Лекція 24. Обхід графів та властивості матриць

24.1. Складність алгоритмів

Цією лекцією ми починаємо розгляд деяких алгоритмів на графах, тому дамо деякі означення, що стосуються алгоритмів. Під час аналізу алгоритму нас буде цікавити передусім його складність, під якою розуміємо час виконання відповідної програми на комп'ютері. Зрозуміло, що цей показник суттєво залежить від типу комп'ютера. Щоб висновки про складність були достатньо універсальними, будемо вважати, що всі обчислення виконуються на якомусь ідеалізованому комп'ютері.

Означення 24.1. **Складність алгоритму** розв'язання задачі – це функція f, яка кожному невід'ємному цілому числу n ставить у відповідність час роботи f(n) алгоритму в найгіршому випадку на входах довжиною n. Час роботи алгоритму вимірюють у кроках (операціях), виконуваних на ідеалізованому комп'ютері.

Наприклад, якщо алгоритм приймає як вхідні дані довільний граф G = (V, E), то під довжиною входу можна розуміти |V| чи max(|V|, |E|).

Аналіз ефективності алгоритмів полягає в з'ясуванні того, як швидко зростає функція f(n) зі збільшенням n. Для порівняння швидкості зростання двох функцій g(n) і f(n) (з невід'ємнини значеннями) використовують такі позначення:

- $f(n) = O(g(n))$ означає, що існують додатна стала c та натуральне число n_0 такі, що $f(n) \le cg(n)$ для всіх $n \ge n_0$;
- $f(n) = \Omega(g(n))$ означає, що існують додатна стала c та натуральне число n_0 такі, що $f(n) \ge cg(n)$ для всіх $n \ge n_0$.

Для аналізу ефективності алгоритмів використовують O-символіку. Вираз «складність алгоритму дорівнює (становить) O(g(n))» має саме такий зміст, як у наведеному вище означенні. Зокрема, складність O(1) означає, що час роботи відповідного алгоритму не залежить від довжини входу.

Алгоритм зі складністю O(n), де n – довжина входу, називають **лінійним**. Такий алгоритм для переважної більшості задач найліпший (за порядком) щодо складності.

Алгоритм, складність якого дорівнює O(p(n)), де p(n) – поліном, називають **поліноміальним**. Часто замість O(p(n)) пишуть $O(n^a)$, де a – константа. Всі задачі дискретної математики, які вважають важкими для алгоритмічного розв'язування, нині не мають поліноміальних алгоритмів. Крім того, поняття «поліноміальний алгоритм» – це найпоширеніша формалізація поняття «**ефективний алгоритм**».

Алгоритми, часова складність яких не піддається подібній оцінці, називають **експоненціальними**. Більшість експоненціальних алгоритмів – це просто варіанти повного перебору, а поліноміальні алгоритми здебільшого можна побудувати лише тоді, коли вдається заглибитись у суть задачі. Задачу називають **важкорозв'язною**, якщо для її розв'язання не існує поліноміального алгоритму.

24.2. Обхід графів. Пошук углиб

Існує багато алгоритмів на графах, які ґрунтуються на систематичному переборі їх вершин або обході вершин, під час якого кожна вершина отримує унікальний порядковий номер. Виділяється два основних алгоритми обходів графів або пошуку у графах – це пошук углиб та пошук вшир.

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

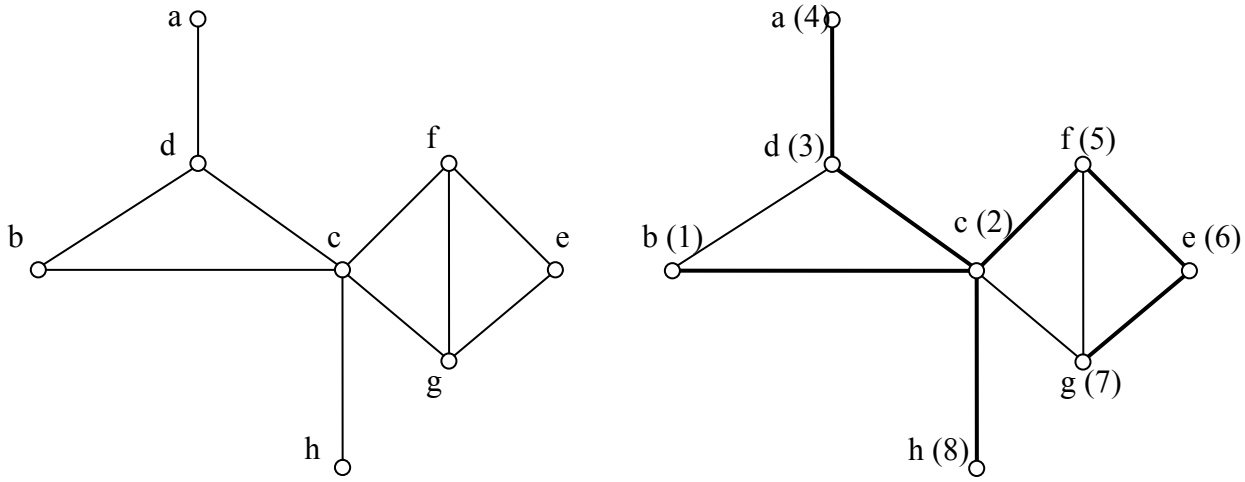
відбувається з одного кінця, який називається верхівкою стеку. DFS-номер вершини x позначають DFS(x).

Алгоритм пошуку вглиб у простому зв'язаному графі.

1. Почати з довільної вершини v_s. Виконати DFS(v_s) := 1. Включити цю вершину в стек.
2. Розглянути вершину у верхівці стеку: нехай це вершина x. Якщо всі ребра, інцидентні вершині x, позначено, то перейти до кроку 4, інакше – до кроку 3.
3. Нехай (x,y) – ребро, в якому DFS(y) не визначено, то позначити ребро (x,y) потовщеною суцільною лінією, визначити DFS(y) як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.
4. Виключити вершину x із 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 їх порядкових номерів (або в алфавітному порядку). Динаміку роботи алгоритму зручно відображати за допомогою таблиці з трьома стовпцями: вершина, DFS-номер, уміст стеку. Її називають протоколом обходу графа пошуком вглиб.

Розглянемо приклад роботи алгоритму на графі з рис. 24.1,а. Розв'язок подано на рис. 24.1,б, а протокол обходу – у табл. 24.1. Алгоритм починає працювати з вершини b графа.



а) б)

Рис. 24.1

Вершина	DFS-номер	Вміст стеку
b	1	b
c	2	bc
d	3	bcd
a	4	bcda
-	-	bcd
-	-	bc
f	5	bcf
e	6	bcfe
g	7	bcfeg
-	-	bcfe
-	-	bcf
-	-	bc
h	8	bch

-	-	bc
-	-	b
-	-	∅

Табл. 24.1

24.3. Пошук вшир

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

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

Алгоритм пошуку вшир у простому зв'язаному графі

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

Щоб результат виконання алгоритму був однозначним, вершини, які суміжні з вершиною x, аналізують за зростанням їх порядкових номерів (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 (вважаємо, що голова черги ліворуч, а хвіст – праворуч).

Для прикладу виконаємо обхід графа, який зображено на рис. 24.1, а, починаючи з вершини b. Результат представлено на рис. 24.2, а протокол обходу – в таблиці 24.2.

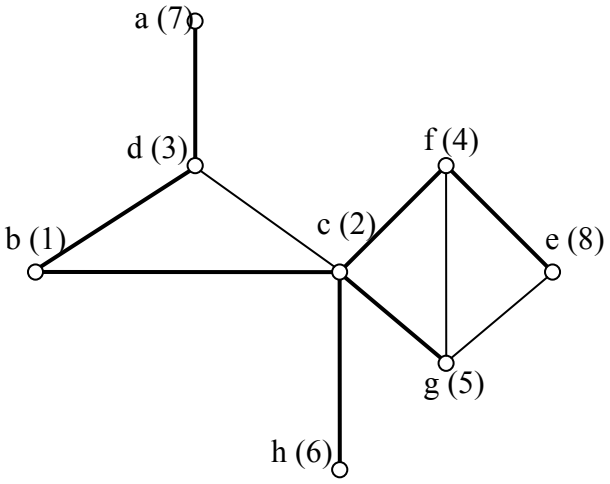


Рис. 24.2.

Вершина	BFS-номер	Вміст черги
b	1	b
c	2	bc
d	3	bcd
-	-	cd
f	4	cdf
g	5	cdfg
h	6	cdfgh
-	-	dfgh
a	7	dfgha
-	-	fgha
8	e	fghae
-	-	ghae
-	-	hae
-	-	ae
-	-	e
-	-	∅

Табл. 24.2.

У процесі роботи наведених алгоритмів будується дерево T пошуку відповідно вглиб і вшир – на рис. 24.1,б та 24.2. його виділено потовщеними лініями. Обчислювальна складність обох алгоритмів обходу однакова й у разі подання графа списками суміжності становить O(m+n), де m – кількість ребер, а n – кількість вершин графа.

24.4. Властивості матриць графів

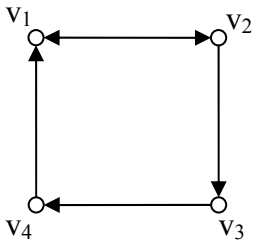
Багато інформації відносно графа G можна представити в зручній формі, використовуючи матриці, відповідні графу G. Матрицю суміжності Δ(G) графа G можна використовувати для підрахування кількості різних маршрутів (шляхів для орграфів) у G. Сама матриця Δ задає ребра G, тобто маршрути довжиною 1. Виявляється, що матриця Δ^k (k-степінь Δ) задає число маршрутів довжини k.

Теорема 24.1. Елемент (i, j) = $\delta_{ij}^{(k)}$ матриці Δ^k графа G дорівнює кількості маршрутів довжини k з v_i у v_j.

Доведення. Методом індукції по довжині k. Для k=1 теорема очевидна: матриця суміжності задає маршрут довжиною 1. Нехай для деякого k теорема вірна, тобто елемент $\delta_{ij}^{(k)}$ матриці Δ^k дорівнює кількості маршрутів довжини k з v_i у v_j. Доведемо її для k+1. Довільний маршрут довжини k+1 з v_i у v_j складається з ребра, яке прямує з v_i у суміжну з нею вершиною v_r, а потім маршруту довжини r з v_r у v_j. Кількість маршрутів довжини r+1 з v_i у v_j, які проходять на першому кроці через v_r, дорівнює $\delta_{ir}\delta_{rj}^{(k)}$ (якщо ребро з v_i у v_r немає, то $\delta_{ir} = 0$ і $\delta_{ir}\delta_{rj}^{(k)} = 0$, а якщо таке ребро є, то $\delta_{ir}\delta_{rj}^{(k)} = \delta_{rj}^{(k)}$, тому що $\delta_{ir} = 1$). Загальна кількість маршрутів довжини k+1 з v_i у v_j отримаємо, якщо просумуємо цю величину по всім r: $\sum_{r=1}^{n} \delta_{ir}\delta_{ir}^{(k)}$. Ця сума дорівнює елементу (i,j) добутку матриці Δ та Δ^k, тобто елементу (i,j) матриці Δ^{k+1}, що й доводить теорему. ►

Наслідок. Елемент (i,j) матриці $\Delta+\Delta^2+\ldots+\Delta^k$ графа G дорівнює кількості всіх маршрутів довжини не більше за k з v_i у v_j.

Ця теорема справедлива як для неорієнтованих, так і для орієнтованих графів. Наприклад, на рис. 24.3 наведено орграф та його матриці суміжності Δ, Δ^2, Δ^3 та Δ^4.



$$\Delta = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad \Delta^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$\Delta^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array} \qquad \Delta^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Рис. 24.3.

Корисною виявляється нова матриця – **матриця відстаней** $D=||d_{ij}||$, де $d_{ij} = d(i,j)$ – відстань від v_i до v_j, яка визначається як довжина найкоротшого маршруту з v_i у v_j. (Нагадаємо, що величина d_{ij} не визначена, якщо маршрут з v_i у v_j не існує).

Теорема 24.2. Нехай граф G має матрицю суміжності Δ та матрицю відстаней D. Тоді, якщо величина d_{ij}, $i \neq j$, визначена, то вона дорівнює найменшому k, для якого елемент (i,j) в Δ^k, тобто $\delta_{rj}^{(k)}$, не дорівнює 0.

Доведення пропонується провести самостійно.

Слідуючи цій теоремі, можна побудувати матрицю відстаней, послідовно підводячи у степінь матрицю суміжності графа. Побудуємо матрицю відстаней для графа з рис. 24.3.

1. Матриця відстаней має нулі на головній діагоналі та спочатку співпадає з матрицею суміжності, тобто вона містить всі маршрути довжиною 1. Інші елементи матриці відстаней поки що не визначені.

2. Матриця Δ^2 вказує на всі маршруті довжиною 2. Невизначеним елементам матриці відстаней $||d_{ij}||$ присвоюється значення 2, якщо $\delta_{rj}^{(2)} \neq 0$.

3. Тим елементам $||d_{ij}||$, які ще не визначені, присвоюємо значення 3, якщо елементи Δ^3 $\delta_{rj}^{(3)} \neq 0$.

Тепер матриця відстаней повністю визначена.

1) $D(G) = \begin{pmatrix} 0 & 1 & \infty & \infty \\ 1 & 0 & 1 & \infty \\ \infty & \infty & 0 & 1 \\ 1 & \infty & \infty & 0 \end{pmatrix}$ 2) $D(G) = \begin{pmatrix} 0 & 1 & 2 & \infty \\ 1 & 0 & 1 & 2 \\ 2 & \infty & 0 & 1 \\ 1 & 2 & \infty & 0 \end{pmatrix}$ 3) $D(G) = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 3 & 0 & 1 \\ 1 & 2 & 3 & 0 \end{pmatrix}$

Теорема 24.3. Для того, щоб n-вершиний граф з матрицею суміжності Δ мав хоча б один цикл, необхідно й достатньо, щоб матриця $K = \Delta^2+\ldots+\Delta^k$ мала хоча б один не нульовий діагональний елемент.

Матриця досяжності $R(G) = ||r_{ij}||$ визначається наступним чином: $r_{ij} = 1$, якщо v_j є досяжною з v_i, і $r_{ij} = 0$ в протилежному випадку. Довільна вершина досяжна сама із себе, тому $r_{ii} = 1$ для всіх i. Матриця досяжності може бути отримана за допомогою матриці суміжності.

Теорема 24.4. Нехай Δ - матриця суміжності і R – матриці досяжності графа G з n вершинами. Тоді

$$R = B(I + \Delta + \Delta^2 + \ldots + \Delta^{n-1}) = B[(I + \Delta)^{n-1}],$$

де B – булеве перетворення ($B: N \to \{0,1\}$; B(x) = 0, якщо x=0, та B(x) = 1, якщо x>0), а I – одинична матриця.

Доведення. Дійсно, якщо v_j є досяжною з v_i, то існує простий ланцюг з v_i у v_j. Довжина цього маршруту не перебільшує n-1, оскільки у простому ланцюгу вершини не повторюються. Відповідно, елемент матриці $I + \Delta + \Delta^2 + \ldots + \Delta^{n-1}$ буде ненульовим, звідки й слідує теорема. ►

У наступній теоремі показано застосування матриці досяжності як метода визначення зв'язності орграфів.

Теорема 24.5. Нехай орграф G має матрицю досяжності R та матрицю суміжності Δ. Тоді:

1) G сильно зв'язан тоді й тільки тоді, коли R = J, де J – одинична матриця.
2) G однобічно зв'язан тоді й тільки тоді, коли B(R + R') = J, де R' – транспонована матриця R;
3) G слабо зв'язан тоді й тільки тоді, коли $B[(I + \Delta + \Delta')^{n-1}] = J$, де Δ' – транспонована матриця Δ.