125-Кібербезпека. Змістовий модуль 5. ГРАФИ І ДЕРЕВА

Тема 18. Дерева та їхні застосування

План лекції

- **Дерева. Основні означення та властивості**
- **Рекурсія. Обхід дерев. Польська нотація**

Дерева. Основні означення та властивості

Поняття дерева широко використовують у багатьох розділах математики й інформатики. Наприклад, дерева використовують як інструмент під час обчислень, як зручний спосіб збереження даних, їх сортування чи пошуку.

Деревом називають зв'язний граф без простих циклів. Граф, який не містить простих циклів і складається з k компонент, називають *лісом* із k дерев.

Приклад. На рис. 1 зображено приклади дерев. Граф, який зображено на рис. 2 – не дерево, бо він незв'язний.

Зауваження. З означення випливає, що дерева й ліси є простими графами.

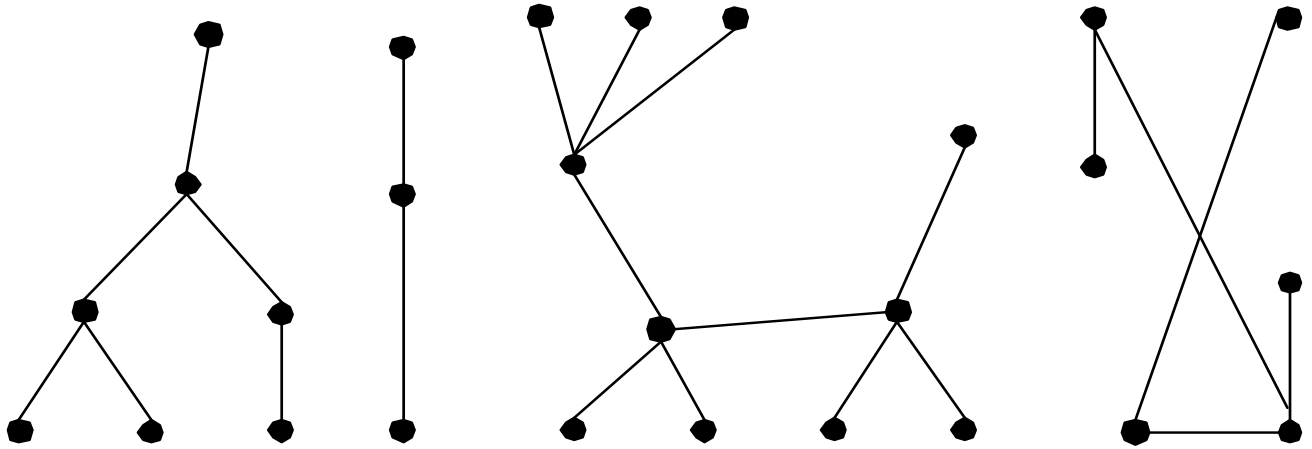

Рис. 1

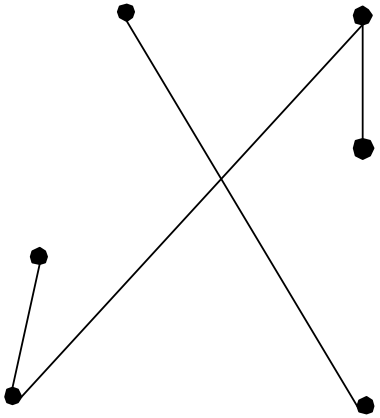

Рис. 2

Теорема. Нехай граф T має n вершин. Тоді такі твердження еквівалентні:

(I) граф T – дерево;

(II) граф T не містить простих циклів і має $(n–1)$ ребро;

(III) граф T зв'язний і має $(n–1)$ ребро;

(IV) граф T зв'язний, але вилучення довільного ребра робить його незв'язним;

(V) довільні дві вершини графа T з'єднані точно одним простим шляхом;

(VI) граф T не містить простих циклів, але, додавши до нього довільне нове ребро (без додавання вершин), ми отримаємо точно один простий цикл.

Доведення (математичною індукцією). У разі $n=1$ твердження тривіальні; припустимо, що $n\geq 2$.

(I)→(II). За означенням T не містить простих циклів. Отже, вилучивши довільне ребро, ми одержимо два графи, кожний з яких являє собою дерево з меншою, ніж у T, кількістю вершин. За припущенням індукції кількість ребер у кожному з отриманих дерев на 1 менша за кількість вершин. Звідси випливає, що граф T має $(n–1)$ ребро.

(II)→(III). Припустимо, що граф T незв'язний. Тоді кожна його компонента являє собою зв'язний граф без простих циклів, тобто дерево. Звідси випливає, що кількість вершин у кожній компоненті на одиницю більша від кількості ребер. Отже, загальна кількість вершин графа T більша за кількість ребер принаймні на 2. Але це суперечить тому, що граф T має $(n–1)$ ребро.

(III)→(IV). Вилучимо довільне ребро, отримаємо граф з n вершинами та $(n–2)$ ребрами. Припущення про зв'язність такого графа суперечить теоремі про оцінку (знизу) кількості ребер звичайного графа.

(IV)→(V). Оскільки граф T зв'язний, то кожну пару його вершин з'єднано принаймні одним простим шляхом (теорема 3.4). Якщо припустити, що якусь пару вершин з'єднано двома простими шляхами, то вони замикаються в простий цикл. Але це суперечить тому, що вилучення довільного ребра робить граф T незв'язним.

(V)→(VI). Припустимо, що граф T містить простий цикл. Тоді довільні дві вершини цього простого циклу з'єднано принаймні двома простими шляхами, що суперечить твердженню (V). Додавши тепер до графа T ребро e, що з'єднує довільні дві його вершини, одержимо єдиний простий цикл, бо інцидентні ребру e вершини вже з'єднано в графі T точно одним простим шляхом.

(VI)→(I). Припустимо, що граф T незв'язний. Тоді додавання будь-якого ребра, що з'єднує вершину однієї компоненти з вершиною іншої, не зумовлює утворення простого циклу, що суперечить твердженню (VI).

Наслідок. Ліс із k дерев, який містить n вершин, має $(n–k)$ ребер.

Розглянемо кореневе дерево. У багатьох застосуваннях певну вершину дерева означають як *корінь*. Тоді можна природно приписати напрямок кожному ребру. Оскільки існує єдиний простий шлях від кореня до кожної вершини графа, то можна орієнтувати кожне ребро в напрямку від кореня. Отже, дерево разом із виділеним коренем утворює орієнтований граф, який називають *кореневим деревом*.

Зазначимо, що різні способи вибору кореня утворюють різні кореневі дерева.

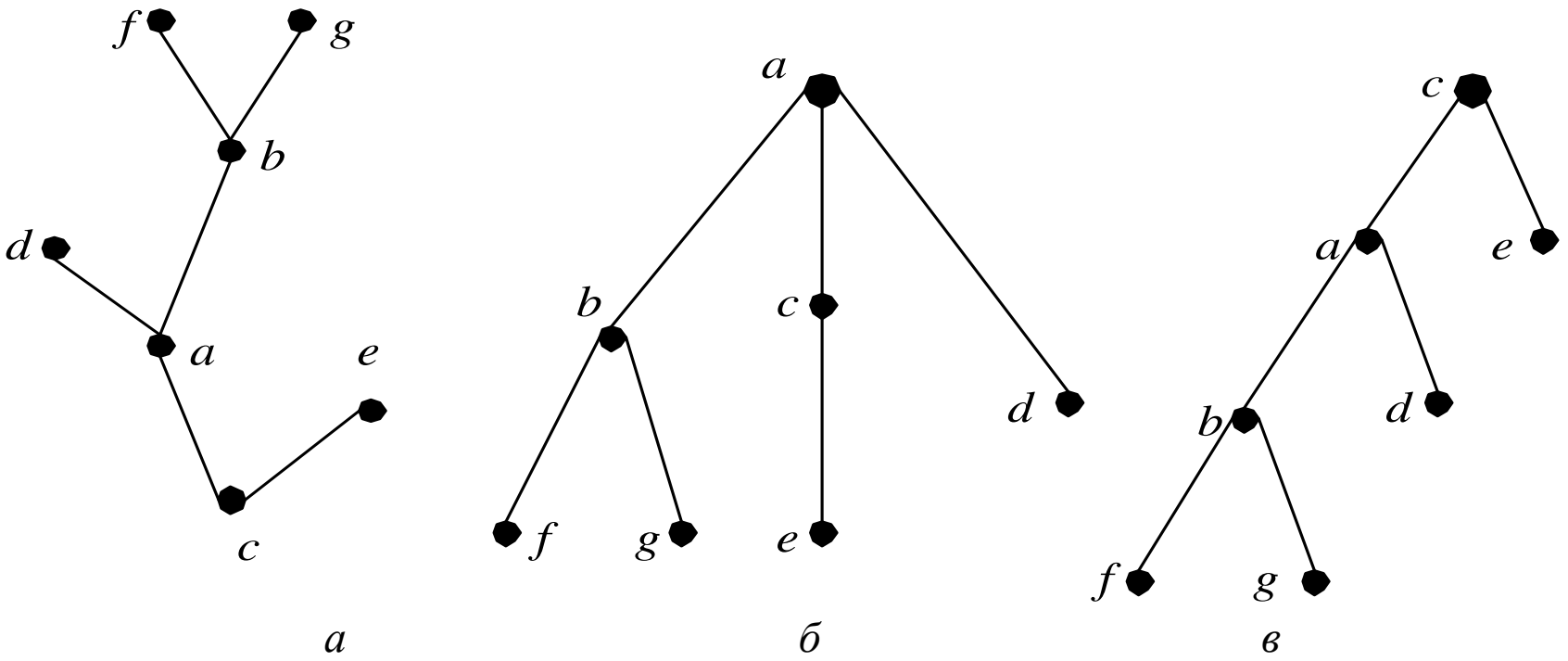

Рис. 3

Приклад. На рис. 3, *а* зображено дерево, а на рис. 3, *б*, *в* – кореневі дерева з коренями відповідно у вершинах *a* та *c*.

Нехай T – кореневе дерево. Якщо v – його вершина, відмінна від кореня, то *батько* v – це єдина вершина u така, що є орієнтоване ребро (u, v). Якщо u – батько, то v – *син*. Аналогічно за генеалогічною термінологією можна означити інших предків і нащадків вершини v. Вершини дерева, які не мають синів, називають *листками*. Вершини, які мають синів, називають *внутрішніми*. Зазначимо, що корінь належить до внутрішніх вершин.

Якщо a – вершина дерева, то *піддерево* з коренем a – це підграф, що містить a та всі вершини – нащадки вершини a, а також інцидентні їм ребра.

Кореневе дерево називають *m-арним*, якщо кожна його внутрішня вершина має не більше ніж m синів. Дерево називають *повним m-арним*, якщо кожна його внутрішня вершина має точно m синів. У разі $m = 2$ дерево називають *бінарним*.

Упорядковане кореневе дерево – це кореневе дерево, у якому сини кожної внутрішньої вершини впорядковано. На рисунку таке дерево зображають так, щоб сини кожної вершини були розміщені зліва направо.

Якщо внутрішня вершина впорядкованого бінарного дерева має двох синів, то першого називають *лівим* сином, а другого – *правим*. Піддерево з коренем у вершині, яка являє собою лівого сина вершини v, називають *лівим піддеревом у вершині* v. Якщо корінь піддерева – правий син вершини v, то таке піддерево називають *правим піддеревом у вершині* v.

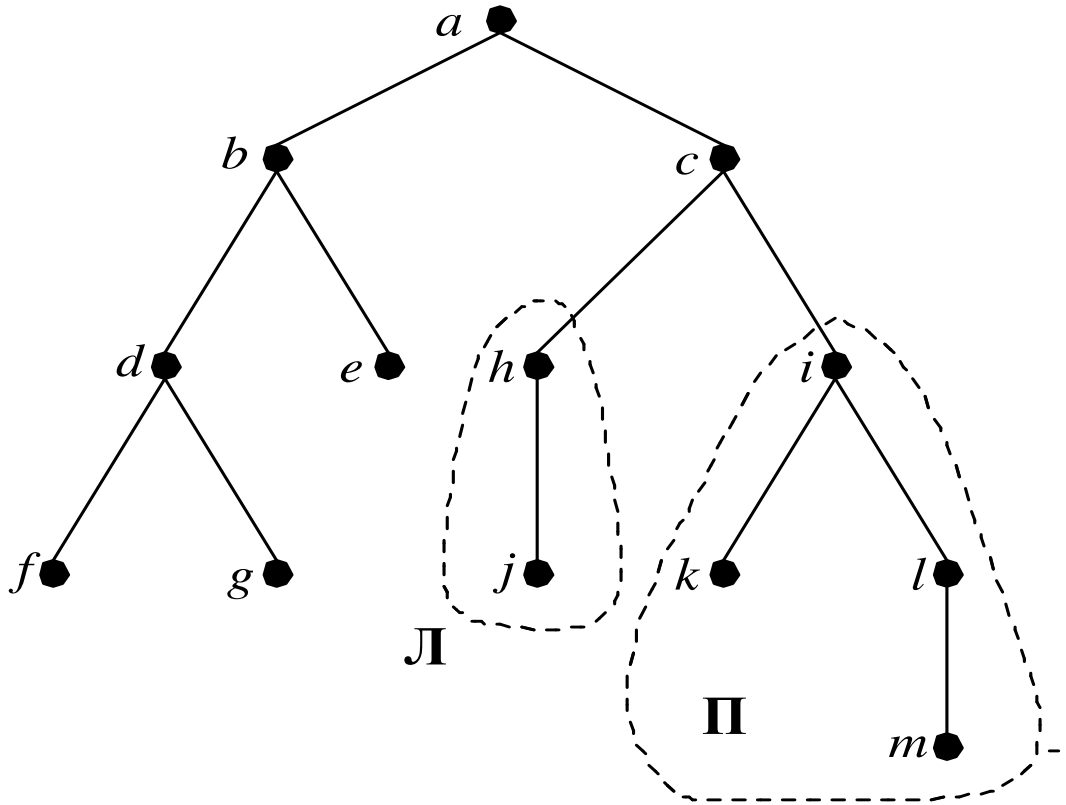

Рис. 4

Приклад. У дереві, зображеному на рис. 4, Л і П – відповідно ліве та праве піддерева у вершині *c*.

Теорема. Повне *m*-арне дерево з *i* внутрішніми вершинами містить $n = m \cdot i + 1$ вершин.

Доведення. Кожна вершина, окрім кореня, – син внутрішньої вершини. Оскільки кожна з i внутрішніх вершин має m синів, то всього є, якщо не враховувати корінь, $m \cdot i$ вершин. З урахуванням кореня всього вершин $n = m \cdot i + 1$, що й потрібно було довести.

Рівнем вершини *v* в кореневому дереві називають довжину простого шляху від кореня до цієї вершини (цей шлях, очевидно, єдиний). Рівень кореня вважають нульовим. *Висотою* кореневого дерева називають максимальний із рівнів його вершини. Інакше кажучи, висота кореневого дерева – це довжина найдовшого простого шляху від кореня до будь-якої вершини.

Повне *m*-арне дерево, у якого всі листки на одному рівні, називають *завершеним m-арним деревом*.

Кореневе *m*-арне дерево з висотою *h* називають *збалансованим*, якщо всі його листки на рівнях *h* або *h*–1.

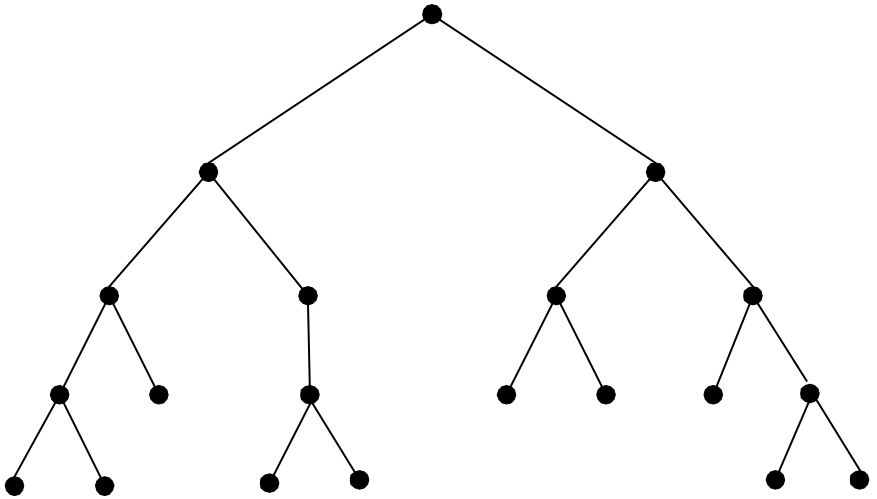

Рис. 5

Приклад. На рис. 5 зображено збалансоване бінарне дерево, яке має висоту 4: усі його листки на рівнях 3 та 4.

Зазначимо, що є й інші означення збалансованості.

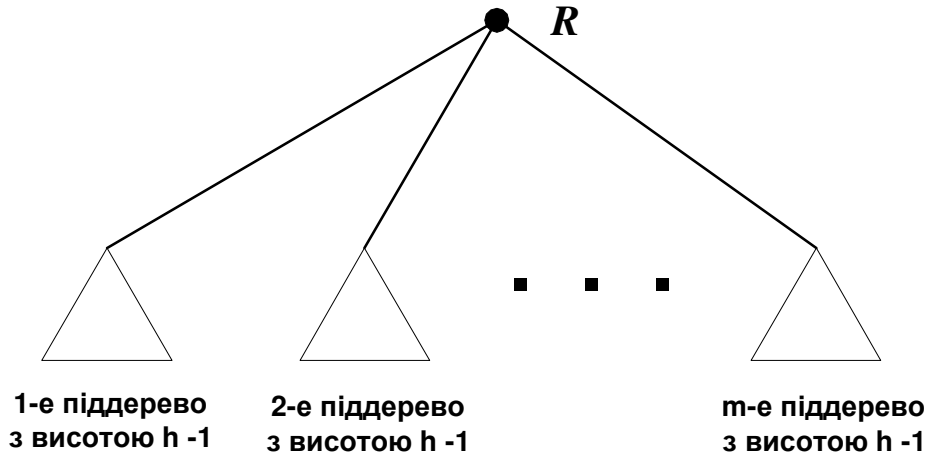

Рис. 6

Теорема. Нехай m-арне дерево має висоту h. Тоді в ньому не більше ніж m^h листків.

Доведення. Застосуємо математичну індукцію за h. У разі $h = 1$ твердження очевидне. Припустимо, що воно справджується для всіх m-арних дерев із меншою висотою, ніж h.

Нехай T – m-арне дерево з висотою h. Листки дерева T – це листки піддерев, які отримують з T вилученням ребер, що з'єднують корінь дерева T із кожною вершиною рівня 1 (рис. 6). Кожне з цих піддерев має не більшу висоту, ніж h–1. За індуктивною гіпотезою кожне з них має не більше ніж m^{h-1} листків. Позаяк таких піддерев не більше ніж m, то загальна кількість листків у дереві T не перевищує $m \cdot m^{h-1} = m^h$. Теорему доведено.

Наслідок. Якщо m-арне дерево з висотою h має l листків, то $h \geq \lceil \log_m l \rceil$. Якщо m-арне дерево повне та збалансоване, то $h = \lceil \log_m l \rceil$. Нагадаємо, що $\lceil x \rceil$ – це найменше ціле, яке більше чи дорівнює x.

Доведення. За теоремою $l \leq m^h$. Логарифмуємо цю нерівність за основою m, тоді $\log_m l \leq h$. Оскільки h – ціле, то $h \geq \lceil \log_m l \rceil$. Тепер припустимо, що дерево повне й збалансоване. Вилучимо всі листки на рівні h (разом з інцидентними їм ребрами). Одержимо завершене m-арне дерево з висотою h–1, це дерево має m^{h-1} листків (див. задачу 12, стор. 226 підручника).

Отже, $m^{h-1} < l \leq m^h$. Звідси $h - 1 < \log_m l \leq h$, тобто $h = \lceil \log_m l \rceil$.

Рекурсія. Обхід дерев. Польська нотація

Об’єкт називають *рекурсивним*, якщо він містить сам себе, або означений за допомогою самого себе. Рекурсія – потужний засіб у математичних означеннях.

Приклад. Раніше ми означили повне бінарне дерево. Тепер означимо його рекурсивно:
а)°(ізольована вершина) – *повне бінарне дерево*;
б) якщо *A* та *B* – повні бінарні дерева, то конструкція, зображена на рис 7, – *повне бінарне дерево*.

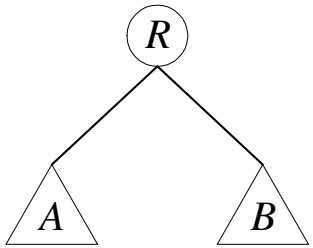

Рис. 7

Приклад. Рекурсивне означення функції $n!$ для невід’ємних цілих чисел має такий вигляд:
а) $0! = 1$;
б) якщо $n > 0$, то $n! = n(n–1)!$

Очевидно, що потужність рекурсії пов’язана з тим, що вона дає змогу означити нескінченну множину об’єктів за допомогою скінченного висловлювання. Так само нескінченні обчислення можна описати за допомогою скінченної рекурсивної програми, навіть якщо ця програма не містить явних циклів. Проте, краще за все використовувати рекурсивні алгоритми в тих випадках, коли задача, яку розв’язують, або функція, яку обчислюють, або дані, які обробляють, задано за допомогою рекурсії.

Чимало задач можна моделювати з використанням кореневих дерев. Поширене таке загальне формулювання задачі: виконати задану операцію **обробити** з кожною вершиною дерева. Тут **обробити** – параметр загальнішої задачі відвідування всіх вершин, або так званого *обходу дерева*. Розглядаючи розв'язування цієї задачі як єдиний послідовний процес відвідування вершини дерева в певному порядку, можна вважати їх розміщеними одна за одною. Опис багатьох алгоритмів істотно спрощується, якщо можна говорити про наступну вершину дерева, маючи на увазі якесь упорядкування. Є три принципи впорядкування вершин, які природно випливають зі структури дерева. Як і саму деревоподібну структуру, їх зручно формулювати за допомогою рекурсії.

Звертаючись до бінарного дерева, де R – корінь, A та B – ліве та праве піддерева (рис. 7), можна означити такі впорядкування.

1. Обхід *у прямому порядку* (*preorder*) або *зверху вниз*: R, A, B (корінь відвідують до обходу піддерев).

2. Обхід *у внутрішньому порядку* (*inorder*) або *зліва направо*: A, R, B.

3. Обхід *у зворотному порядку* (*postorder*) або *знизу вверх* A, B, R (тобто корінь відвідують після обходу піддерев).

Нижче наведено рекурсивні алгоритми обходу бінарних дерев. У кожному способі обходу використано команду **обробити**(v), де v – вершина. Цей термін залишено невизначеним, бо його значення залежить від того, що ми хочемо зробити під час проходження вершин. Для наших цілей достатньо використати звичайну команду друку символу.

Алгоритм обходу в прямому порядку позначено як **ОПП**(x), у внутрішньому – як **ОВП**(x) і в зворотному – як **ОЗП**(x). У всіх трьох алгоритмах x – це корінь дерева, яке обходять. Лівого сина вершини v позначено як лс(v), а правого – як пс(v).

Алгоритм обходу дерева в прямому порядку – **ОПП**(*корінь*)

- **Обробити**(*корінь*)
- Якщо лс(*корінь*) існує, то **ОПП**(лс(*корінь*))
- Якщо пс(*корінь*) існує, то **ОПП**(пс(*корінь*))

Алгоритм обходу дерева у внутрішньому порядку – **ОВП**(*корінь*)

- Якщо лс(*корінь*) існує, то **ОВП**(лс(*корінь*))
- **Обробити**(*корінь*)
- Якщо пс(*корінь*) існує, то **ОВП**(пс(*корінь*)

Алгоритм обходу дерева у зворотному порядку – **ОЗП**(*корінь*)

- Якщо лс(*корінь*) існує, то **ОЗП**(лс(*корінь*))
- Якщо пс(*корінь*) існує, то **ОЗП**(пс(*корінь*))
- **Обробити**(*корінь*)

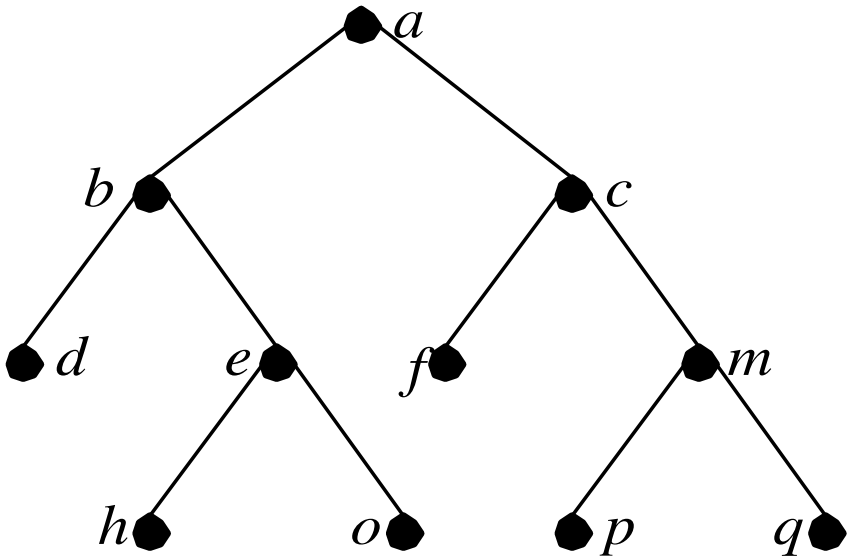

Рис. 8

Приклад. На рис. 4.8 зображено бінарне дерево. Різні обходи дадуть такі послідовності вершин:

- обхід у прямому порядку: *a b d e h o c f m p q*;
- обхід у внутрішньому порядку: *d b h e o a f c p m q*;
- обхід у зворотному порядку: *d h o e b f p q m c a.*

Зазначені способи обходу бінарних дерев можна узагальнити й на довільні *m*-арні дерева.

Простий метод обходу дерев (A Shortcut for Tree Traversal)

Обведемо впорядковане кореневе дерево кривою лінією вздовж ребер, почавши з кореня (рис. 9).

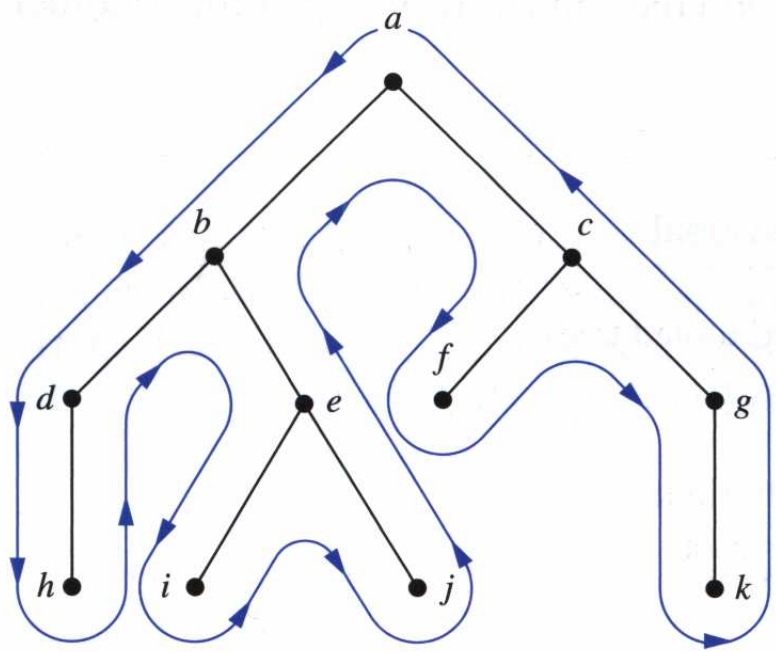

Рис. 9. Простий метод обходу впорядкованого кореневого дерева в прямому, внутрішньому й зворотному порядках

1. Ми отримаємо послідовність вершин *у прямому порядку обходу*, якщо будемо записувати вершину, коли крива *перший раз* проходить її:

$$a,\ b,\ d,\ h,\ e,\ i,\ j,\ c,\ f,\ g,\ k$$

2. Ми отримаємо послідовність вершин *у внутрішньому порядку обходу*, якщо будемо записувати листок, коли крива *перший раз* проходить його, а внутрішню вершину, – коли крива *другий раз* проходить її:

$$h,\ d,\ b,\ i,\ e,\ j,\ a,\ f,\ c,\ k,\ g$$

3. Ми отримаємо послідовність вершин *у зворотному порядку обходу*, якщо будемо записувати вершину, коли крива *останній раз* проходить її на шляху до її батька:

$$h,\ d,\ i,\ j,\ e,\ b,\ f,\ k,\ g,\ c,\ a$$

Одне важливе застосування обходу дерев

Надзвичайно поширене застосування в інформатиці обходу дерев – зіставлення виразам (арифметичним, логічним тощо) дерев і побудова на цій основі різних форм *нотації* виразів (нотація означає запис). Суть справи зручно пояснити на прикладі. Розглянемо арифметичний вираз (зірочкою позначено операцію множення)

$$\left(a+\frac{b}{c}\right)*(d-e*f).$$

Подамо його у вигляді дерева. Послідовність дій відтворено на рис. 10. Рамкою на ньому обведено дерево, яке відповідає заданому арифметичному виразу. Внутрішнім вершинам цього дерева відповідають символи операцій, а листкам – операнди.

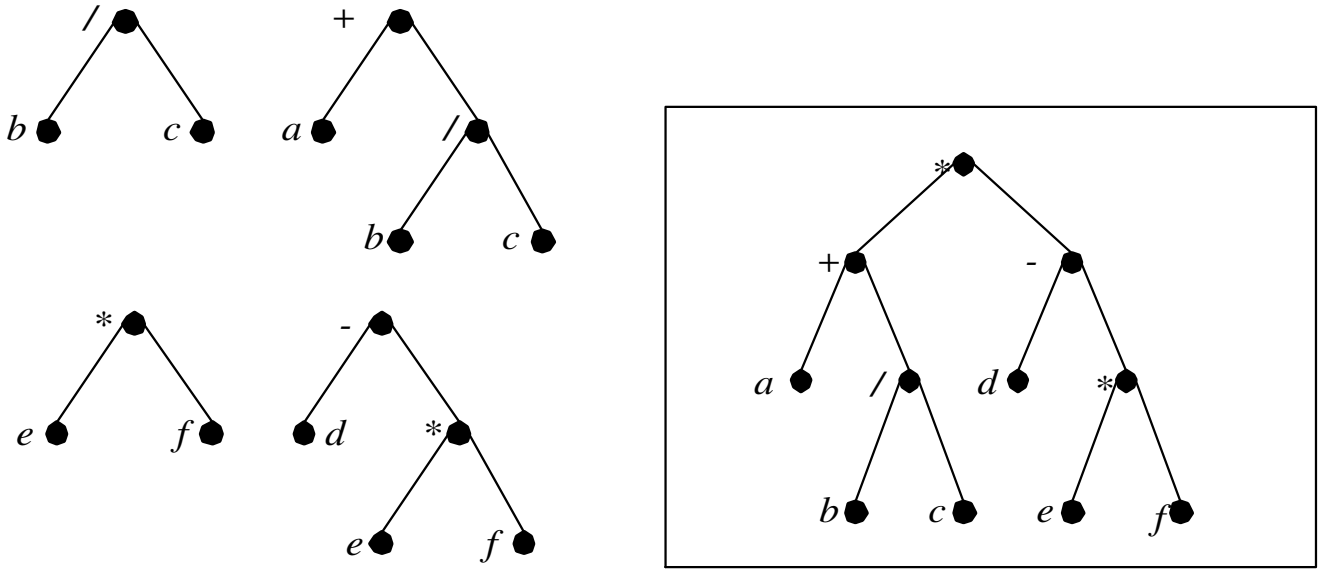

Рис. 10. Подання виразу $\left(a+\frac{b}{c}\right)*(d-e*f)$ у вигляді дерева

Обійдемо це дерево, записуючи символи у вершинах у тому порядку, у якому вони зустрічаються в разі заданого способу обходу. Тоді отримаємо такі три послідовності:

- у разі обходу в прямому порядку – *польська* (або *префіксна*) *нотація*:

$$* \; + \; a \; / \; b \; c \; - \; d \; * \; e \; f .$$

- у разі обходу у внутрішньому порядку – *інфіксна нотація* (поки що без дужок, потрібних для визначення порядку операцій):

$$a \; + \; b \; / \; c \; * \; d \; - \; e \; * \; f .$$

- у разі обходу в зворотному порядку – *зворотна польська* (або *постфіксна*) *нотація*:

$$a \; b \; c \; / + d \; e \; f \; * \; - \; * .$$

Звернімося спочатку до інфіксної нотації виразу. Без дужок вона неоднозначна: одна нотація може відповідати різним деревам. Наприклад, дереву, зображеному на рис. 11, у разі обходу зліва направо відповідає той самий вираз $a + b / c * d – e * f$, що й дереву на рис. 10 (у рамці), хоча на цих рисунках зображено різні дерева. Щоб уникнути неоднозначності інфіксної нотації, використовують круглі дужки щоразу, коли зустрічають операцію. Повністю «одужкований» вираз, отриманий під час обходу дерева у внутрішньому порядку, називають *інфіксною нотацією*. Отже, для дерева з рис. 10 інфіксна нотація така: $((a+(b/c))*(d–(e*f)))$; для дерева, зображеного на рис. 11, інфіксна нотація має такий вигляд: $(a+(((b/(c*d))–e)*f))$.

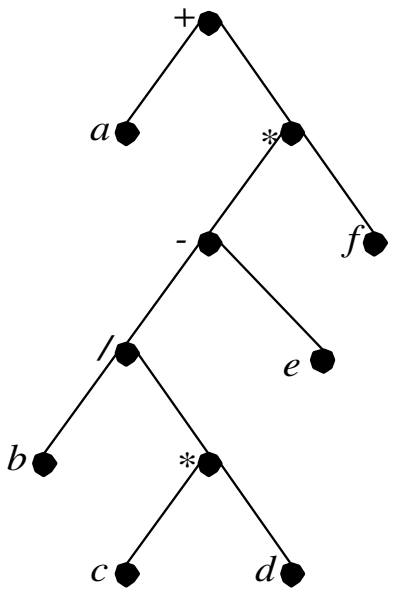

Рис. 11

Наведені міркування свідчать, що інфіксна нотація виразів незручна. На практиці використовують польську (або префіксну) та зворотну польську (тобто постфіксну) нотації, бо вони однозначно відповідають виразу й не потребують дужок. Ці форми запису називають *польськими нотаціями* (на честь польського математика й логіка Яна Лукасевича, народився у Львові).

Приклад. Розглянемо логічний вираз $(\neg(p\wedge q))\sim(\neg p\vee\neg q)$. Послідовні етапи побудови відповідного бінарного дерева зображено на рис. 12.

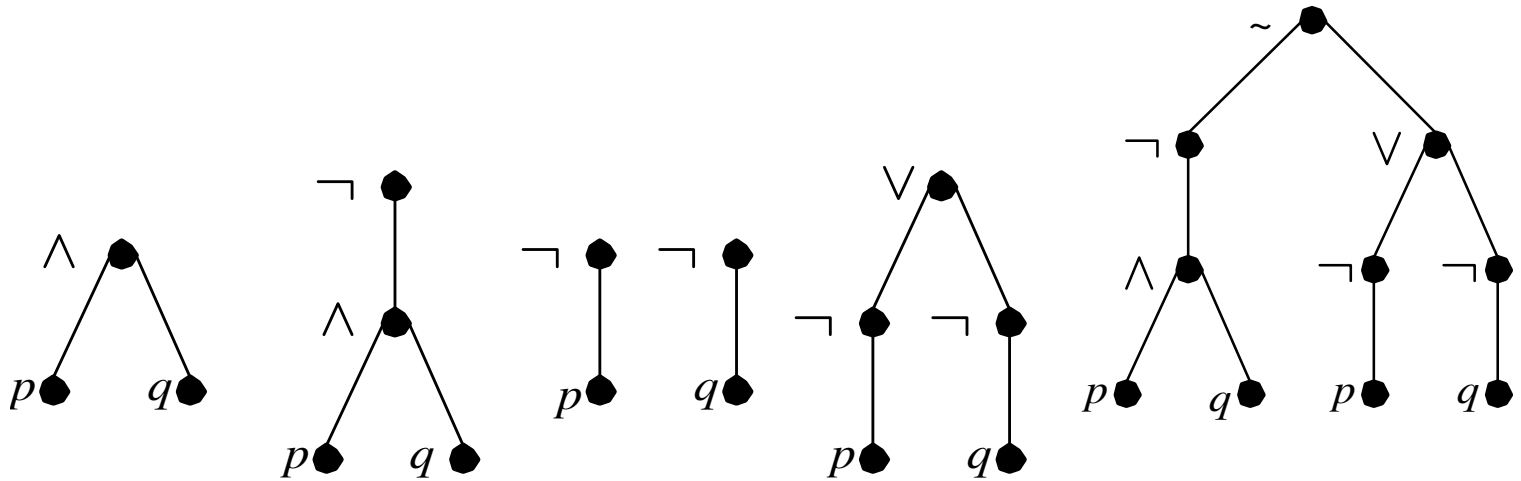

Рис. 12

Отримаємо такі форми запису виразу:

♦ польська нотація (відповідає обходу дерева виразу в прямому порядку)

$$\sim\neg\wedge pq\vee\neg p\neg q;$$

♦ зворотна польська нотація (відповідає обходу дерева виразу в зворотному порядку)

$$pq\wedge\neg p\neg q\neg\vee\sim.$$

Обчислення значення виразу в польській і зворотній польській нотаціях

Для обчислення значення виразу в польській нотації її проглядають **справа наліво** та знаходять два операнди разом зі знаком операції **перед** ними. Ці операнди та знак операції вилучають із нотації, виконують операцію, а її результат записують на місце вилучених символів.

Приклад. Обчислимо значення виразу в польській нотації (стрілка означає піднесення до степеня)

$$+ - * 2\ 3\ 5\ /\ \uparrow 2\ 3\ 4.$$

За сформульованим правилом виділимо $\uparrow 2\ 3$, ці символи вилучимо й обчислимо $2 \uparrow 3 = 8$, результат записуємо на місце вилучених символів:

$$+ - * 2\ 3\ 5\ /\ 8\ 4.$$

Продовжимо обчислення. Динаміку процесу відображено в таблиці.

Таблиця обчислення значення виразу в польській нотації

Крок	Вираз	Виділені символи	Виконання операції
1	$+ - * 2\ 3\ 5\ /\ \uparrow 2\ 3\ 4$	$\uparrow 2\ 3$	$2 \uparrow 3 = 8$
2	$+ - * 2\ 3\ 5\ /\ 8\ 4$	$/\ 8\ 4$	$8 / 4 = 2$
3	$+ - *\ 2\ 3\ 5\ 2$	$*\ 2\ 3$	$2 * 3 = 6$
4	$+ -\ 6\ 5\ 2$	$-\ 6\ 5$	$6 - 5 = 1$
5	$+\ 1\ 2$	$+\ 1\ 2$	$1 + 2 = 3$
6	**3**		

Для обчислення значення виразу в зворотній польській нотації її проглядають **зліва направо** та виділяють два операнди разом зі знаком операції **після** них. Ці операнди та знак операції вилучають із нотації, виконують операцію, а її результат записують на місце вилучених символів.

Приклад. Обчислимо значення виразу в зворотній польській нотації

$$7\ 2\ 3\ *\ -\ 4\ \uparrow\ 9\ 3\ /\ +.$$

Динаміку обчислень відображено в таблиці.

Таблиця обчислення значення виразу в зворотній польській нотації

Крок	Вираз	Виділені символи	Виконання операції
1	7 2 3 * – 4 ↑ 9 3 / +	2 3 *	2 * 3 = 6
2	7 6 – 4 ↑ 9 3 / +	7 6 –	7 – 6 = 1
3	1 4 ↑ 9 3 / +	1 4 ↑	1 ↑ 4 = 1
4	1 9 3 / +	9 3 /	9 / 3 = 3
5	1 3 +	1 3 +	1 + 3 = 4
6	**4**		

Оскільки польські нотації однозначні та їх значення можна обчислити без сканування назад і вперед, їх широко використовують у комп'ютерних науках, особливо для конструювання компіляторів.

Ян Лукасевич (пол. *Jan Łukasiewicz*)

(іноді Лукашевич)

Народився 21 грудня 1878 р., Львів

Помер 13 лютого 1956 р., Дублін

Alma mater Львівський університет

Галузь наукових інтересів Логіка, філософія

Доктор філософії

Член Польської академії наук.

Відомий завдяки: Польська нотація, логіка Лукасевича.