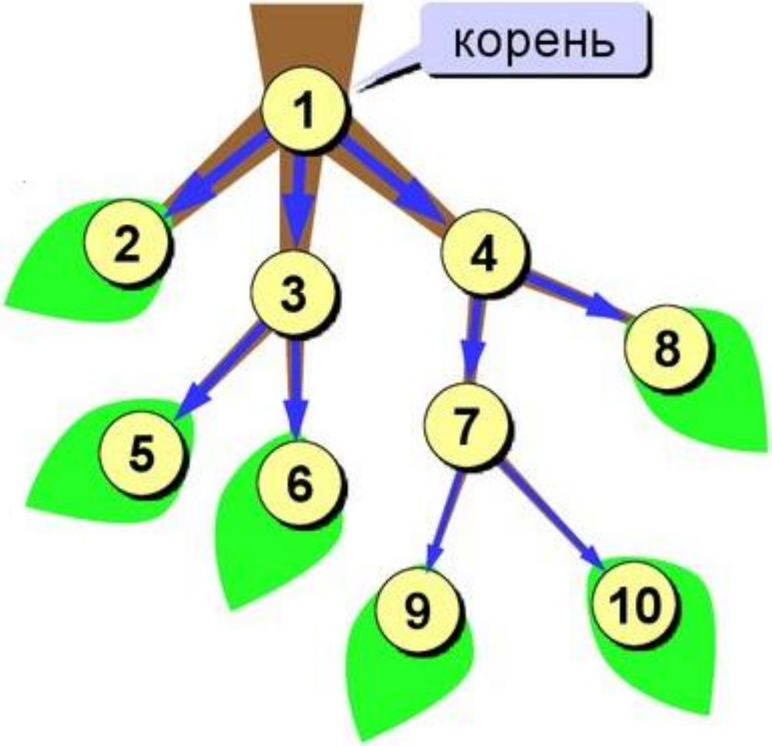
Дерева

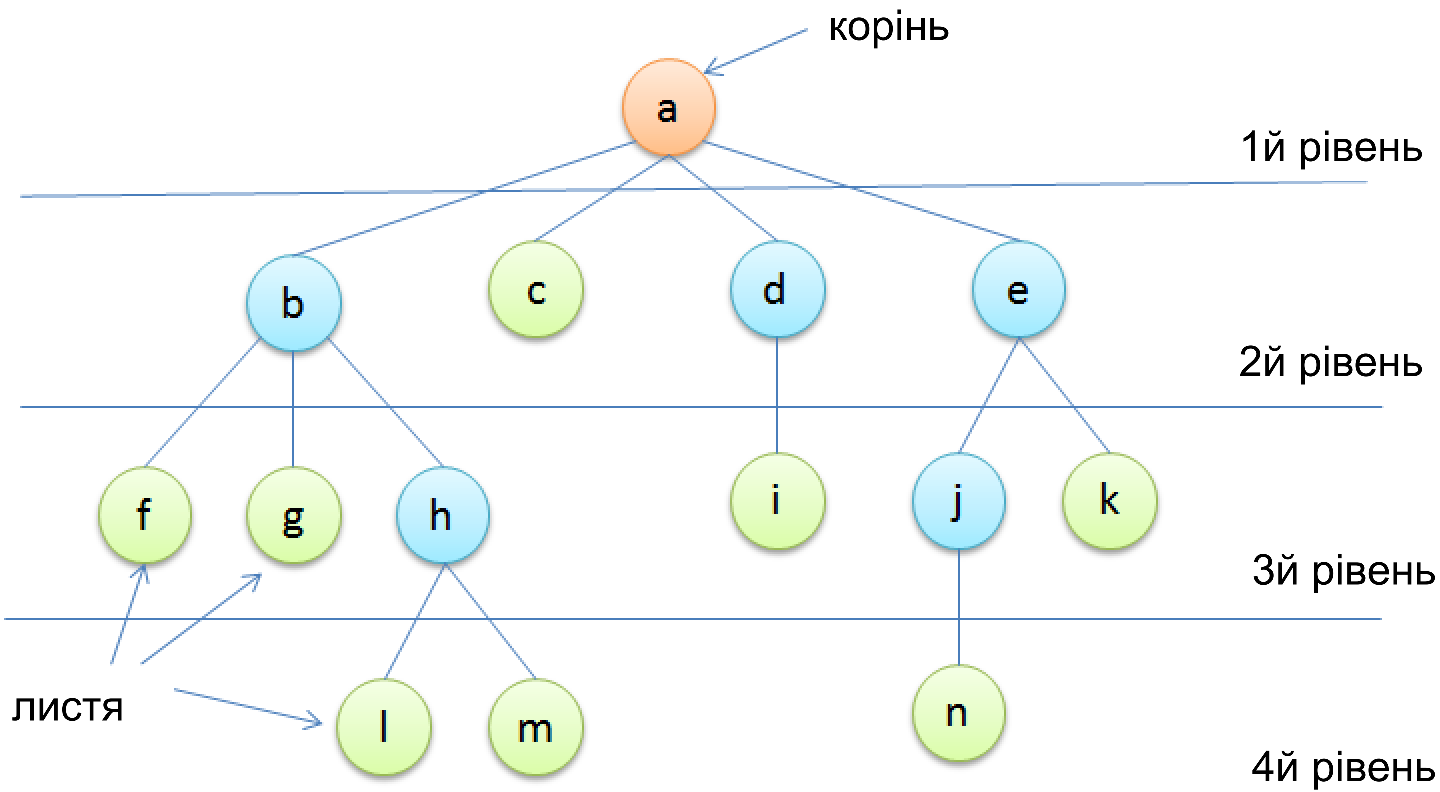


https://ppt-online.org/10946

Визначення

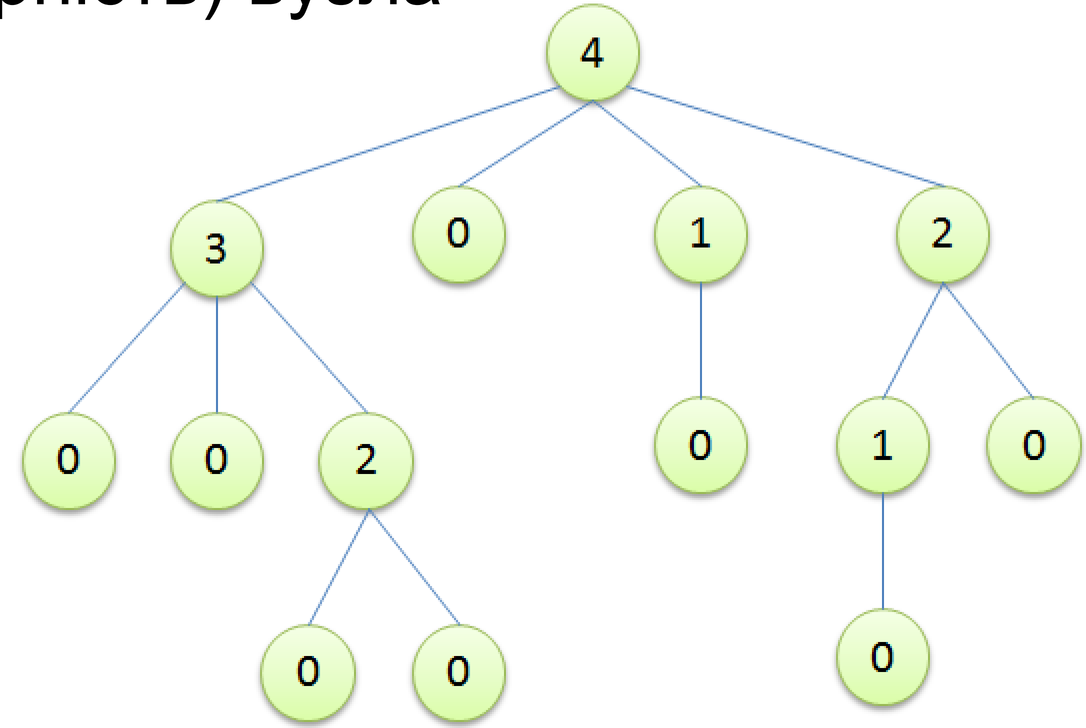
Непорожнє дерево – це скінченна множина T, що складається з одного або більше вузлів. Виділяють:

- один спеціально позначений вузол – корінь даного дерева;
- інші вузли (за винятком кореня), розташовані в $m \geq 0$ попарно непересічних множинах T_1, T_2, ..., T_m, кожна із яких, у свою чергу, є дерево. T_1, T_2, ..., T_m називають піддеревами цього кореня.



Усі вершини, в які входять дуги, що виходять із однієї вершини, називають **нащадками** даної вершини, а саму вершину – **предком**

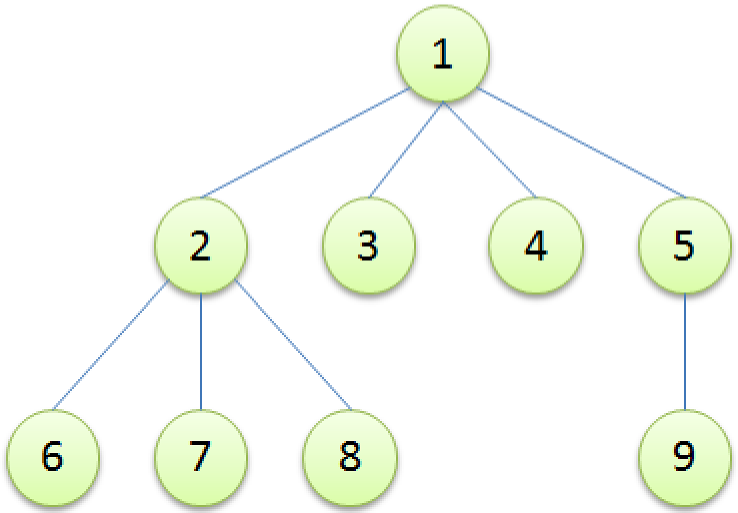
Степень (арність) вузла

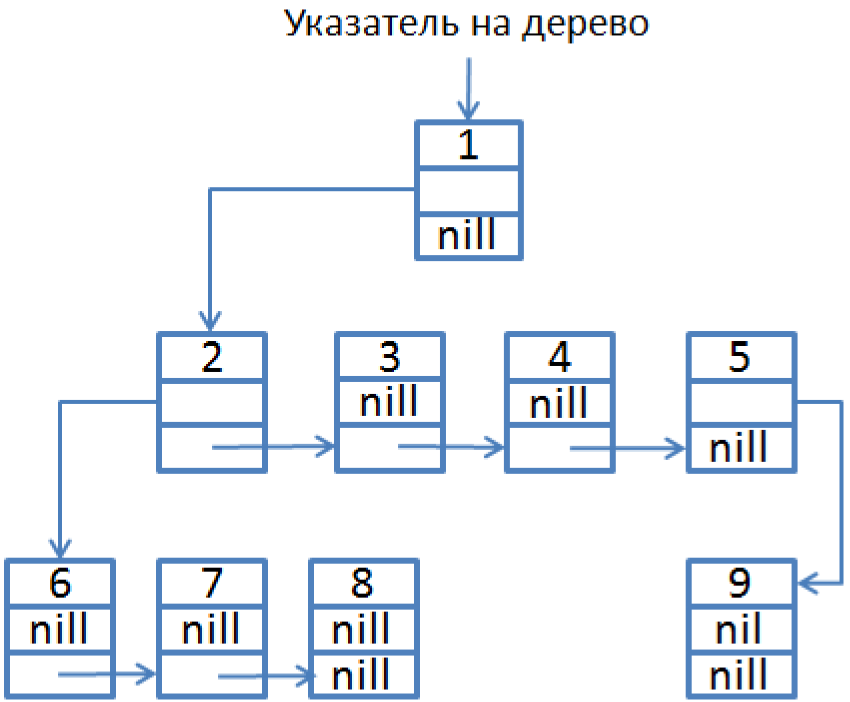


Степень дерева дорівнює максимальному степеню вершины, що входить у дерево

Якщо в дереві всі вузли мають один і той самий степінь, то їх називають *регулярними*

Реалізація





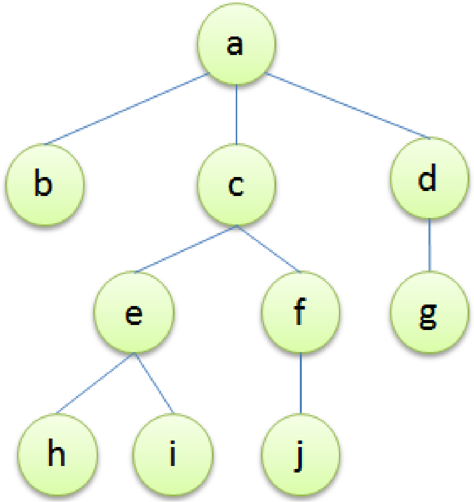
* Існують також реалізації, в яких кожен вузол має покажчик на вузол-предок

Шляхом із вузла n_i у вузол n_k називають послідовність вузлів $n_1, n_2, \ldots, n_k$, де для всіх $i, 1 < i < k$, вузол n_i є предок вузла n_{i+1}.

Довжиною шляху називають число, на одиницю менше від кількості вузлів, що складають цей шлях. Таким чином, шляхом нульової довжини буде шлях із будь-якого вузла до самого себе.

Висотою вузла дерева називають довжину найдовшого шляху з цього вузла до будь-якого листка. **Висота дерева** збігається з висотою кореня.

Глибину вузла визначають як довжину шляху (він єдиний) від кореня до даного вузла.

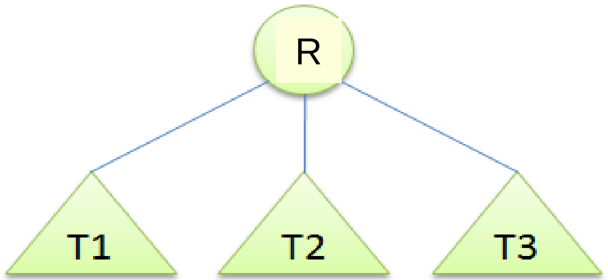


Обходи дерев: у глибину або у ширину

Існує декілька варіантів обходу (відвідування усіх вузлів) дерева **у глибину**.

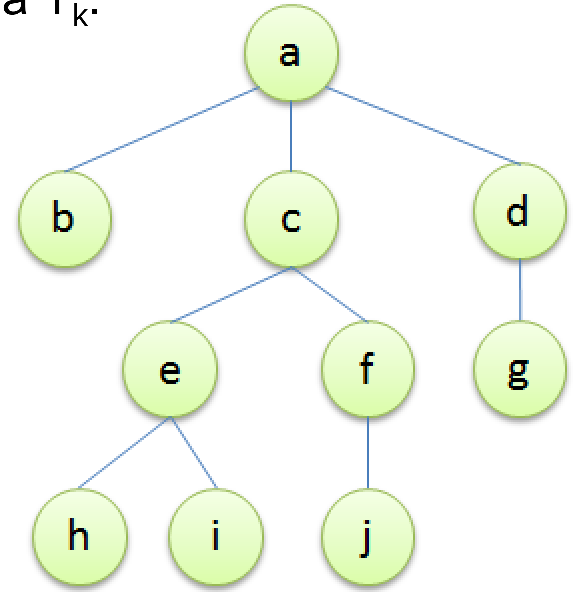
Рекурсивно ці способи можна визначити таким чином:

- якщо T є нульове дерево, то в список обходу заносять порожній запис;
- якщо T складається з одного вузла, то в список обходу записують даний вузол;
- якщо T – дерево із коренем R і піддеревами T1 T2, ..., Tk. Тоді для різних способів обходу матимемо таке:



Обхід в прямому порядку (префіксний)

У разі проходження в прямому порядку вузлів дерева T спочатку відвідують корінь R, потім у прямому порядку – вузли піддерева T_1, далі всі вузли піддерева T_2 і т.д. також у прямому порядку. Останніми відвідують вузли піддерева T_k.



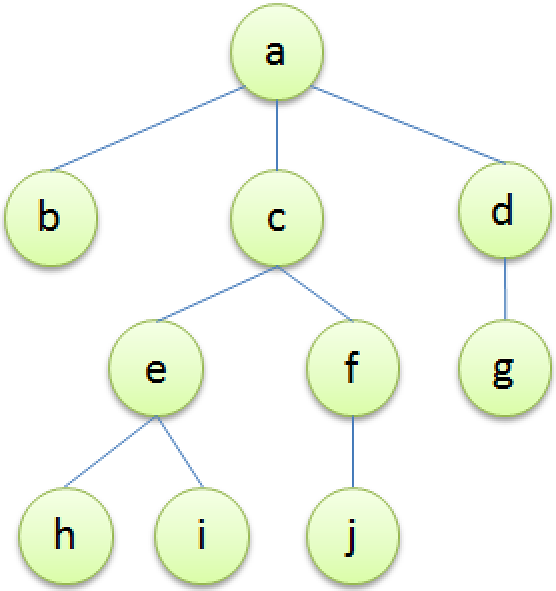
Обход вершин в прямом порядке:

a, b, c, e, h, i, f, j, d, g.

```
PreOrder (R: корінь)
Відвідати вершину R
// Нехай T1, T2…, Tk – нащадки вершини R
for  i = 1 to k do
    PreOrder(Ti)
```

Обхід у зворотному порядку (постфіксний)

Під час обходу у зворотному порядку спочатку відвідують у зворотному порядку всі вузли піддерева T1, потім послідовно всі вузли піддерев T2, ..., Tk (також у зворотному порядку), останнім відвідують корінь R.



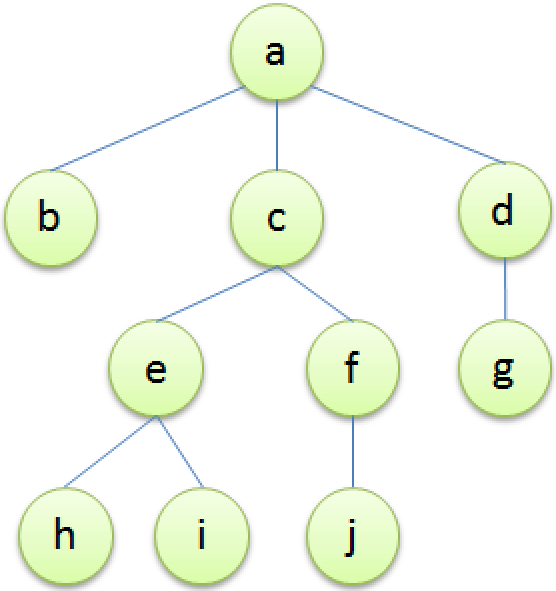
Обход вершин в обратном порядке:

b, h, i, e, j, f, c, g, d, a.

PostOrder(R: корінь)
// Нехай $T_1, T_2 \ldots, T_k$ – нащадки вершини R
for i = 1 to k do
  PostOrder(T_i)
Відвідати вершину R

Обхід в симетричному (інфиксному) порядку

У разі симетричного обходу вузлів дерева T спочатку відвідують у симетричному порядку всі вузли піддерева T_1, далі корінь R, потім послідовно в симетричному порядку всі вузли піддерев T_2, ..., T_k.



Обход вершин в симметричном порядке:

b, a, h, e, i, c, j, f, g, d.

```
InOrder (R: корінь);
if R – листок then відвідати вершину R
else // Нехай T₁, T₂…, Tₖ – нащадки вершини R
    InOrder (T₁)
    Відвідати вершину R
    for  i = 2 to k do
        InOrder(Tᵢ);
```

Обхід у ширину

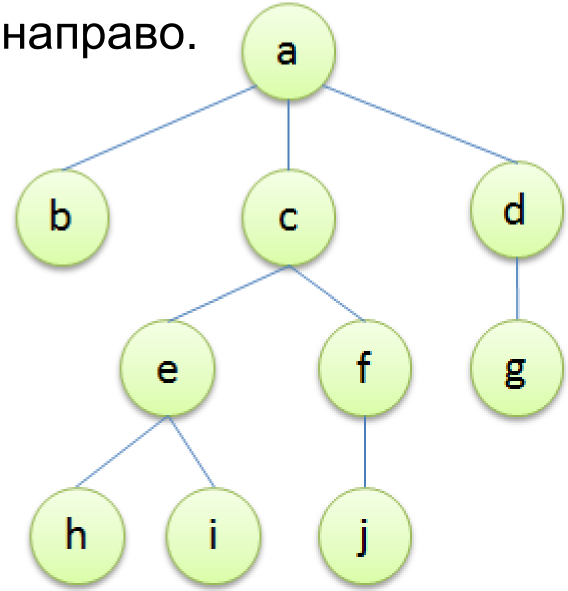
Помістити корінь дерева в порожню чергу O;

while черга O не пуста do

    // Нехай p перша вершина черги O;

    Відвідати вершину p і видалити її з O;

    Помістити всіх нащадків вершини p у чергу O зліва направо.

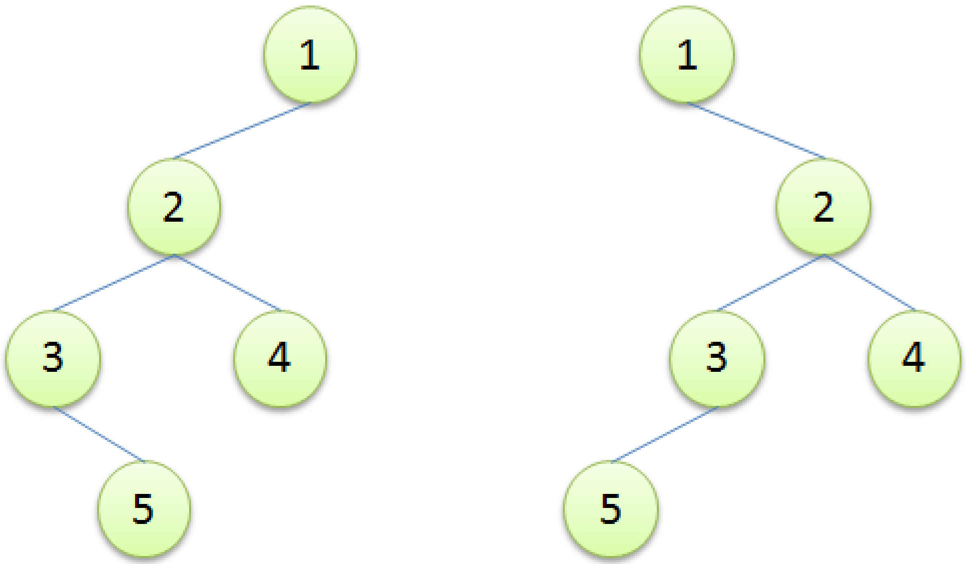


Обход вершин в ширину:

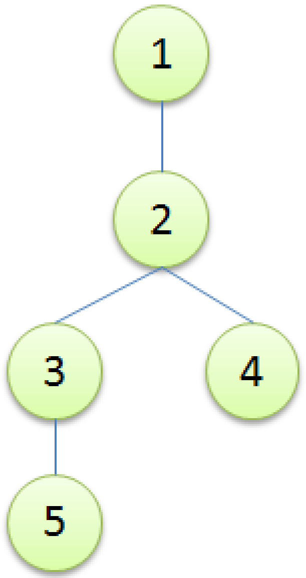
a, b, c, d, e, f, g, h, i, j.

Двійкове (бінарне) дерево

Бінарне дерево – це дерево зі степенем не більше двох, тобто кожен його вузол має не більше двох нащадків, які називають лівим і правим

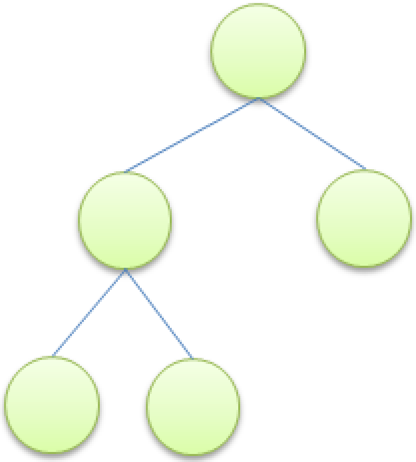


Два різні бінарні дерева

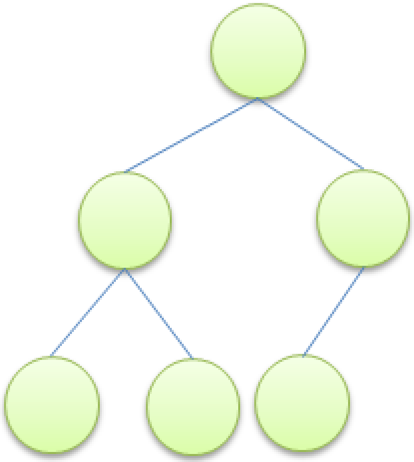


довільне дерево зі степенем два

За степенем вершин виділяють такі двійкові дерева:
строгі – вершини дерева мають степінь два або нуль (листки);
нестрогі – вершини дерева можуть мати степінь як один, так і два, і нуль.

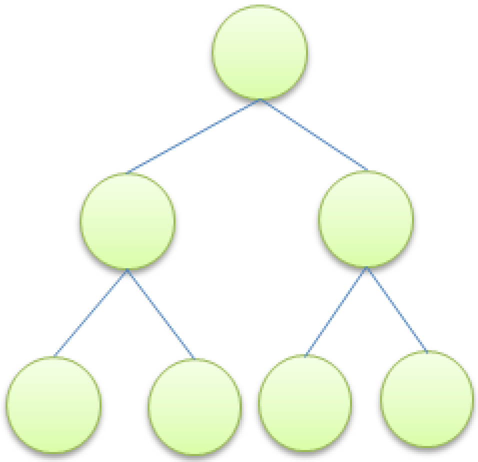


строге

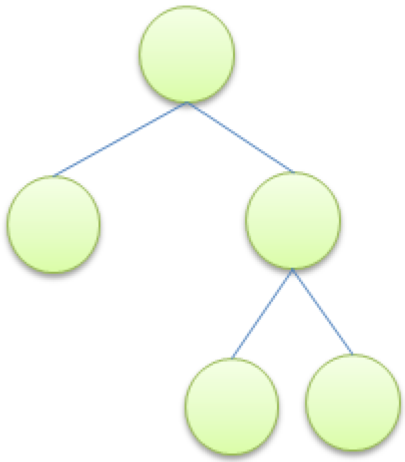


нестроге

У загальному випадку на k-му рівні бінарного дерева може бути до 2^{k-1} вершин. Двійкове дерево, що містить тільки повністю заповнені рівні (тобто 2^{k-1} вершин на кожному k-му рівні), називають *повним.*

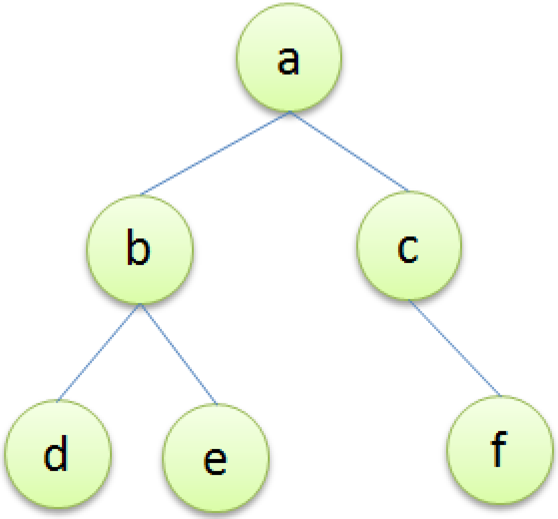


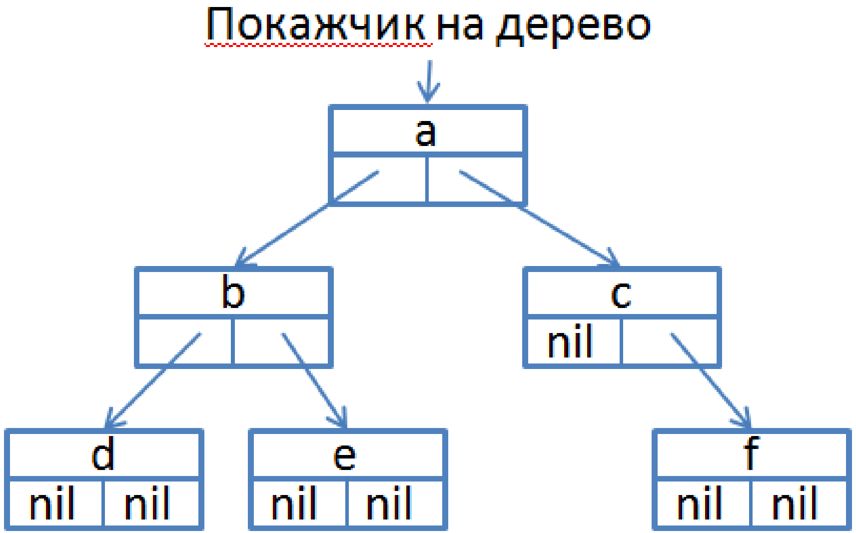
повне



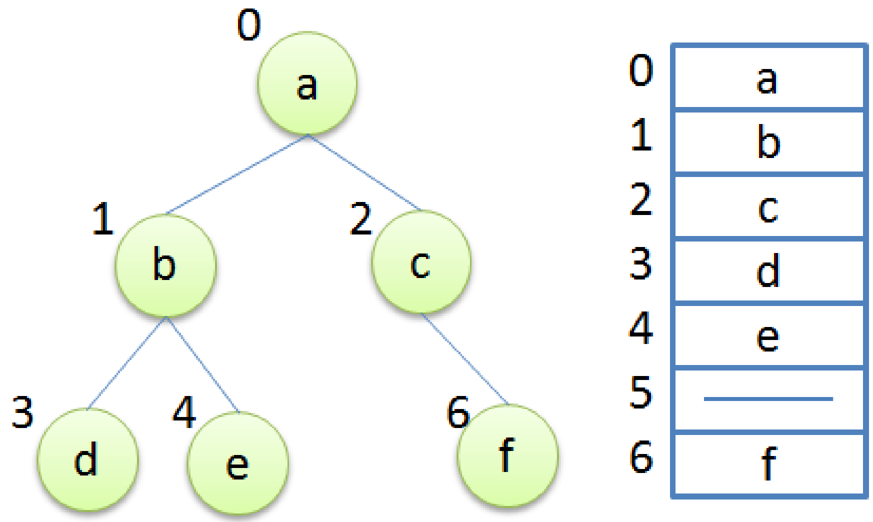
неповне

Бінарні дерева. Динамічна реалізація





Бінарні дерева. Статична реалізація



Адреса будь-якої вершини в масиві :

$$2^{k-1}+i-2,$$

k – номер рівня вершини;

i – номер на рівні k в повному двійковому дереві.

Для будь-якого вузла з індексом *j* у масиві можна обчислити:

адресу_лівого нащадка = 2*j+1,

адресу_правого нащадка = 2*j+2.

Двійкове (бінарне) дерево пошуку

Бінарне дерево пошуку (англ. *binary search tree*, *BST*) – це двійкове дерево, для якого виконуються такі додаткові умови (властивості дерева пошуку):

- для довільного вузла X значення ключів усіх вузлів лівого піддерева менше значення ключа самого вузла X, а значення ключів усіх вузлів правого піддерева (того ж вузла X) більші значення ключа вузла X:

$$key[X.left] < key[X] \leq key[X.right]$$

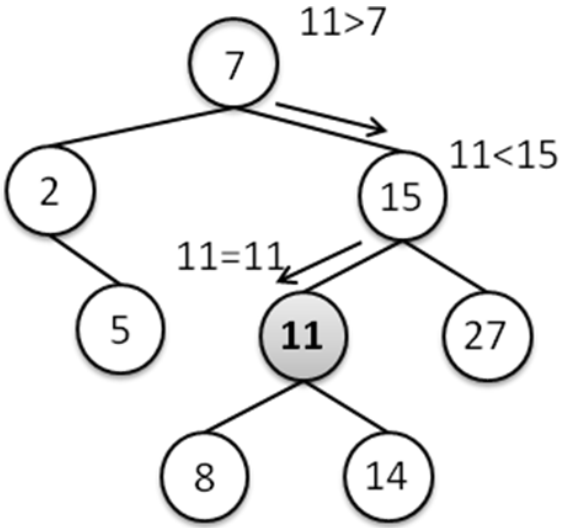
- для довільного вузла X обидва піддерева – ліве і праве – є двійкові дерева пошуку.

Алгоритм пошуку елемента за ключем

1) якщо дерево порожнє, повідомити, що вузол не знайдений і зупинитися;

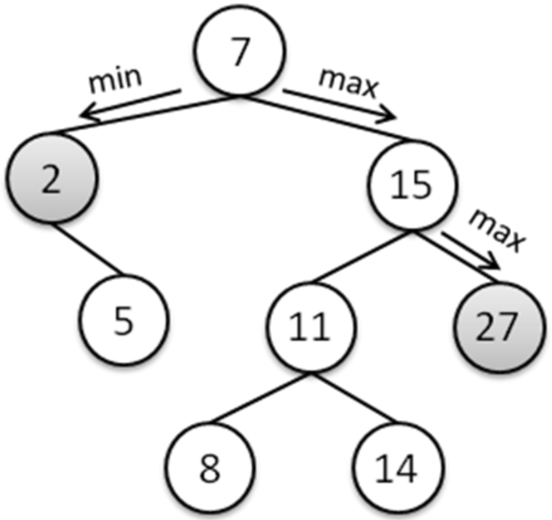
2) в іншому випадку порівняти елемент пошуку зі значенням ключа кореневого вузла:

- у випадку рівності видати посилання на цей вузол і зупинитися;
- якщо шуканий елемент більший за ключ, рекурсивно шукати в правому піддереві, якщо менший – у лівому.



Пошук мінімального й максимального елементів

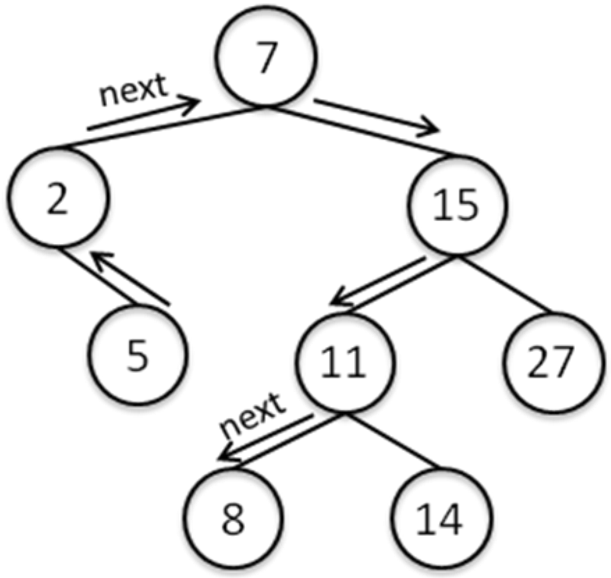
Щоб знайти мінімальний елемент у бінарному дереві пошуку, необхідно рухатися за покажчиками на лівий нащадок від кореня дерева до зустрічі зі значенням null. Відповідний листовий елемент і буде мінімальним у дереві. Аналогічно шукають максимальний елемент, тільки рухатися потрібно за покажчиками на правий нащадок



Алгоритм пошуку наступного за значенням ключа елемента:

- якщо у вузла є праве піддерево, то наступний за ним елемент буде мінімальним елементом у ньому;

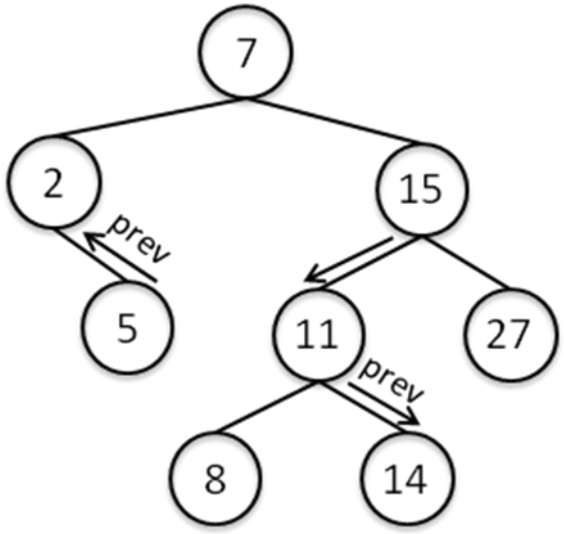
- в іншому випадку слід перевірити, чи є вузол лівим нащадком свого предка, і рухатися вгору до зустрічі з таким елементом. Його предок і буде шуканим значенням.



Алгоритм пошуку попереднього за значенням ключа елемента:

- якщо у вузла є ліве піддерево, то попередній елемент буде максимальним елементом у цьому піддереві;

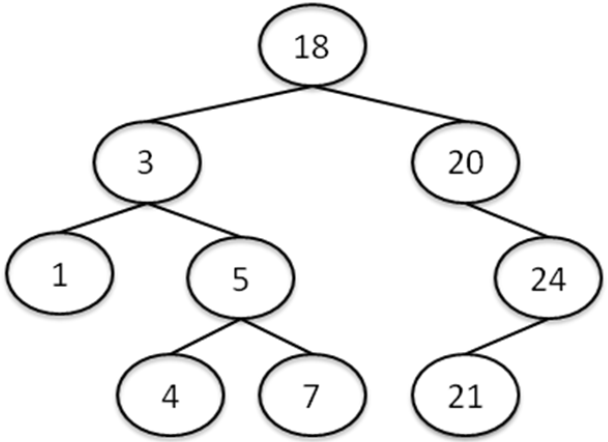
- якщо у вузла немає лівого піддерева, то необхідно перевірити, чи є вузол правий нащадок свого предка й рухатися вгору до зустрічі з таким елементом. Його предок і буде шуканим значенням.



Алгоритм додавання елемента в бінарне дерево пошуку:

1) якщо дерево порожнє, слід замінити його на дерево з одним кореневим вузлом і зупинитися;
2) в іншому випадку необхідно порівняти ключ елемента K із ключем кореневого вузла X:
   - якщо K > X, рекурсивно додати елемент у праве піддерево;
   - якщо K < X, рекурсивно додати елемент у ліве піддерево;
   - якщо K=X, то існують різні варіанти (рекурсивно додати у праве піддерево, замінити значення вузла на нове, не здійснювати вставлення, організувати список значень у вузлі).

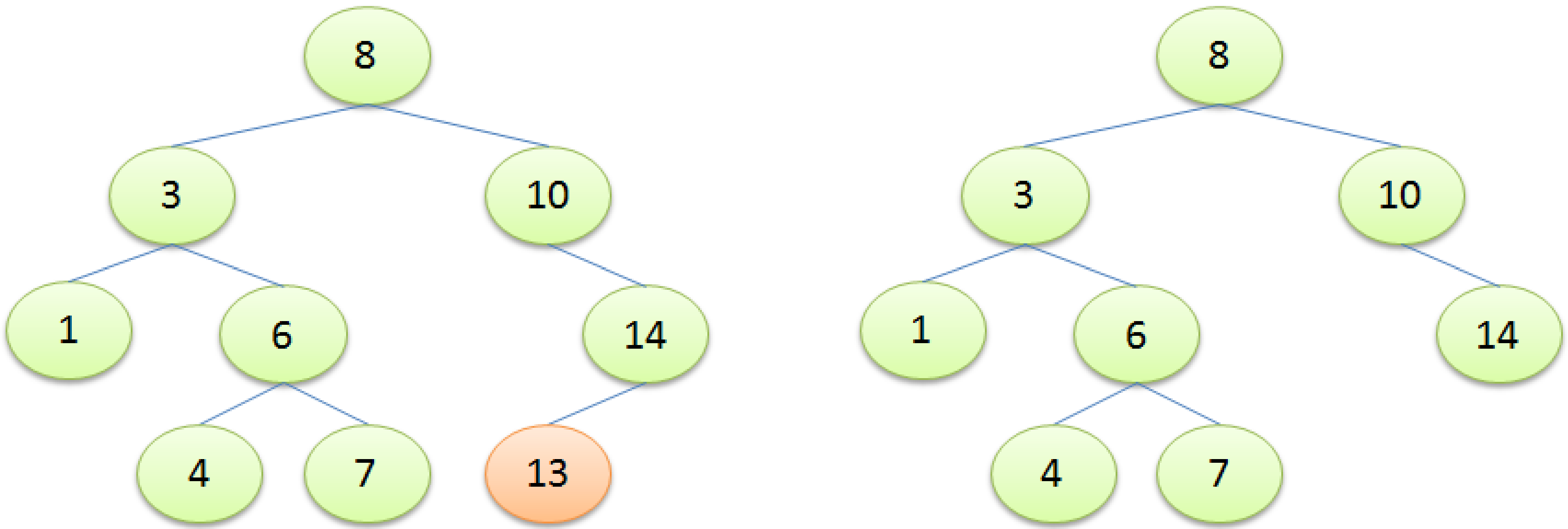
Приклад. Із набору даних 18, 3, 20, 24, 5, 21, 1, 4, 7 побудувати двійкове дерево пошуку



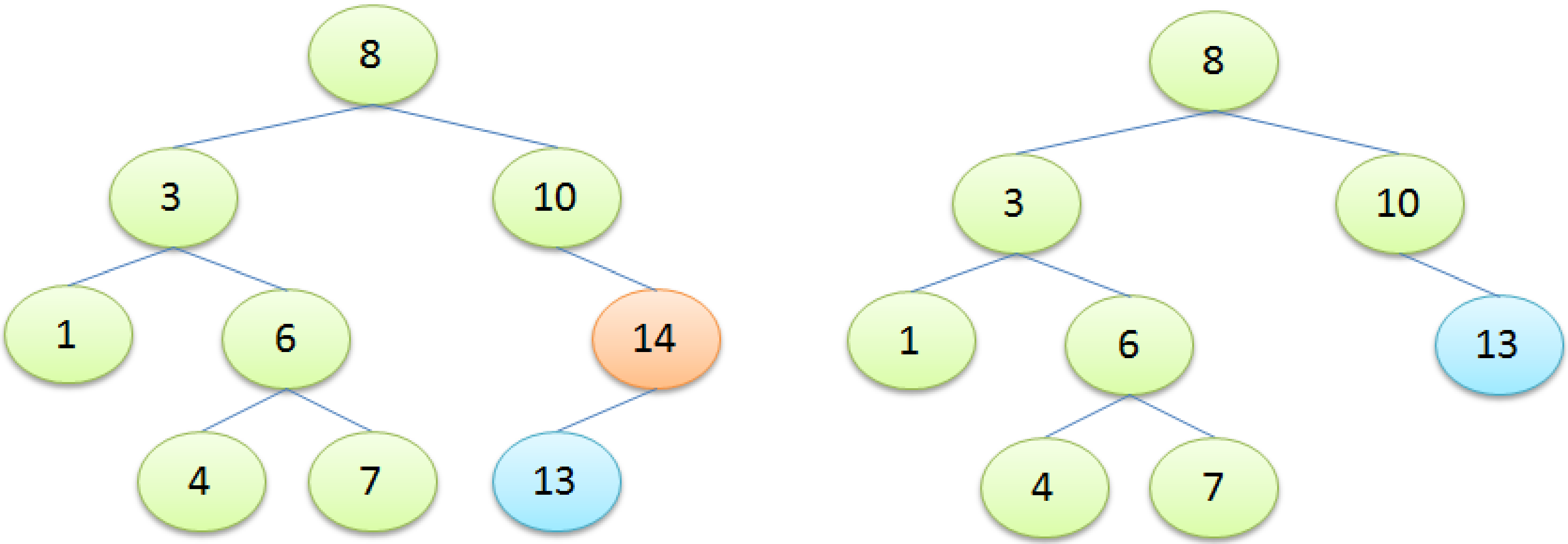
Алгоритм видалення елемента з бінарного дерева пошуку

Для видалення вузла потрібно розглянути три можливі ситуації:

1) якщо у вузла немає дочірніх вузлів, тобто він є листковий, то у його предка слід просто замінити відповідний покажчик на nil



2) якщо у вузла є тільки один дочірній вузол, то необхідно створити новий зв’язок між предком і нащадком вузла, що видаляють



3) якщо у вузла два нащадки, то потрібно знайти наступний (мінімум із правого піддерева) або попередній (максимум із лівого піддерева) за значенням ключа елемент і перемістити його на місце видаленого вузла