуку вглиб (його іноді називають *глибинним деревом*). На рисунку це дерево позначено потовщеними лініями.

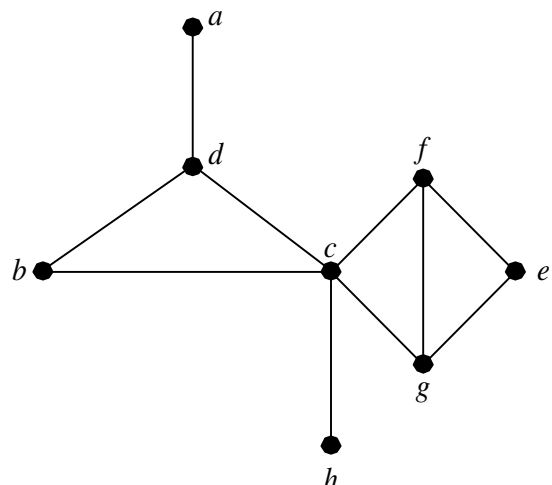


Рис. 8

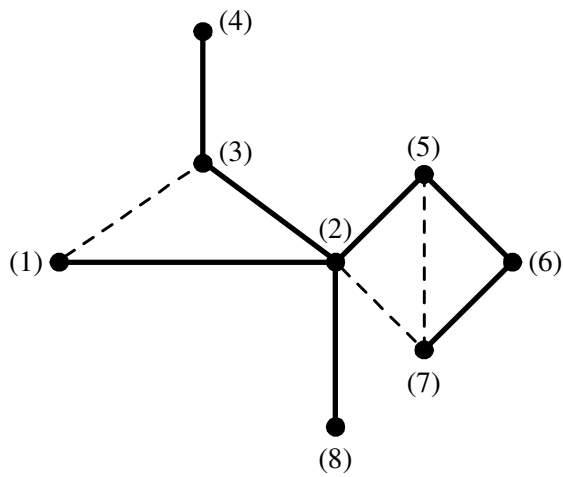




Рис. 9

Таблиця. Протокол обходу графа пошуком вглиб

Вершина	DFS-номер	Уміст стеку □  	Список ребер дерева
<i>b</i>	1	<i>b</i>	—
<i>c</i>	2	<i>bc</i>	{ <i>b</i> , <i>c</i> }
<i>d</i>	3	<i>bcd</i>	{ <i>c</i> , <i>d</i> }
<i>a</i>	4	<i>bcda</i>	{ <i>d</i> , <i>a</i> }
—	—	<i>bcd</i>	—
—	—	<i>bc</i>	—
<i>f</i>	5	<i>bcf</i>	{ <i>c</i> , <i>f</i> }
<i>e</i>	6	<i>bcfe</i>	{ <i>f</i> , <i>e</i> }
<i>g</i>	7	<i>bcfeg</i>	{ <i>e</i> , <i>g</i> }
—	—	<i>bcfe</i>	—
—	—	<i>bcf</i>	—
—	—	<i>bc</i>	—
<i>h</i>	8	<i>bch</i>	{ <i>c</i> , <i>h</i> }
—	—	<i>bc</i>	—
—	—	<i>b</i>	—
—	—	∅	—

Пошук ушир у простому зв'язному графі

У процесі *пошуку вшир* вершини графа проглядають в іншій послідовності, ніж у методі пошуку *вглиб*, і їм надають BFS-номери (від англ. Breadth First Search). BFS-номер вершини x позначають як $BFS(x)$. Назва пояснюється тим, що під час пошуку рухаються вшир, а не вглиб: спочатку проглядаються всі сусідні вершини, після цього – сусіди сусідів і так далі.

У ході реалізації алгоритму використовують структуру даних для збереження множин, яку називають *чергою* (англ. queue). 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 в *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 із черги). Використання вершини полягає в перегляді одразу всіх іще не відвіданих її сусідів. Усю процедуру подано нижче.

Алгоритм пошуку вшир у простому зв'язному графі

- Крок 1. Почати з довільної вершини v_s . Виконати $\text{BFS}(v_s) := 1$. Включити вершину v_s у хвіст черги.
- Крок 2. Розглянути вершину, яка знаходиться в голові черги, нехай це буде вершина x . Якщо для всіх вершин, суміжних із вершиною x , уже визначені BFS-номери, то перейти до кроку 4, інакше – до кроку 3.
- Крок 3. Нехай $\{x, y\}$ – ребро, у якому номер $\text{BFS}(y)$ не визначений. Позначити це ребро потовщеною суцільною лінією (або внести в список ребер дерева), визначити $\text{BFS}(y)$ як черговий BFS-номер, включити вершину y у хвіст черги й перейти до кроку 2.
- Крок 4. Вилучити вершину x із голови черги. Якщо черга порожня, то зупинитись, інакше – перейти до кроку 2.

Щоб результат виконання алгоритму був однозначним, вершини, які суміжні з вершиною x , аналізують за зростанням їх нумерації (або в алфавітному порядку). Динаміку роботи алгоритму пошуку вшир також зручно відображати за допомогою протоколу обходу. Він аналогічний попередньому й відрізняється лише третім стовпцем: тепер це – уміст черги (уважаємо, що голова черги ліворуч, а хвіст – праворуч).

Приклад. Виконаємо обхід графа на рис. 10 пошуком вшир, починаючи з вершини *b*. Розв’язок зображено на рис. 11; протокол пошуку вшир подано в таблиці поруч із рисунками. У четвертому стовпці таблиці накопичуються ребра дерева пошуку вшир. На рисунку це дерево позначено потовщеними лініями. Пошук углиб і пошук ушир – важлива складова проектування алгоритмів на графах.

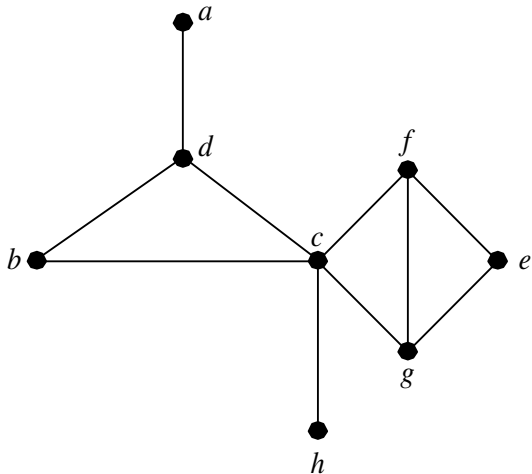


Рис. 10

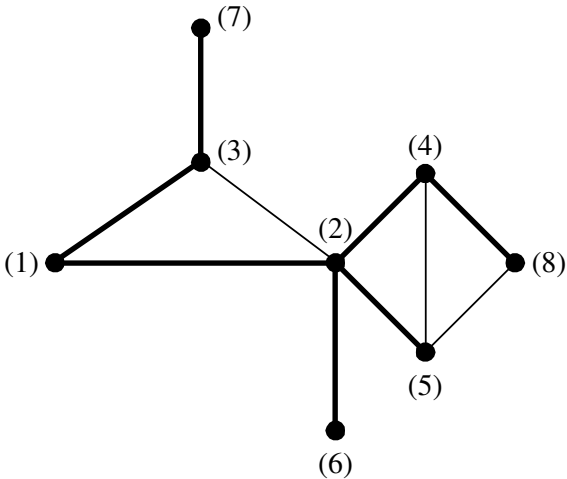


Рис. 11

Таблиця. Протокол обходу графа пошуком вшир

Вершина	BFS-номер	Вміст черги ← ←	Список ребер дерева
<i>b</i>	1	<i>b</i>	—
<i>c</i>	2	<i>bc</i>	{ <i>b</i> , <i>c</i> }
<i>d</i>	3	<i>bcd</i>	{ <i>b</i> , <i>d</i> }
—	—	<i>cd</i>	—
<i>f</i>	4	<i>cdf</i>	{ <i>c</i> , <i>f</i> }
<i>g</i>	5	<i>cdfg</i>	{ <i>c</i> , <i>g</i> }
<i>h</i>	6	<i>cdfgh</i>	{ <i>c</i> , <i>h</i> }
—	—	<i>dfgh</i>	—
<i>a</i>	7	<i>dfgha</i>	{ <i>d</i> , <i>a</i> }
—	—	<i>fgha</i>	—
<i>e</i>	8	<i>fghae</i>	{ <i>f</i> , <i>e</i> }
—	—	<i>ghae</i>	—
—	—	<i>hae</i>	—
—	—	<i>ae</i>	—
—	—	<i>e</i>	—
—	—	∅	—

Ейлерів цикл у графі

Початок теорії графів як розділу математики пов'язують із задачею про кенігсберзькі мости. Сім мостів міста Кенігсберга (нині – Калінінград у Росії) було розміщено на річці Прегель так, як зображено на рис. 12. Чи можна, починаючи з якоїсь точки міста, пройти через усі мости точно по одному разу й повернутись у початкову точку? Швейцарський математик Л. Ейлер розв'язав цю задачу. Його розв'язання, опубліковане 1736 р., було першим явним застосуванням теорії графів. Ейлер поставив у відповідність плану міста мультиграф G , вершини якого відповідають чотирьом частинам A, B, C, D міста, а ребра – мостам. Цей мультиграф зображено на рис. 13.

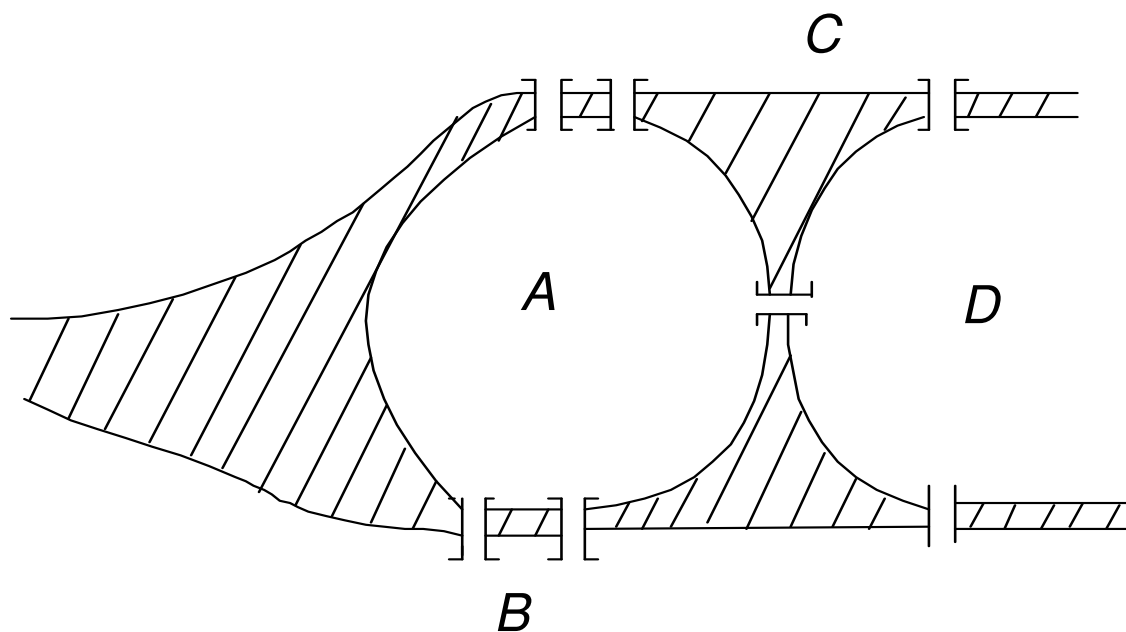


Рис. 12

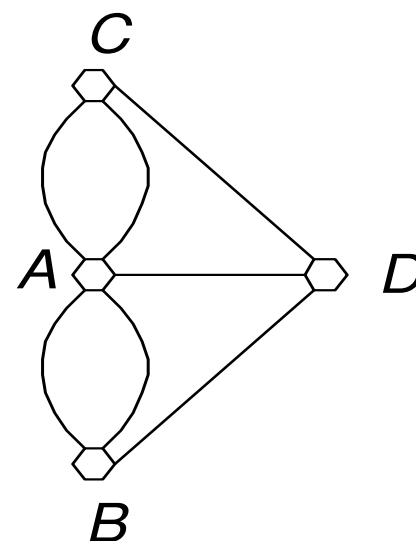


Рис. 13

Отже, задачу про кенігсберзькі мости мовою теорії графів можна сформулювати так: чи існує в мультиграфі G простий цикл, який містить усі ребра цього мультиграфа? Ейлер довів нерозв'язність задачі про кенігсберзькі мости. Нагадаємо, що в простому циклі ребра не повторюються, а вершини можуть повторюватись.

Ейлеровим циклом у зв'язному мультиграфі G називають простий цикл, який містить усі ребра графа. Ейлеровим шляхом у зв'язному мультиграфі G називають простий шлях, який містить усі ребра графа.

Інакше кажучи, ейлерів цикл проходить через кожне ребро графа точно один раз. Отже, його довжина дорівнює кількості ребер графа.

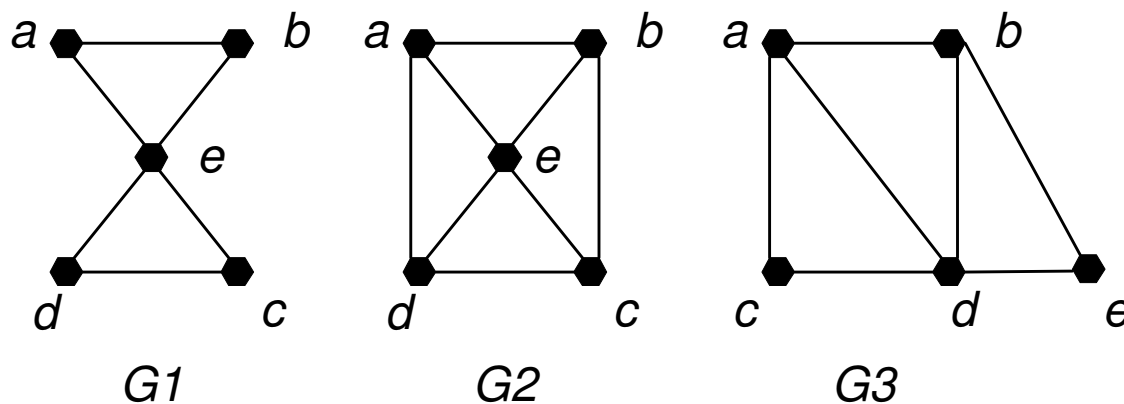


Рис. 14

Приклад. На рис. 14 проілюстровано концепцію ейлерових циклів і шляхів. Граф G_1 має ейлерів цикл, наприклад, a, e, c, d, e, b, a ; граф G_3 не має ейлерового циклу, але має ейлерів шлях: a, c, d, e, b, d, a, b ; граф G_2 не має ні ейлерового циклу, ні ейлерового шляху.

Існує простий критерій (необхідна й достатня умова) для виявлення, чи має граф ейлерів цикл.

Теорема. Зв'язний мультиграф має ейлерів цикл тоді й лише тоді, коли степені всіх його вершин парні.

Доведення. Необхідність. Нехай у графі G існує ейлерів цикл. Тоді він проходить через кожную вершину графа та входить до неї по одному ребру, а виходить по іншому. Це означає, що кожна вершина інцидентна парній кількості ребер ейлерового циклу. Оскільки такий цикл містить усі ребра графа G , то звідси випливає парність степенів усіх його вершин.

Достатність. Припустимо тепер, що всі вершини графа G мають парний степінь. Почнемо шлях P_1 із довільної вершини v_1 і продовжимо його, наскільки це можливо, вибираючи щоразу нове ребро. Позаяк степені всіх вершин парні, то, увійшовши в будь-яку вершину, відмінну від v_1 , ми завжди маємо можливість вийти з неї через іще не пройдене ребро. Тому шлях P_1 можна продовжити, додавши це ребро. Отже, побудова шляху P_1 завершиться у вершині v_1 , тобто P_1 обов'язково виявиться циклом.

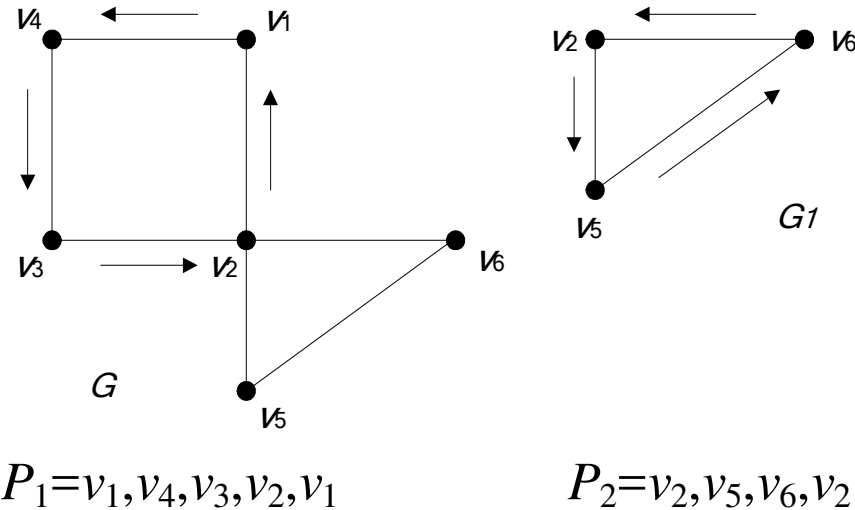


Рис. 15

Якщо з'ясується, що цикл P_1 містить усі ребра графа G , то це ейлерів цикл. У протилежному випадку вилучимо з G всі ребра циклу P_1 і всі вершини, інцидентні лише вилученим ребрам. Отримаємо якийсь граф G_1 . Зауважимо, що граф G_1 може і не бути зв'язним. Оскільки P_1 та G мають вершини лише парних степенів, то, очевидно, і граф G_1 матиме цю властивість. Окрім того, позаяк граф G зв'язний, то графи P_1 та G_1 мають принаймні одну спільну вершину (на рисунку це вершина v_2).

Тепер із вершини v_2 будуємо цикл P_2 у графі G_1 аналогічно до того, як ми будували цикл P_1 у графі G . Цикл P_2 вставимо в цикл P_1 на місце вершини v_2 . Одержимо цикл P_3 . Описані побудови показано на рис. 15. Цикл $P_3 = v_1, v_4, v_3, P_2, v_1 = v_1, v_4, v_3, v_2, v_5, v_6, v_2, v_1$ – ейлерів. Якби він виявився не ейлеровим, то потрібно продовжити аналогічні побудови й отримати ще більший цикл. Цей процес закінчиться побудовою ейлерового циклу. Теорему доведено.

Зазначимо, що доведення достатності конструктивне: подано алгоритм побудови ейлерового циклу. Його іноді називають алгоритмом об'єднання циклів.

Обчислювальна складність цього алгоритму становить $O(m)$.

Існує й інший алгоритм побудови ейлерового циклу, який дає змогу побудувати цей цикл одразу. Це алгоритм Флері (Fleuri).

Алгоритм Флері побудови ейлерового циклу.

Робота полягає в нумерації ребер у процесі побудови ейлерового циклу.

Крок 1 (початок). Починаємо з довільної вершини u та присвоюємо довільному ребру $\{u, v\}$ номер 1. Викреслюємо ребро $\{u, v\}$ і переходимо у вершину v .

Крок 2 (ітерація). Нехай w – вершина, у яку ми перейшли на попередній ітерації, k – останній присвоєний номер. Вибраємо довільне ребро, інцидентне вершині w , причому міст вибираємо лише тоді, коли немає інших можливостей. Присвоюємо вибраному ребру номер $(k+1)$ і викреслюємо його.

Крок 3 (закінчення). Цей процес закінчуємо, коли всі ребра графа викреслено та пронумеровано – ці номери задають послідовність ребер в ейлеровому циклі.

За складністю цей алгоритм гірший за алгоритм об'єднання циклів, його оцінка складності становить $O(m^2)$. Хоча виглядає, що маємо $O(m)$, але пошук мостів – це ще $O(m)$, тому виходить $O(m^2)$. Якщо використовувати спеціальний алгоритм пошуку мостів, то буде трохи ефективніше: $O(m \cdot \log^3 m \cdot \log \log m)$.

Повертаючись до задачі про кенігсберзькі мости, виявляємо, що мультиграф на рис. 2 має всі вершини непарного степеня. Отже, цей мультиграф не має ейлерового циклу, тому неможливо пройти кожний міст по одному разу й повернутись в початкову точку шляху.

Теорема. Зв'язний мультиграф має ейлерів шлях, але не має ейлерового циклу тоді й лише тоді, коли він має точно дві вершини непарного степеня.

Доведення. Необхідність. Припустимо, що зв'язний мультиграф має ейлерів шлях від u до v , але не має ейлерового циклу. Перше ребро циклу додає одиницю до степеня вершини u . Щоразу, коли шлях проходить через вершину u , він додаватиме до її степеня двійку. Останнє ребро шляху додасть одиницю до степеня вершини v , а кожне проходження шляху через вершину v додаватиме до її степеня двійку. Отже, вершини u та v мають непарний степінь. Усі інші вершини – парного степеня, бо шлях додає двійку до степеня вершини коли проходить через неї.

Достатність. Припустимо, що граф G має точно дві вершини, нехай u та v , з непарним степенем. Розглянемо граф G' , одержаний з графа G додаванням нового ребра $\{u, v\}$. Кожна вершина графа G' має парний степінь, отже, G' має ейлерів цикл. Тепер вилучимо нове ребро і одержимо ейлерів шлях у графі G . Теорему доведено.

Зазначимо, що будь-який ейлерів шлях починається в одній із цих двох вершин непарного степеня, а закінчується в іншій.

Оскільки мультиграф для кенігсберзьких мостів має чотири вершини з непарними степенями, можна дійти висновку про неможливість пройти кожний міст по одному разу, навіть якщо не потрібно повертатись у початкову точку.

Граф, який має ейлерів цикл, часто називають ейлеровим графом.

Гамільтонів цикл у графі

Ми дослідили необхідні й достатні умови існування шляхів і циклів, які містять кожне ребро зв'язного мультиграфа G точно один раз. Чи можемо ми зробити це саме для шляху й циклу, які містять кожну вершину графа точно один раз?

Шлях у неорієнтованому зв'язному графі, який проходить через кожну вершину графа точно один раз називають *гамільтоновим шляхом*. Цикл, який проходить через кожну вершину точно один раз (окрім першої і останньої) називають *гамільтоновим циклом*.

Отже, шлях $x_0, x_1, \dots, x_{n-1}, x_n$ у графі $G=(V, E)$ – гамільтонів шлях, якщо $V = \{x_0, x_1, \dots, x_{n-1}, x_n\}$ і $x_i \neq x_j$ для $0 \leq i < j \leq n$. Цикл $x_0, x_1, \dots, x_{n-1}, x_n, x_0$ (тут $n \geq 2$) у графі G гамільтонів цикл, якщо $x_0, x_1, \dots, x_{n-1}, x_n$ – гамільтонів шлях.

Зазначимо, що гамільтонові цикл і шлях, узагалі кажучи, не містять усіх ребер графа.

За означенням гамільтонові цикли розглядають лише для графів, які мають не менше трьох вершин (отже, довжина гамільтонового циклу не менша трьох).

Якщо використовувати терміни «елементарний шлях» і «елементарний цикл» (не містять повторюваних вершин), то можна так означити гамільтонів шлях і цикл.

Гамільтоновим шляхом називають елементарний шлях, який містить усі вершини графа. Гамільтоновим циклом називають елементарний цикл, який містить усі вершини графа.

Термін „гамільтонів” у цих означеннях походить від імені відомого ірландського математика Вільяма Ровена Гамільтона (W. R. Hamilton), який 1857 року запропонував гру „Навколосвітня подорож”. Кожній із двадцяти вершин додекаедра (правильного дванадцятигранника, грані якого – п’ятикутники) приписано назву одного з великих міст світу. Потрібно, розпочавши з довільного міста, відвідати решту 19 міст точно один раз, і повернутись у початкове місто. Перехід дозволено ребрами додекаедра.

Приклад. Ту саму задачу можна зобразити й на площині (рис. 16) Вона зводиться до відшукування в графі гамільтонівського циклу. Один із можливих розв’язків показано потовщеними лініями.

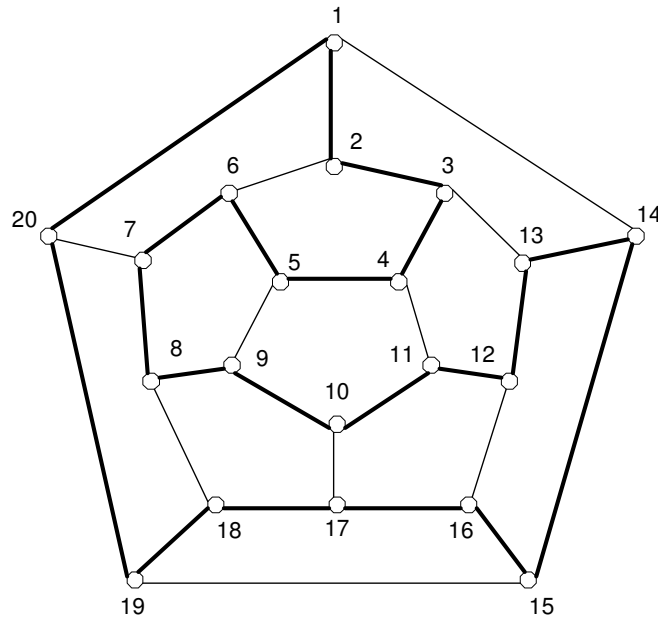


Рис. 16

Не всі зв'язні графи мають гамільтонів цикл хоча б тому, що такі графи мають бути двозв'язними (тобто граф, який має точки з'єднання, не може мати гамільтонів циклу). Приклад графа, зображеного на рис. 17, свідчить, що для наявності гамільтонів циклу двозв'язності недостатньо.

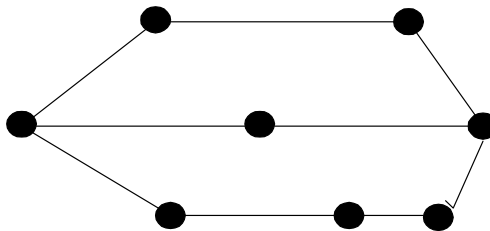


Рис. 17

Незважаючи на зовнішню подібність формулювань задач про існування ейлерового й гамільтонів циклів, ці задачі принципово різні. Використовуючи результати попереднього підрозділу, легко виявити, чи має граф ейлерів цикл, і, якщо має, то

побудувати його. Ситуація для гамільтонового циклу істотно інша. Відповісти на питання, чи має граф гамільтонів цикл, зазвичай, дуже важко. Вивчення достатніх умов наявності в графі гамільтонового циклу – один із важливих напрямів у теорії графів. Інтуїтивно зрозуміло, що граф із багатьма ребрами, достатньо рівномірно розподіленими, з великими шансами має гамільтонів цикл.

Теорема (Г. Дірак, 1952 р.). Якщо G – зв'язний простий граф з $n \geq 3$ вершинами, і для кожної вершини v виконується нерівність $\deg(v) \geq n/2$, то граф G має гамільтонів цикл.

Теорема (О. Оре, 1960 р.). Якщо G – зв'язний простий граф з $n \geq 3$ вершинами, такий, що $\deg(u) + \deg(v) \geq n$ для кожної пари несуміжних вершин u та v , то граф G має гамільтонів цикл.

Теорему Дірака можна довести як наслідок теореми Оре, бо із виконання умов теореми Дірака випливає виконання умов теореми Оре.

Обидві теореми (і Дірака, і Оре) дають лише ДОСТАТНІ умови існування гамільтонового циклу. Наприклад, умови жодної з цих теорем не можна застосувати до графа C_5 , хоча він має гамільтонів цикл.

Завдання. Навести приклад графа, до якого не можна застосувати теорему Дірака, але можна застосувати теорему Оре.

Як знайти гамільтонів цикл або переконатись, що його немає? Очевидний алгоритм, який можна застосувати, – це „повний перебір усіх можливостей”, тобто $n!$ перестановок усіх вершин графа й перевірок. Зрозуміло, що такий спосіб практичного застосування не знаходить. Розгляд практично застосовного алгоритму *бектрекінг* відкладемо до наступного розділу.

Найкращий алгоритм для знаходження гамільтонів циклу має експоненціальну складність (за кількістю вершин графа).

Знаходження поліноміального алгоритму розв’язання цієї проблеми було би дуже великим досягненням, бо задача знаходження гамільтонів циклу NP -повна. Отже, із існування такого алгоритму впливає, що багато інших на вигляд важкорозв’язних задач можуть бути розв’язані поліноміальними алгоритмами. Але саме це дає підстави думати, що такого алгоритму не існує (хоча це і не доведено).

Граф, який містить гамільтонів цикл, часто називають гамільтонів графом.