5.4 Обхід вершин графу

Існує багато алгоритмів на графах, які ґрунтуються на систематичному переборі їх вершин або обході вершин, під час якого кожна вершина отримує унікальний порядковий номер. Алгоритми обходу вершин графа називають *методами пошуку*. Виділяється два основних алгоритми обходів графів або пошуку у графах — це пошук вшир та пошук вглиб.

5.4.1 Пошук вглиб або DFS-метод (*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). 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 $\mathrm{DFS}(x)$.

Алгоритм пошуку вглиб у простому зв'язному графі

Наведемо кроки алгоритму

Крок 1. Почати з довільної вершини v_s. Виконати $\mathrm{DFS}(v_s):=1$. Включити цю вершину в стек.

Крок 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.

Крок 3. Нехай $\{x, y\}$ — непозначене ребро. Якщо $\mathrm{DFS}(y)$ уже визначено, то позначити ребро $\{x, y\}$ штриховою лінією та перейти до кроку 2. Якщо $\mathrm{DFS}(y)$ не визначено, то позначити ребро $\{x, y\}$ потовщеною суцільною лінією, визначити $\mathrm{DFS}(y)$ як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.

Крок 4. Виключити вершину x зі стеку. Якщо стек порожній, то зупинитись, інакше — перейти до кроку 2.

Щоб результат виконання алгоритму був однозначним, вершини, які суміжні з вершиною v, аналізують за зростанням їх порядкових номерів (або в алфавітному порядку). Динаміку роботи алгоритму зручно відображати за допомогою таблиці з трьома стовпцями: вершина, DFS-номер, вміст стеку. Цю таблицю називають **протоколом обходу** графу пошуком вглиб.

Приклад. Виконаємо обхід пошуком вглиб графу, який зображено на рис. 1, починаючи з вершини *b*. Результат обходу зображено на рис. 2, протокол обходу — в таблиці 1. У цій таблиці в третьому стовпці вважаємо, що верхівка стеку праворуч.

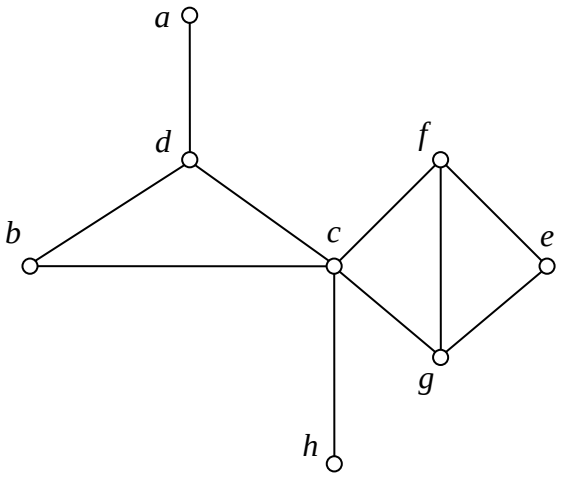


Рис. 1

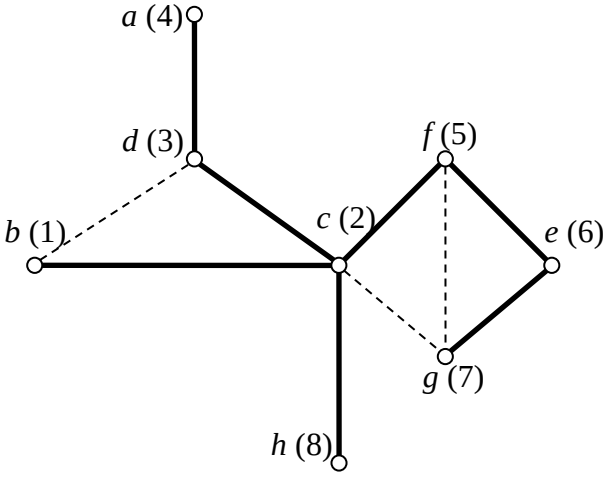


Рис. 2

Таблиця 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*
-	-	∅

5.4.2 Пошук вшир або **BFS-метод** (*breadth first search*)

У процесі пошуку вшир вершини графу G проглядають в іншій послідовності, ніж у методі пошуку вглиб, і їм надають BFS-номери. Під час пошуку рухаються вшир, а не вглиб: спочатку переглядають усі сусідні вершини, після цього — сусіди сусідів і т.д.

У ході реалізації алгоритму використовують структуру даних для збереження множин, яку називають *чергою*. З черги можна вилучити тільки той елемент, який було додано до неї першим: черга працює за принципом «першим прийшов — першим вийшов» (*first in, first out* — скорочено FIFO). Елемент включається у *хвіст* черги, а виключається з її *голови*. Пошук вшир, взагалі 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'язному графі

Наведемо кроки алгоритму

Крок 1. Почати з довільної вершини v_s. Виконати BFS(v_s):=1. Включити вершину v_s у чергу.

Крок 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.

Крок 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 y у чергу й перейти до кроку 2.

Крок 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 — вміст черги (вважаємо, що голова черги ліворуч, а хвіст — праворуч).

Приклад. Виконаємо обхід пошуком вшир графу, який зображено на рис. 1, починаючи з вершини b. Результат обходу зображено на рис. 3, протокол обходу наведено в таблиці 2.

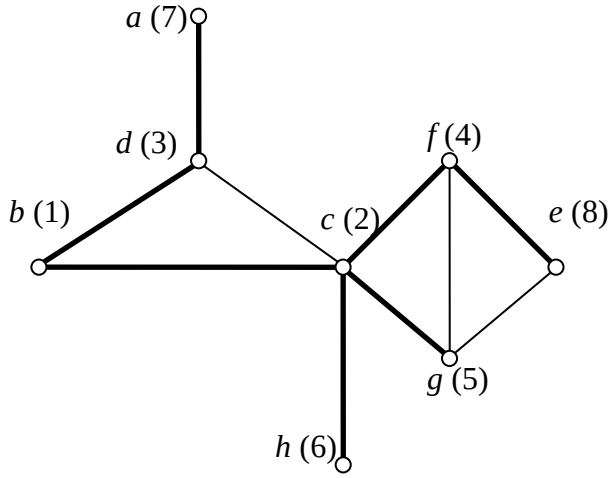


Рис. 3

Таблиця 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*
-	-	∅

У процесі роботи наведених алгоритмів будується *дерево T пошуку* відповідно *вглиб* і *вшир* — на рис. 2 і 3 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.

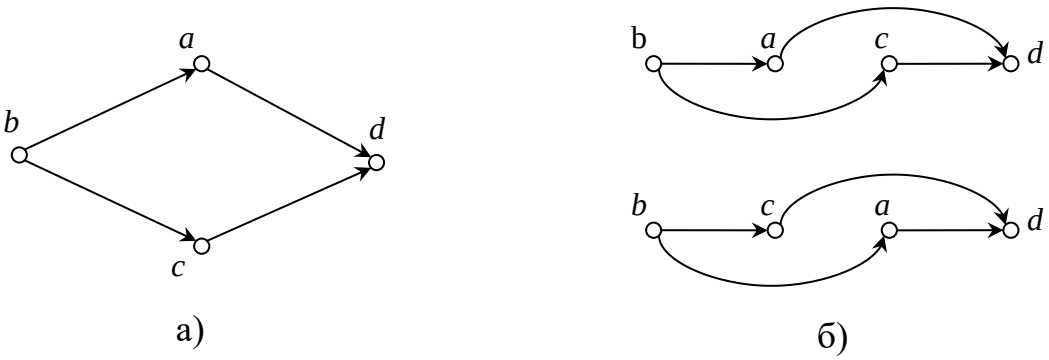
5.4.3 Топологічне сортування

Топологічним сортуванням орієнтованого ациклічного графу $G = (V, E)$ називається таке лінійне впорядкування всіх його вершин, що якщо граф містить ребро (v, u), то вершина v в такому впорядкуванні розташовується до вершини u.

Топологічне сортування графу можна розглядати як таке впорядкування його вершин уздовж горизонтальної лінії, що всі ребра спрямовані зліва направо. Топологічне сортування має багато практичних застосувань, серед яких, наприклад, впорядкування задач в певному робочому плані.

Відзначимо важливість того факту, що граф повинен бути орієнтованим ациклічним графом (directed acyclic graph — DAG). В іншому випадку, коли в графі є хоча б один орієнтований цикл, то в такому графі топологічне сортування неможливе.

В загальному випадку, топологічне сортування, якщо воно існує, то воно не обов'язково буде єдиним. Наприклад, для графу на рис. 4, *а*) існує два можливих впорядкування вершин в рамках топологічного сортування (рис. 4, *б*).



а) б)

Рис. 4. Приклад топологічного сортування

Канонічне застосування топологічного сортування полягає в плануванні послідовності робіт або завдань на основі їх залежностей. Роботи представлені вершинами, і є ребро від *x* до *y*, якщо завдання *x* має бути завершено до того, як можна буде розпочати роботу *y* (наприклад, під час прання білизни пральна машина повинна закінчити, перш ніж ми покладемо білизну в сушильну машину). Тоді топологічне сортування дає порядок виконання завдань. Застосування алгоритмів топологічного сортування було вперше досліджено на початку 1960-х років для планування в управлінні проєктами.

У інформатиці топологічне сортування застосовується у плануванні інструкцій, упорядкуванні оцінки комірки формули під час повторного обчислення значень формул у електронних таблицях, логічного синтезу, визначенні порядку завдань компіляції, серіалізації даних та розв'язанні залежностей символів у компоновщиках. Також топологічне сортування використовується для вирішення, в якому порядку завантажувати таблиці із зовнішніми ключами в базах даних.

Існують різні алгоритми топологічного сортування. Розглянемо один із найпростіших — **алгоритм Кана**.

Ідея алгоритму полягає в наступному. На кожному кроці з множини V вершин графу G вилучається вершина, яка не має попередників (тобто вершина, яка не має вхідних ребер), і поміщається у результуючу впорядковану множину, при цьому з множини ребер E вилучаються усі ребра, що починаються в цій вершині.

Схема алгоритму топологічного сортування Кана

1 **Вхід:** множина вершин $V = \{1, \dots, n\}$
множина ребер $E = \{(v, u)\}$

Вихід: L — впорядкована послідовність вершин графу (черга), що буде містити топологічно відсортовані вершини

$L := (\,),\ V' := V,\ E' := E$

Визначити множину $S \subseteq V'$ вершин, які не мають вхідних ребер

while множина V' не пуста

    В множині S **обрати** вершину v

    Помістити вершину v в кінець черги $L := (L, v)$

    Вилучити вершину v з V': $V' := V' \backslash v$

    Вилучити з множини E' усі ребра, що починаються в вершині v: $E' := E' \backslash \{(v,u) \mid \forall\, u\ (v,u) \in E'\}$

    Оновити множину S (додати до неї усі вершини, попередники яких вилучені з розгляду)

end while

Приклад. Дано граф, зображений на рисунку 5.

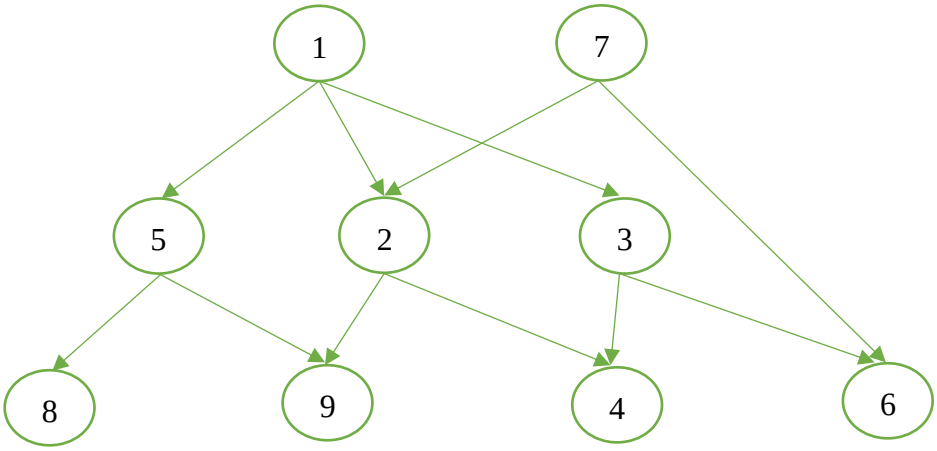


Рис. 5

Вершини 2, 3, 5 є «дочірніми» для вершини 1, а вершина 1 є для них «батьківською» вершиною і т.д.

Виконаємо топологічне сортування даного графа за алгоритмом Кана, тобто впорядкуємо множину вершин графа таким чином, щоб «дочірні» вершини в цій послідовності були розташовані після відповідних «батьківських».

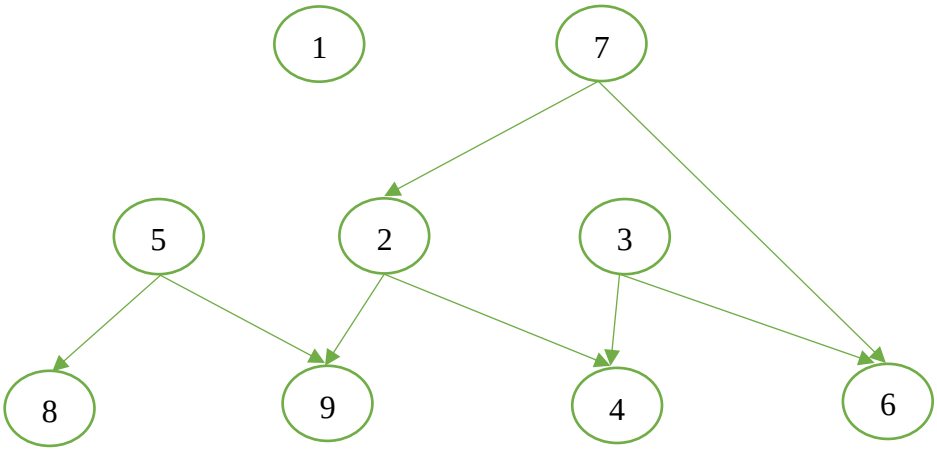
В даному випадку маємо граф, що складається з дев'яти вершин. Вершини 1 та 7 не мають вхідних ребер, отже, складають набір «початкових» вершин *S*.

$L = (\,),\ S = \{1, 7\}$.

Вершину 1 помістимо в кінець черги множини L та вилучимо з множини *S*:

$L = \{1\},\ S = \{7\}$.

Вилучимо всі ребра, що починаються у вершині 1: (1, 5), (1, 2), (1, 3):



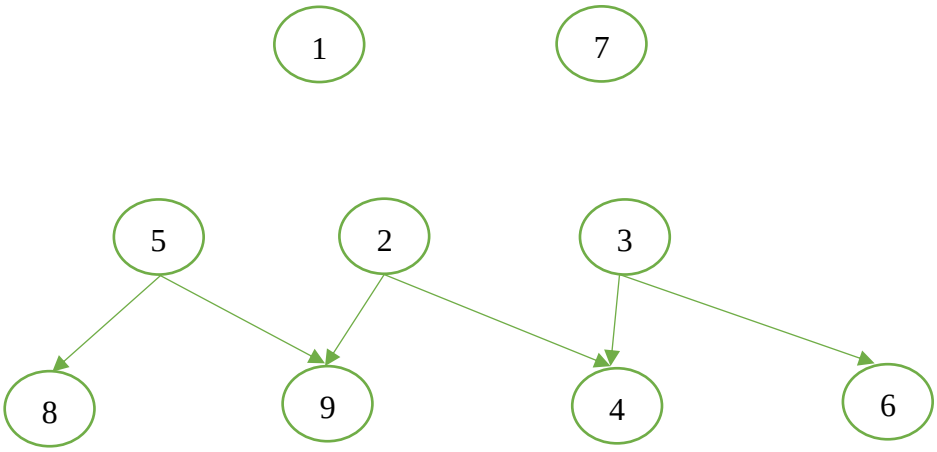
Оновимо множину *S*: додамо до неї вершини 5 та 3 (вершини, що не мають вхідних ребер): $L = \{1\}$, $S = \{7, 5, 3\}$.

Продовжуємо роботу алгоритму з наступною вершиною з множини *S*, а саме, з вершиною 7.

Вершину 7 помістимо в кінець черги множини *L* та вилучимо з множини *S*:

$L = \{1, 7\}$, $S = \{5, 3\}$.

Вилучимо всі ребра, що починаються у вершині 7: (7, 2), (7, 6).



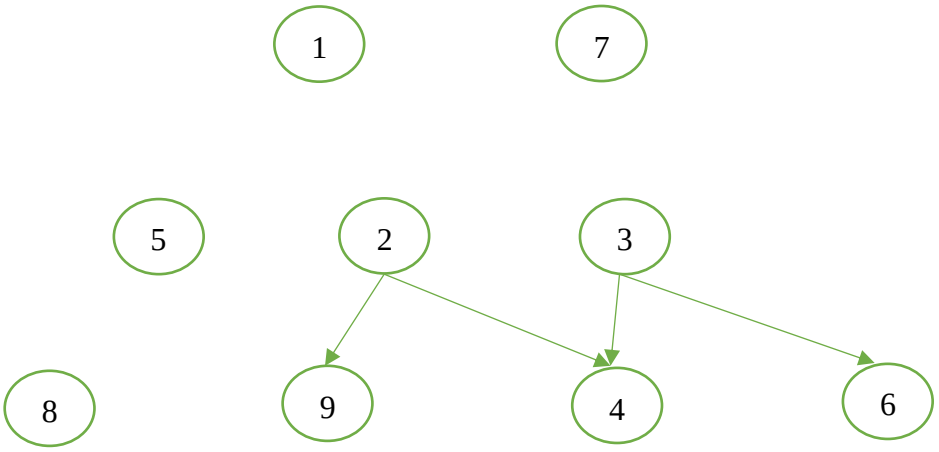
Додамо вершину 2 до множини *S:*

$L = \{1, 7\}$, $S = \{5, 3, 2\}$.

Аналогічно на наступних кроках роботи алгоритму отримаємо:

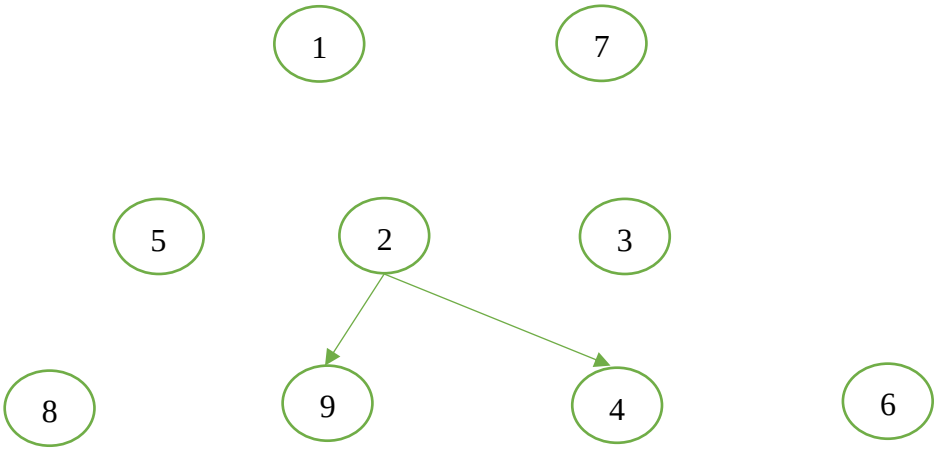
$L = \{1, 7, 5\}$, $S = \{3, 2\}$;

$L = \{1, 7, 5\}$, $S = \{3, 2, 8\}$.



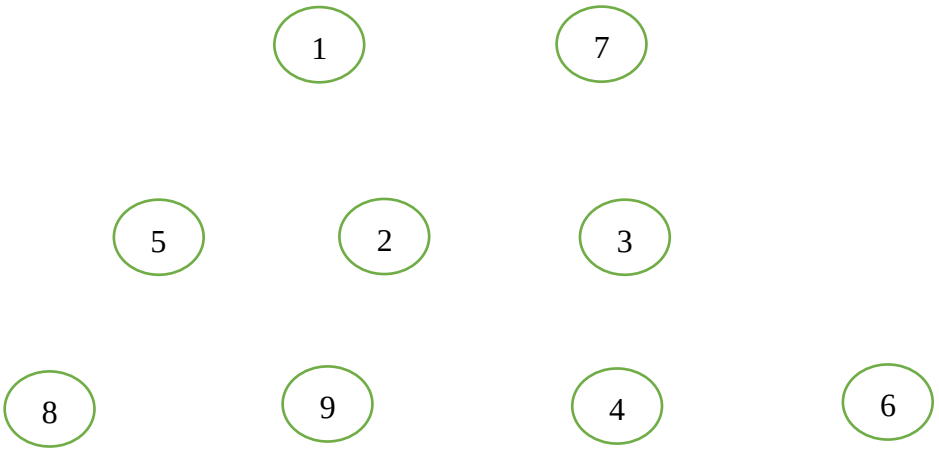
$L = \{1, 7, 5, 3\}$, $S = \{2, 8\}$;

$L = \{1, 7, 5, 3\}$, $S = \{2, 8, 6\}$.



$L = \{1, 7, 5, 3, 2\}$, $S = \{8, 6\}$;

$L = \{1, 7, 5, 3, 2\}$, $S = \{8, 6, 9, 4\}$.



$L = \{1, 7, 5, 3, 2, 8\}$, $S = \{6, 9, 4\}$.

$L = \{1, 7, 5, 3, 2, 8, 6\}$, $S = \{9, 4\}$.

$L = \{1, 7, 5, 3, 2, 8, 6, 9\}$, $S = \{4\}$.

$L = \{1, 7, 5, 3, 2, 8, 6, 9, 4\}$, $S = \{\varnothing\}$.

Таким чином, топологічне сортування в даному випадку має наступний вигляд (рис. 6): $L = \{1, 7, 5, 3, 2, 8, 6, 9, 4\}$.

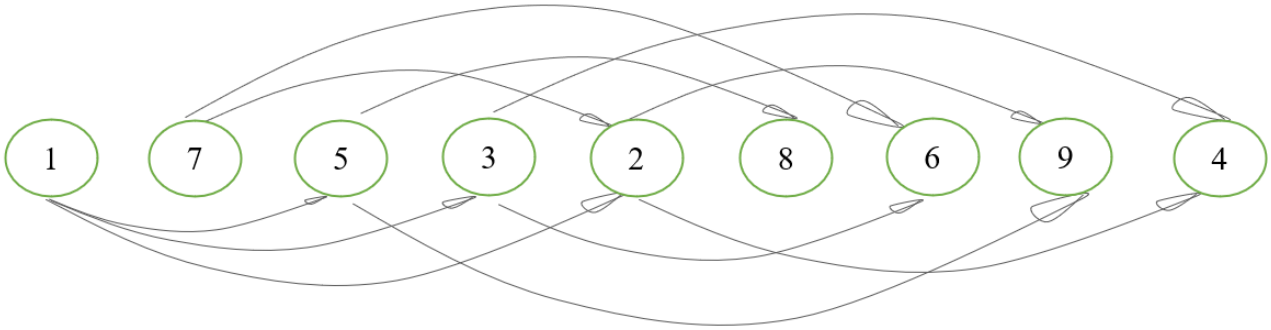


Рис. 6

На кроці 6 алгоритму Кана вибір вершини v, яка на поточний момент не має вхідних ребер, не є однозначним. Можливі варіанти правил вибору вершин з множини S: FIFO, LIFO, RANDOM, за кількістю підпорядкованих (дочірніх) вершин тощо. В залежності від того, яке правило використовується може бути отримана інша топологічна послідовність L. При цьому в послідовності L можуть бути переставлені будь-які дві вершини, що стоять поруч і між якими не встановлено відношення часткового порядку.

Графічна ілюстрація топологічного сортування, отриманого для графу, зображеного на рис. 5, представлена на рисунку 7 (вершини графу, що стоять поруч і між якими не встановлено відношення часткового порядку, позначено одним кольором).

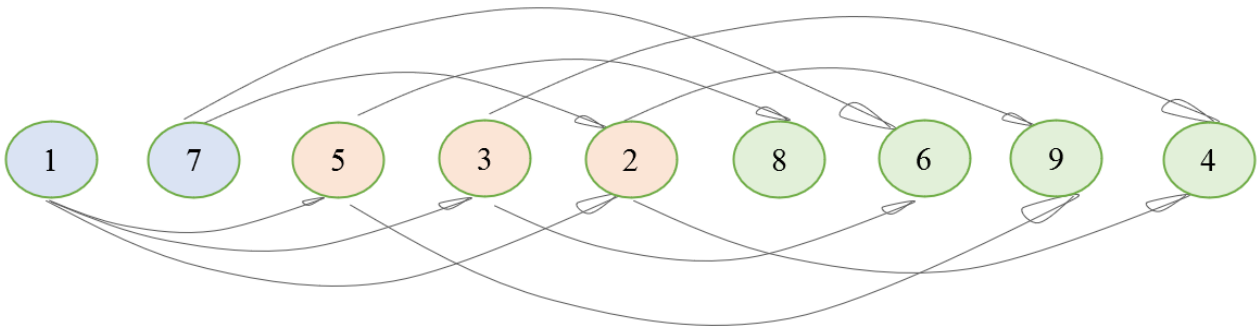


Рис. 7

Отже, в загальному випадку для графа може існувати декілька топологічних сортувань. Зокрема, для графу, що розглядається, існує 2!3!4!=144 топологічних сортування.

На рисунку 8 наведено ще один з можливих варіантів топологічного сортування вершин даного графу, який відрізняється від упорядкування з рис. 7.

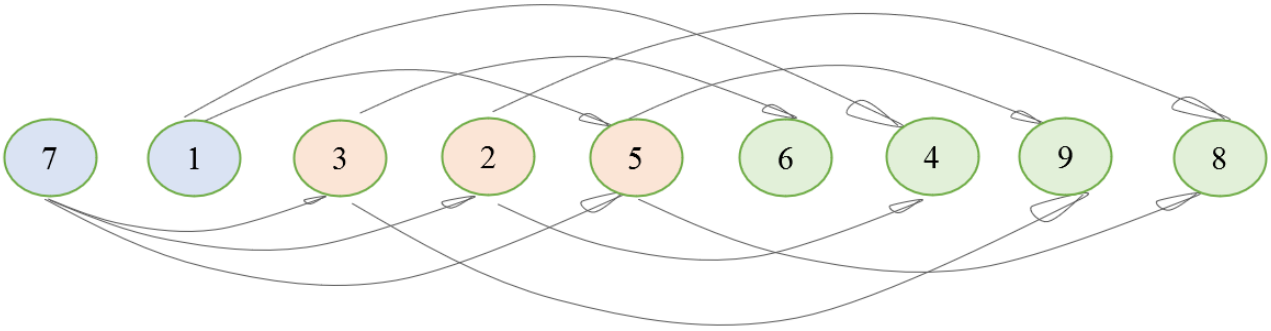


Рис. 8