Обхід дерев. Префіксна та постфіксна форми запису виразів. Бінарне дерево пошуку.

Обхід дерев.

Розглядаючи розв'язування задачі обходу дерева, як єдиний послідовний процес відвідування вершин 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.

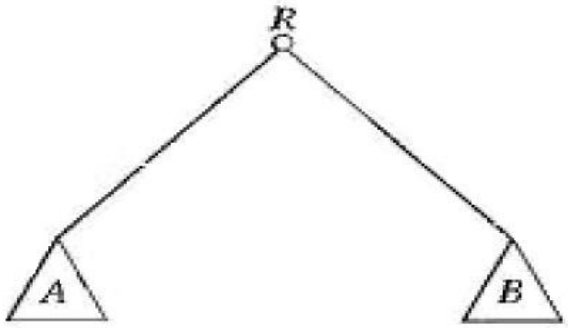

Рис.1

Звертаючись до бінарного дерева, де *R* корінь, *A* та *B* – ліве та праве піддерева (рис.1), можна означити такі впорядкування.

- Обхід *у прямому порядку (preorder)*, або *зверху вниз*. *R, A, B* (корінь відвідують до обходу піддерев).
- Обхід *у внутрішньому порядку (inorder)*, або *зліва направо*: *A, R, B.*
- Обхід *у зворотному порядку (postorder)*, або *знизу вверх*: *A, B, R* (корінь відвідують після обходу піддерев).

На рис.2 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. Обхід таких дерев у прямому порядку (зверху вниз) схематично зображено на рис. 3, у внутрішньому порядку (зліва направо) — на рис. 4, у зворотному (знизу вверх) - на рис 5.

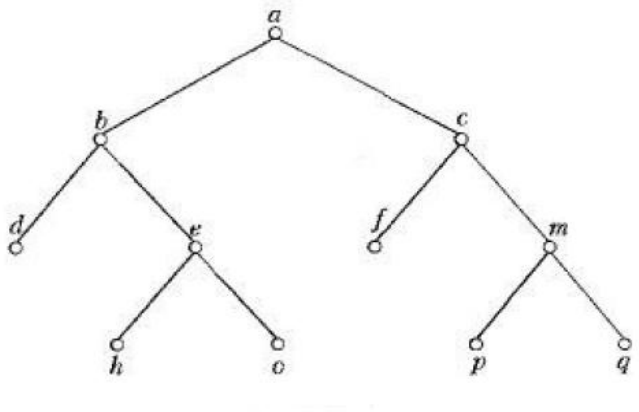

Рис. 2

Крок 1: відвідати R

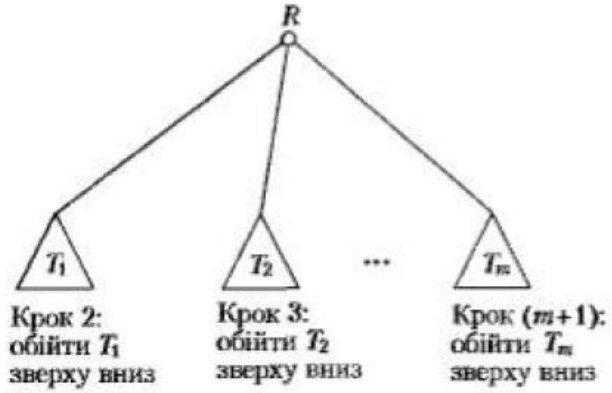

Рис. 3

Крок 2: відвідати R

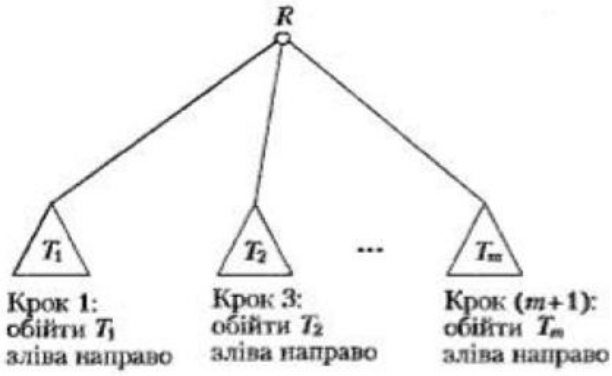

Рис. 4

Крок (m + 1) : відвідати R

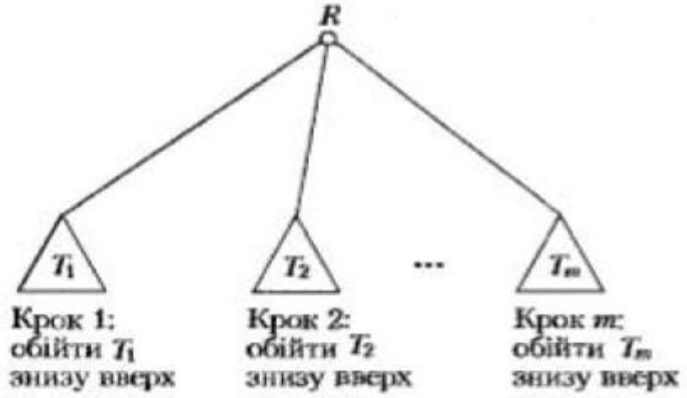

Рис. 5

Префіксна та постфіксна форми запису виразів.

Надзвичайно поширене в інформатиці застосування обходу дерев – зіставлення виразам (арифметичним, логічним тощо) дерев і побудова на цій основі різних форм запису виразів. Суть справи зручно пояснити на прикладі Розглянемо арифметичний вираз

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

Подамо його у вигляді дерева. Послідовність дій відтворено на рис. 6. Рамкою на ньому обведено дерево, яке відповідає заданому арифметичному виразу. Внутрішнім вершинам цього дерева відповідають символи операцій, а листкам операнди.

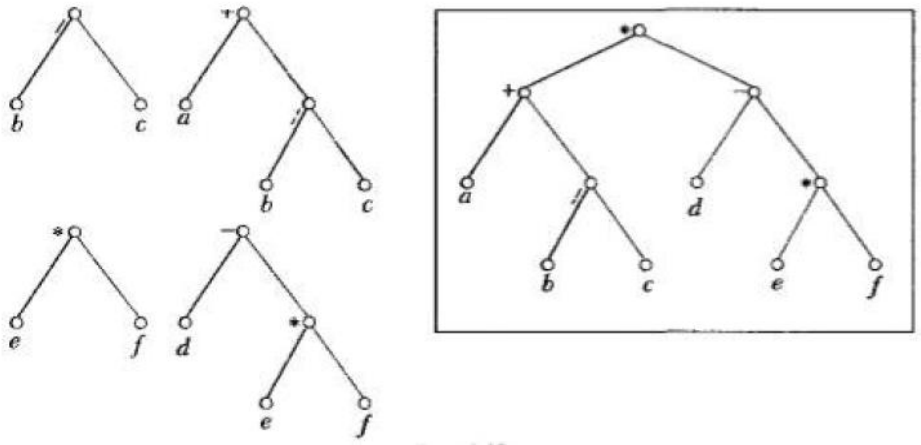

Рис. 6

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

- у разі обходу в прямому порядку — *префіксний (польський)* запис

$$* + a / bc - d * ef$$

- у разі обходу у внутрішньому порядку — *інфіксний запис* (поки що без дужок, потрібних для визначення порядку операцій)

$$a + b/c * d - e * f$$

- у разі обходу в зворотному порядку — *постфіксний (зворотний польський) запис*

$$abc/+def * - *$$

Звернімося спочатку до інфіксної форми запису виразу. Вона неоднозначна: один запис може відповідати різним деревам. Наприклад, дереву, зображеному на рис. 7, у разі обходу зліва направо відповідає той самий вираз $a + b/c * d - e * f$, що й дереву на рис. 6 (у рамці), хоча на цих рисунках зображено різні дерева. Щоб уникнути неоднозначності інфіксної форми, використовують круглі дужки щоразу, коли зустрічають операцію Повністю "одушкований" вираз, одержаний під час обходу дерева у внутрішньому порядку, називають інфіксною формою запису. Отже, для дерева з рис. 6 інфіксна форма така: $((a + (b/c)) * (d - (e * f)))$; для дерева, зображеного на рис. 7. Інфіксна форма має такий вигляд:

$(a + (((b/(c * d)) - e) * f))$.

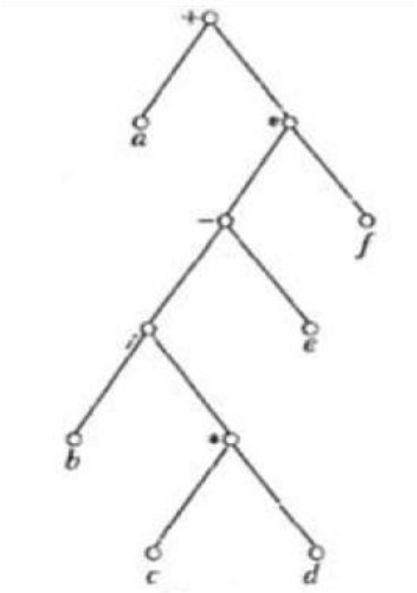

Рис. 7

Наведені міркування свідчать, що інфіксна форма запису виразів незручна. На практиці використовують префіксну та постфіксну форми, бо вони однозначно відповідають виразу й не потребують дужок. Ці форми запису називають *польським записом* (на честь польського математика й логіка Яна Лукасевича, українця за походженням).

Бінарне дерево пошуку.

Під час побудови бінарного дерева пошуку використовують його рекурсивну властивість, яку можна описати так. Кожна вершина розбиває дерево на два піддерева. Ліве піддерево містить лише ключі, менші від ключа цієї вершини, а праве — ключі, більші від ключа вершини. Властивість бінарного дерева пошуку дозволяє вивести всі ключі, які знаходяться у дереві, у відсортованому порядку за допомогою простого рекурсивного алгоритму, який називається обходом дерева у внутрішньому порядку (inorder). Цей алгоритм обходить вершини дерева таким чином, що корінь піддерева обходиться після свого лівого

піддерева та перед правим піддеревом. Є й інші способи обходу дерева: обхід у прямому порядку (preorder), коли спочатку обходиться корінь, а потім його нащадки зліва направо, та у зворотному порядку (postorder), коли спочатку обходяться нащадки зліва направо, а потім корінь піддерева.

Розглянемо множину *S = {e1, e2, ..., en},* яка містить n різних елементів таких що *e1 < e2 < ... < en.* Розглянемо бінарне дерево пошуку, що складається з елементів *S*. Чим частіше відбувається запит до елементу, тим ближче він має розташовуватися до кореня.

Вартістю $cost$ доступу до елементу e_i з *S* у дереві будемо називати значення $cost(e_i)$, яке дорівнює кількості ребер на шляху, що сполучає корінь з вершиною, яка містить елемент. Маючи частоту запитів до елементів із *S, (f(e1), f(e2), ..., f(en)),* визначимо загальну вартість дерева наступним чином:

$$f(e1)\cdot cost(e1) + f(e2)\cdot cost(e2) + ... + f(en)\cdot cost(en)$$

Дерево, яке має найменшу вартість, вважається найкращим для пошуку елементів із S. Тому воно називається Оптимальним Бінарним Деревом Пошуку.

Створення Бінарного Дерева Пошуку

Бінарним Деревом Пошуку (БДП) називається бінарне дерево з коренем, що має наступні властивості:

- Ліве піддерево містить лише вершини, значення яких строго менші за корінь.
- Праве піддерево містить лише вершини, значення яких строго більші за корінь.
- Усі значення вершин різні.
- Ліве та праве піддерево рекурсивно є бінарним деревом пошуку.

При вставці нової вершини запускається наступний алгоритм:

1. Якщо дерево порожнє, то нова вершина стає коренем і процес вставки завершується. Інакше перейти до кроку 2.
2. Оголосимо поточною вершиною корінь дерева.
3. Якщо значення нової вершини менше кореня, то:
4. Якщо ліве піддерево кореня порожнє, то встановимо нову вершину лівим сином кореня і зупиняємося.
5. Інакше встановимо поточною вершиною корінь лівого піддерева і повторимо крок 3.
6. Якщо значення нової вершини більше кореня, то:
7. Якщо праве піддерево кореня порожнє, то встановимо нову вершину правим сином кореня і зупиняємся.
8. Інакше встановимо поточною вершиною корінь правого піддерева і повторимо крок 3.

Структура БДП залежить від вхідної послідовності. Різні послідовності можуть породжувати різні структури, не дивлячись на те, що числа у них однакові.

Алгоритм пошуку об'єкта в дереві

Наведемо кроки алгоритму.

- Крок 1. Почати з кореня.

- Крок 2. Якщо об'єкт менший, ніж ключ у вершині, то перейти до лівого сина.
- Крок 3. Якщо об'єкт більший, ніж ключ у вершині, то перейти до правого сина.
- Крок 4. Якщо об'єкт дорівнює ключу у вершині, то об'єкт знайдено; виконати потрібні дії й вийти.
- Крок 5. Повторювати кроки 2, 3 та 4, доки не досягнемо вершини, яку не визначено.
- Крок 6. Якщо досягнуто невизначену вершину, то даний об'єкт не зберігається в дереві; виконати потрібні дії й вийти.

Алгоритм додавання об'єкта до дерева

Наведемо кроки алгоритму.

- Крок 1. Почати з кореня,
- Крок 2, Якщо об'єкт менший, ніж ключ у вершині, то перейти до лівого сина.
- Крок 3. Якщо об'єкт більший, ніж ключ у вершиш, то перейти до правого сина.
- Крок 4. Повторювати кроки 2 та 3, доки не досягнемо вершини, яку не визначено (тобто її немає).
- Крок 5. Якщо досягнуто невизначену вершину, то визначити (тобто додати) вершину з новим об'єктом як ключем.

Алгоритм побудови дерева рішень – ID3

- Крок 1. Якщо вершині дерева прийняття рішень поставлено у відповідність множину прикладів R і всі ці приклади мають однакове значення атрибута прийняття рішення d, то цю вершину визначають як листок дерева та позначають цим значенням d.
- Крок 2. Якщо для певної вершини умови кроку 1 не виконано, то розглядають множину умовних атрибутів A. Якщо $A \neq \emptyset$, то вибирають довільний атрибут $a \in A$; нехай множина його значень $E_a = \{e_1, e_2, \dots, e_k\}$. Цю вершину позначають атрибутом a, сам атрибут a вилучають із множини A й виконують такі дії:
    - у вершині a утворюють ребра $e_1, e_2, \dots, e_k$ (їх кількість становить $|E_a|$, і кожне ребро позначають елементом множини E_a);
    - множину прикладів R вершини a розбивають на підмножини з однаковим значенням атрибута a (усього таких підмножин k, значення атрибута a в першій підмножині дорівнює e_1, у другій – e_2,…, у k-ий – e_k);
    - кінцю ребра e_j ставлять у відповідність підмножину прикладів, у яких значення атрибута a дорівнює e_j.
- Крок 3. Якщо на кроці 2 виявиться, що A $\neq \emptyset$, то щодо вершини, яка має стати листком, приймають спеціальне рішення залежно від специфіки задачі.

Отже, кожна внутрішня вершина є коренем піддерева, якому відповідають усі приклади з однаковим значенням одного з атрибутів і різними значеннями атрибута прийняття рішень. Кожному листку дерева відповідають приклади, що мають однакові значення одного з атрибутів і однакові значення атрибута прийняття рішень.