SQL Workshop

Tree (B* tree) Traversal

sjapit@uow.edu.au

19 March 2024

Agenda

- Binary Search Tree (BST)
- Why B-Tree?
  - 2-4 Tree
- B* Tree


Binary Search
Tree
(Revision)

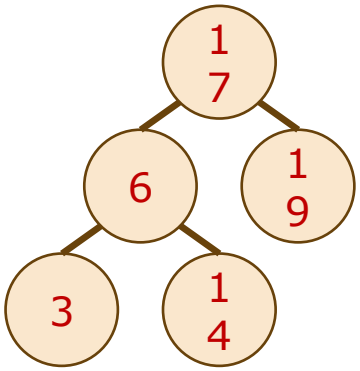
Binary Search Tree

A Binary search tree is:

- A **Binary Search Tree** is a data structure used in computer science for organizing and storing data in a sorted manner.
- Each node in a **Binary Search Tree** has at most two children, a **left** child and a **right** child
- The value in each node is greater than or equal to all the values in its left child or any of that child's descendants
- The value in each node is less than all the values in its right child or any of that child's descendants

Binary Search Tree

- The inorder traversal of a binary search tree produces an ordered list.



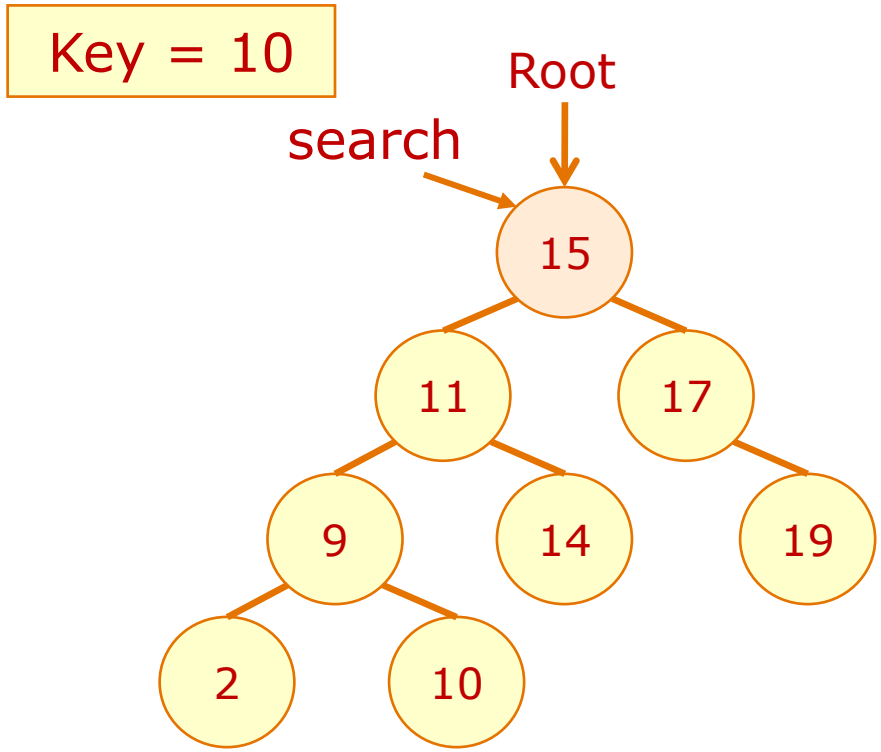
Inorder: 3, 6, 14, 17, 19

Binary Search Tree

Search

Problem:

- Given a "key" and a pointer "search" to the root
- Return the pointer to the node whose value is equal to key; return nil if there is no match

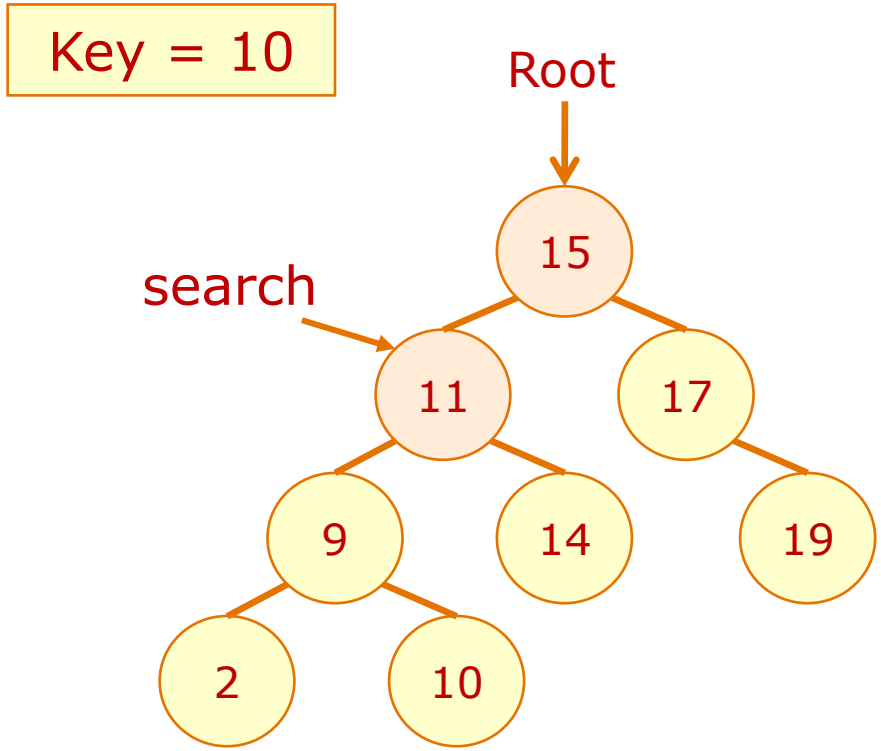


Binary Search Tree

Search

Problem:

- Given a "key" and a pointer "search" to the root
- Return the pointer to the node whose value is equal to key; return nil if there is no match

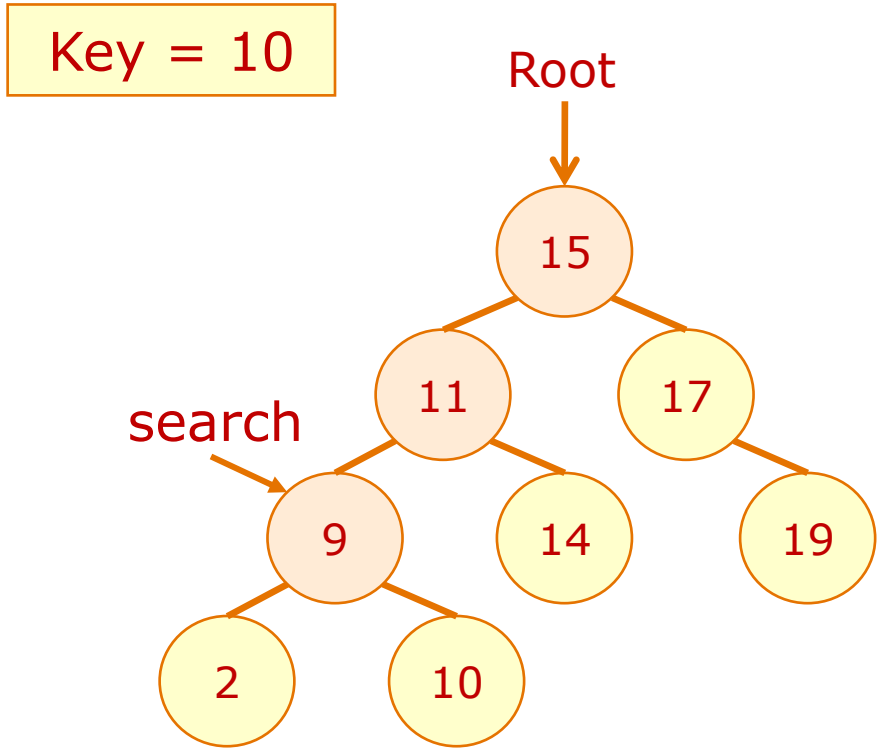


Binary Search Tree

Search

Problem:

- Given a "key" and a pointer "search" to the root
- Return the pointer to the node whose value is equal to key; return nil if there is no match

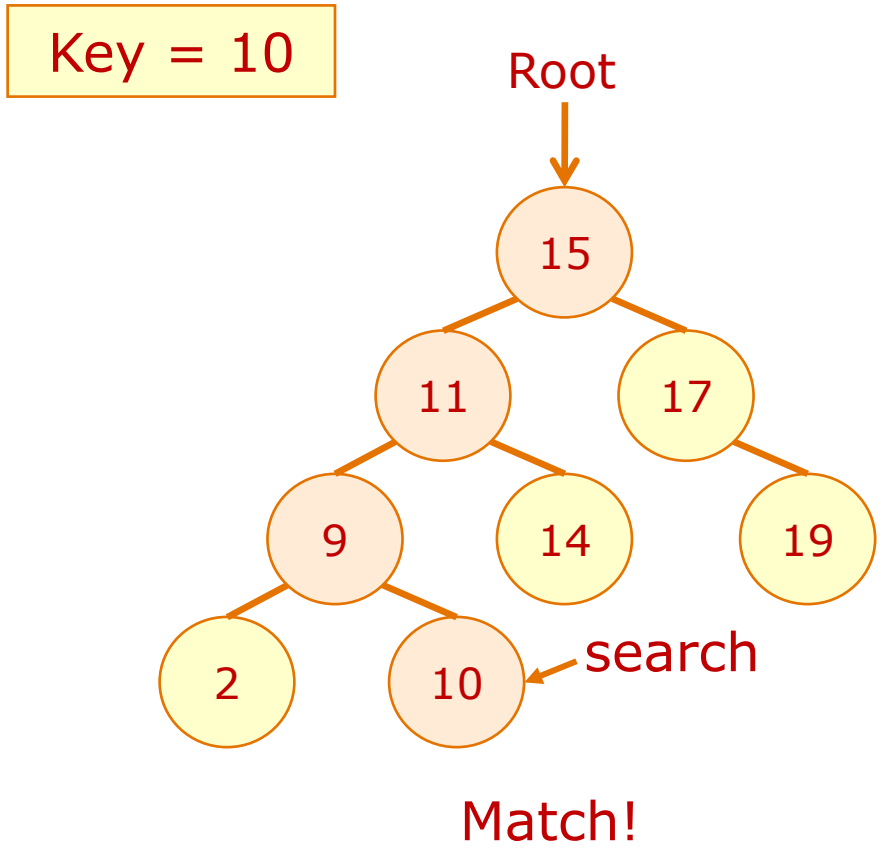


Binary Search Tree

Search

Problem:

- Given a "key" and a pointer "search" to the root
- Return the pointer to the node whose value is equal to key; return nil if there is no match

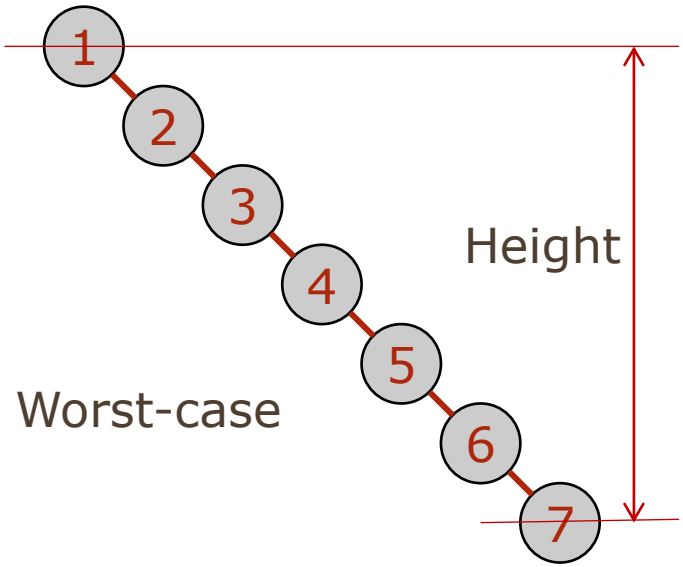


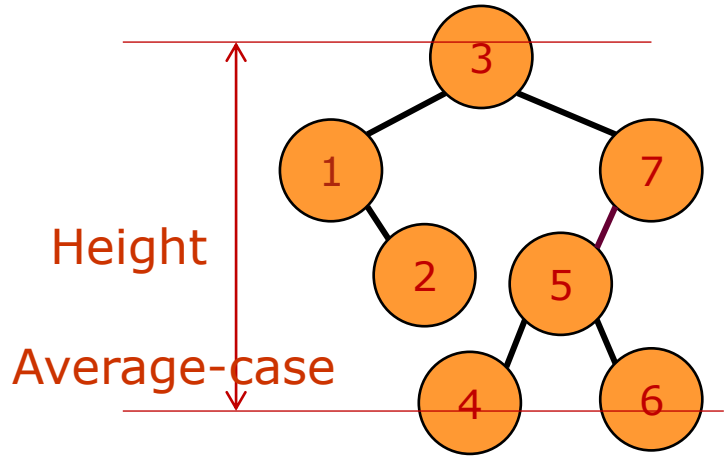
Binary Search Tree

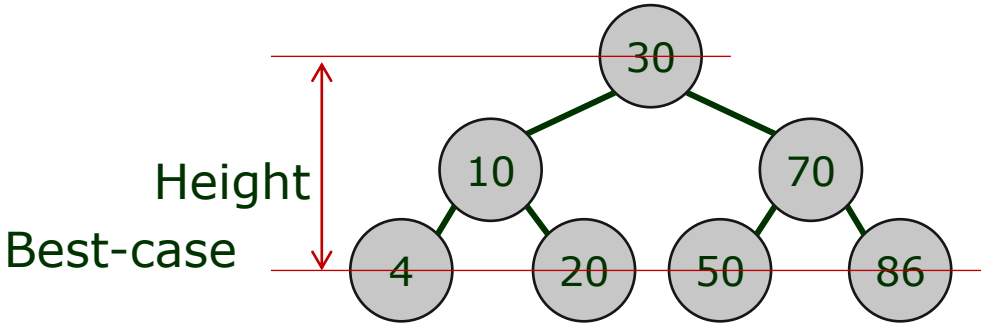
Idea:

- BST property
- Compare key with the $search \rightarrow value$
  - if $key == (search \rightarrow value) \implies match!$
  - if $key < (search \rightarrow value)$
    - Search for the node on the $left$ sub-tree
    - else search for the node on the $right$ sub-tree

Worst-case, Best-case and Average-case BST

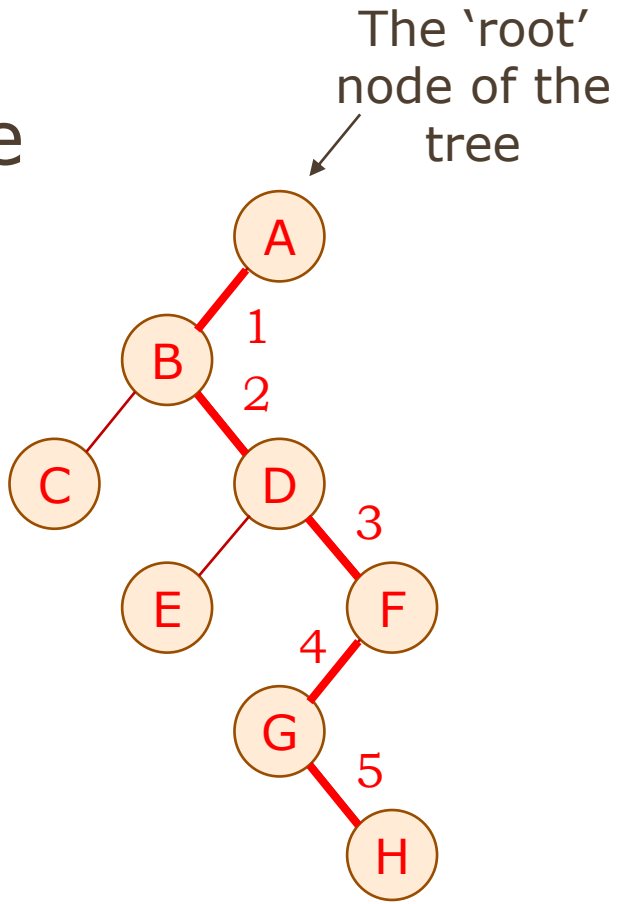






Tree

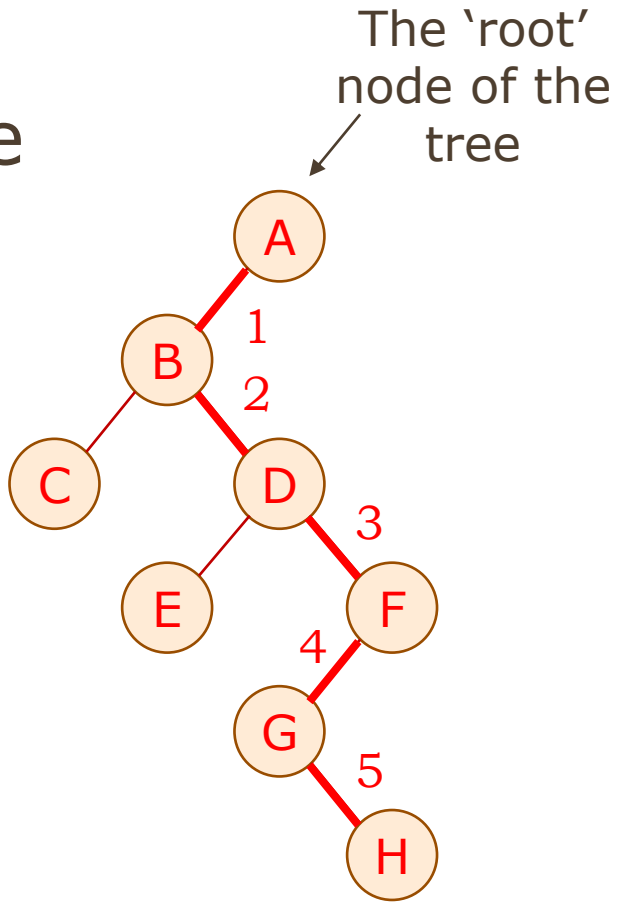
- The height of a node is the number of edges from the node to its most distance leaf node.



Tree

- The height of a node is the number of edges from the node to its most distance leaf node.

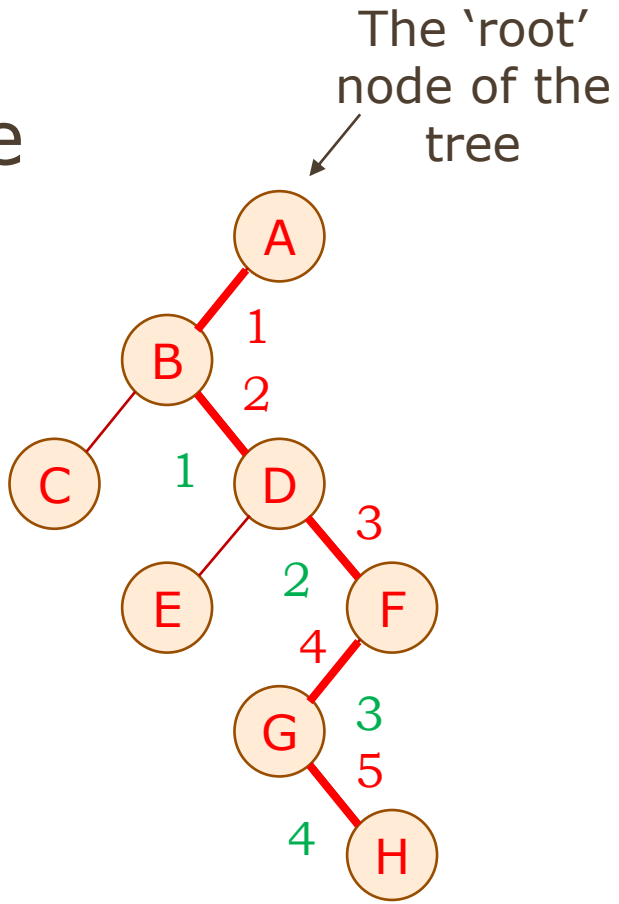
The height of the node A is 5.



Tree

- The height of a node is the number of edges from the node to its most distance leaf node.

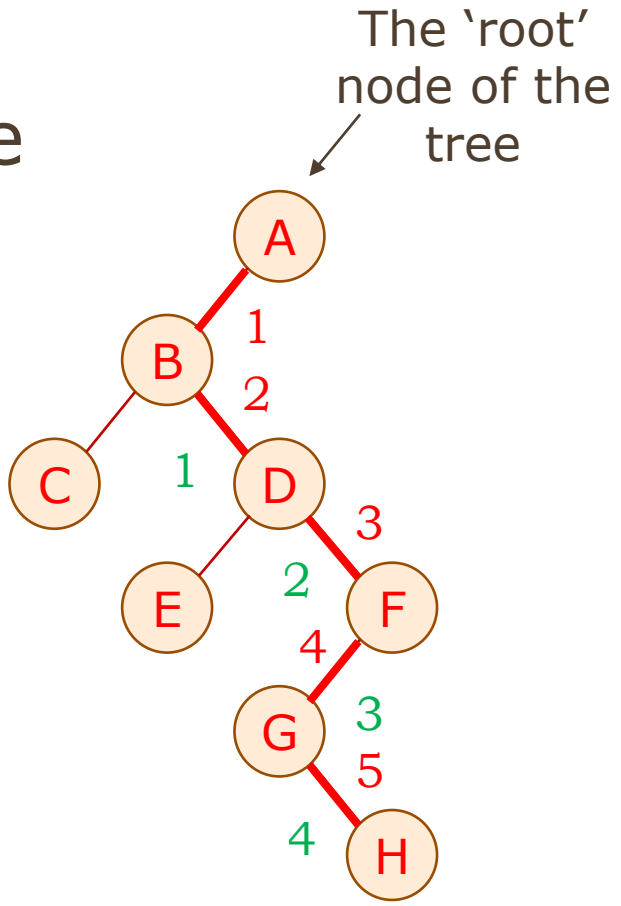
The height of the node A is 5.
The height of the node B is 4.



Tree

- The height of a node is the number of edges from the node to its most distance leaf node.

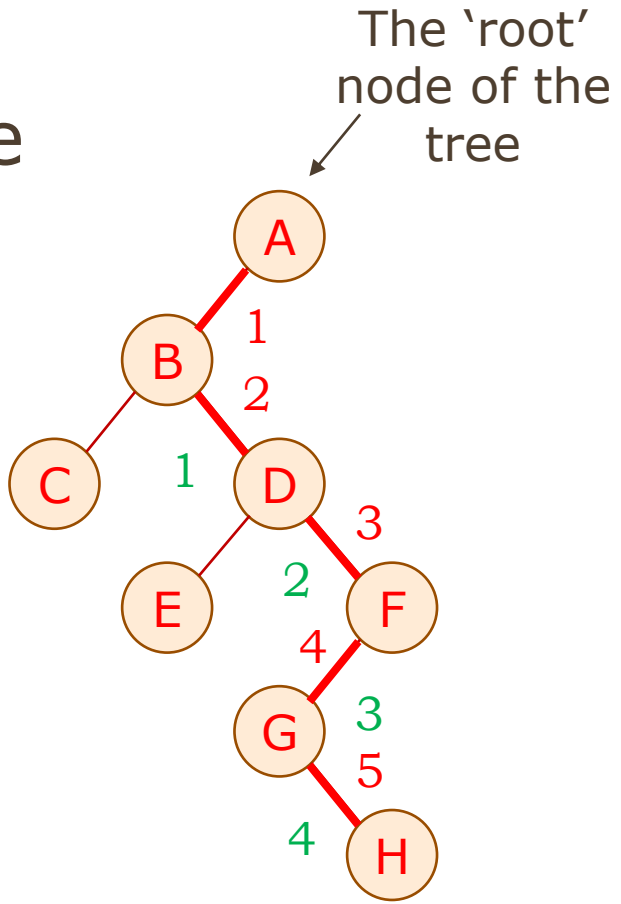
The height of the node A is 5.
The height of the node B is 4.
The height of the node C is 0.



Tree

- The height of a node is the number of edges from the node to its most distance leaf node.

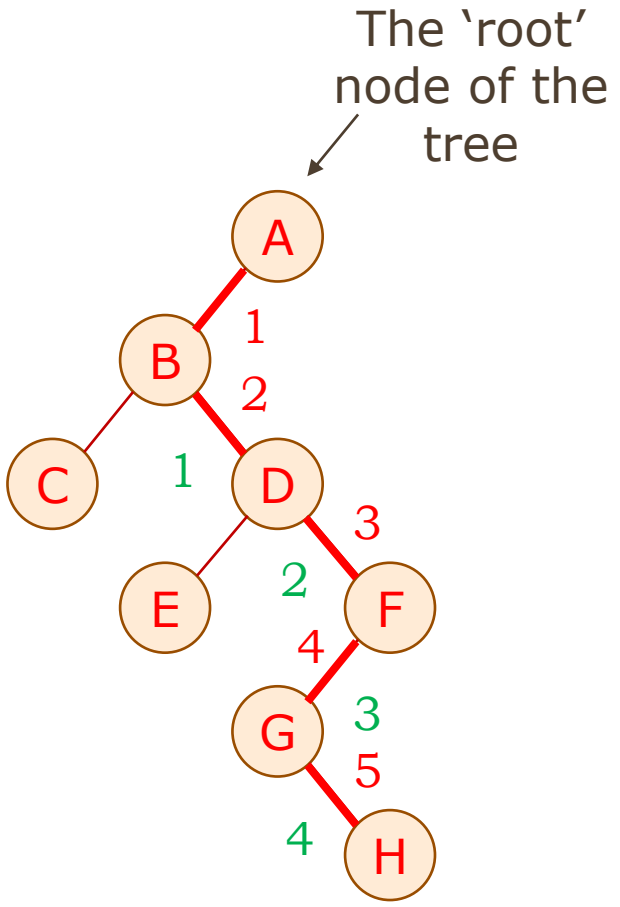
The height of the node A is 5.
The height of the node B is 4.
The height of the node C is 0.
The height of the node G is 1.



Tree

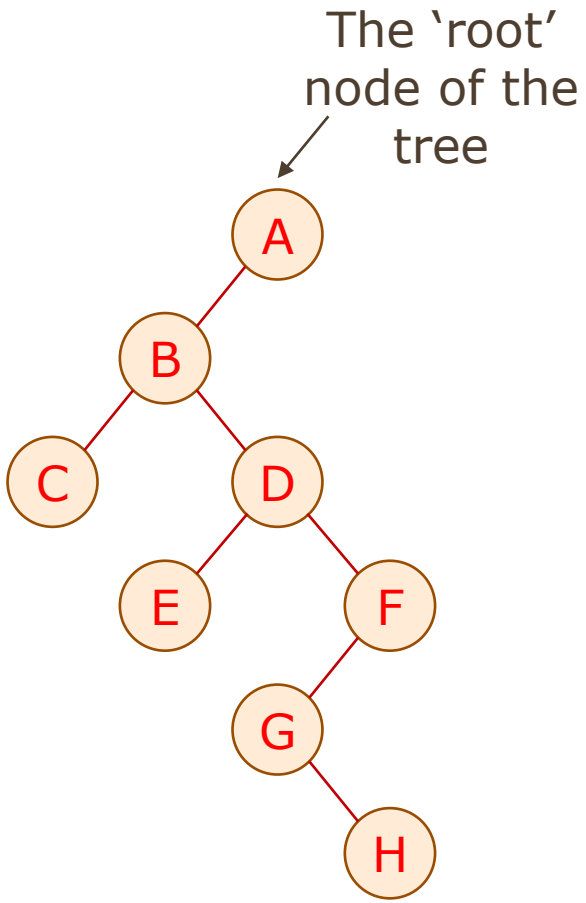
Note:

- The height of the leaf node of a path will have a height of 0.
- The height of the root of a tree is the height of the tree.



Tree

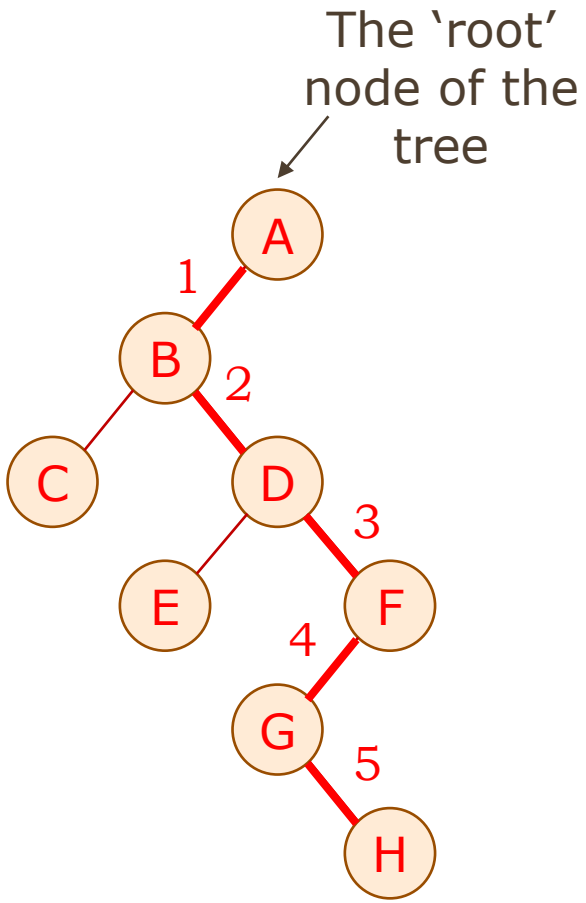
- The depth of a node is the number of edges in the path from the root node to that node.



Tree

- The depth of a node is the number of edges in the path from the root node to that node.

The depth of the node H is 5.

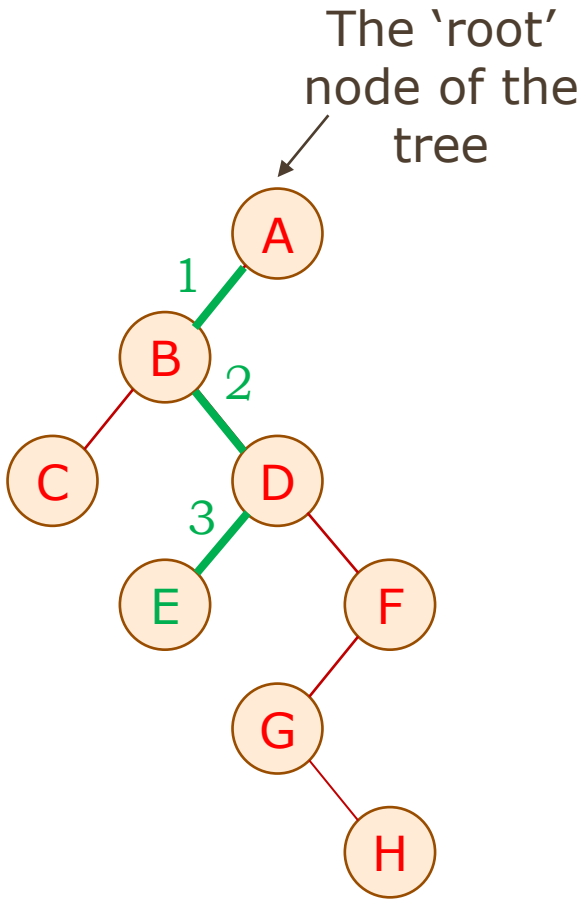


Tree

- The depth of a node is the number of edges in the path from the root node to that node.

The depth of the node H is 5.

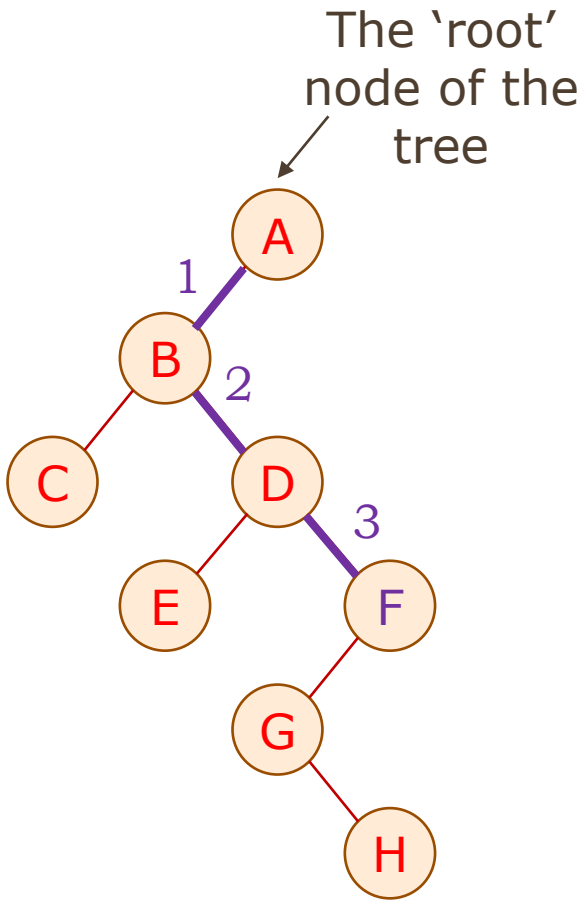
The depth of the node E is 3.



Tree

- The depth of a node is the number of edges in the path from the root node to that node.

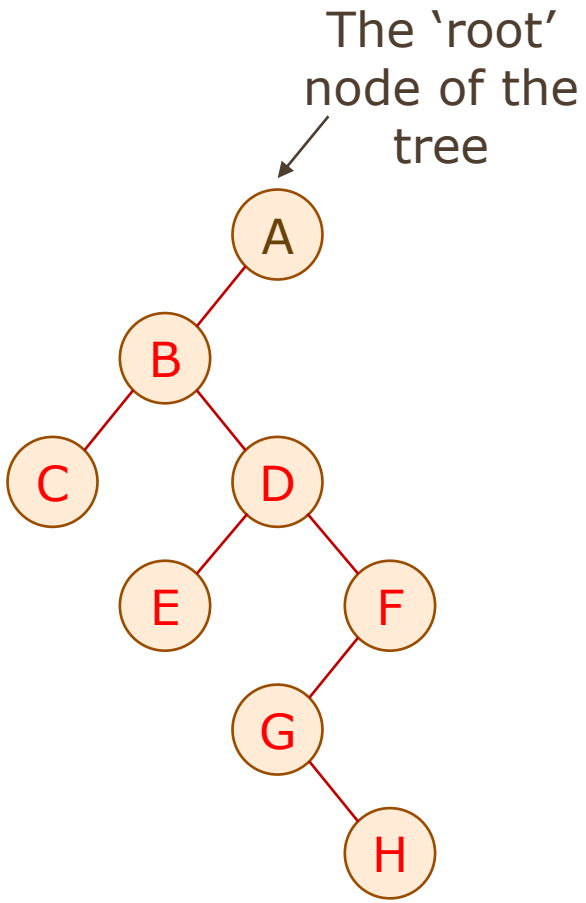
The depth of the node H is 5.
The depth of the node E is 3.
The depth of the node F is 3.



Tree

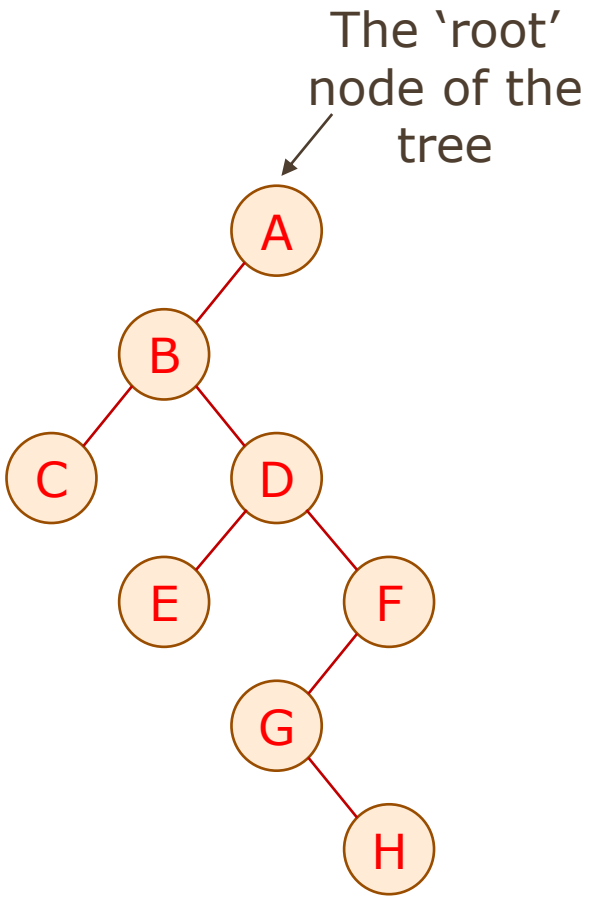
- The depth of a node is the number of edges in the path from the root node to that node.

The depth of the node H is 5.
The depth of the node E is 3.
The depth of the node F is 3.
The depth of the node A is 0.



Tree

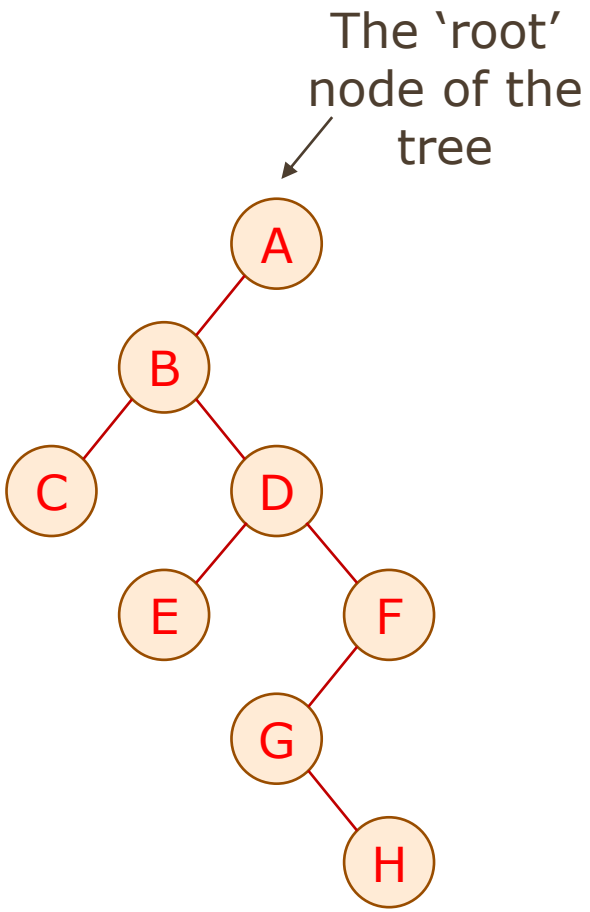
- The level of a node is equal to the number of edges along the unique path between the node and the root node of the tree.



Tree

- The level of a node is equal to the number of edges along the unique path between the node and the root node of the tree.

  The level of the node A = 0.

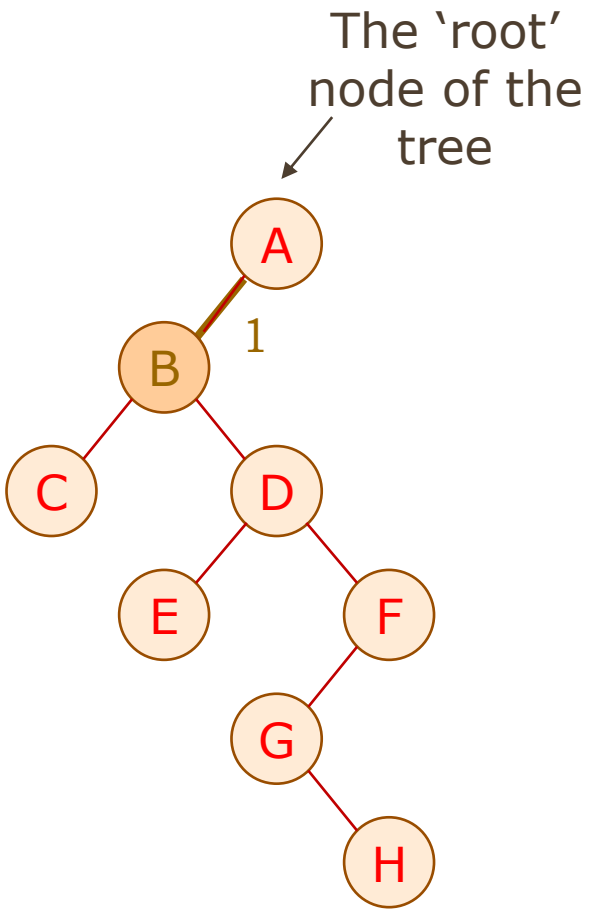


Tree

- The level of a node is equal to the number of edges along the unique path between the node and the root node of the tree.

  The level of the node A = 0.

  The level of the node B = 1.



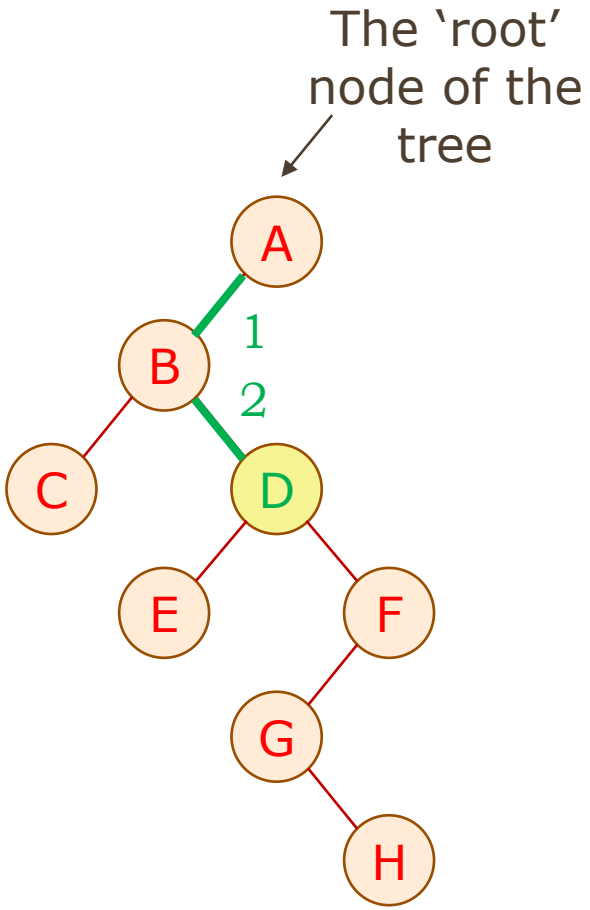
Tree

- The level of a node is equal to the number of edges along the unique path between the node and the root node of the tree.

  The level of the node A = 0.

  The level of the node B = 1.

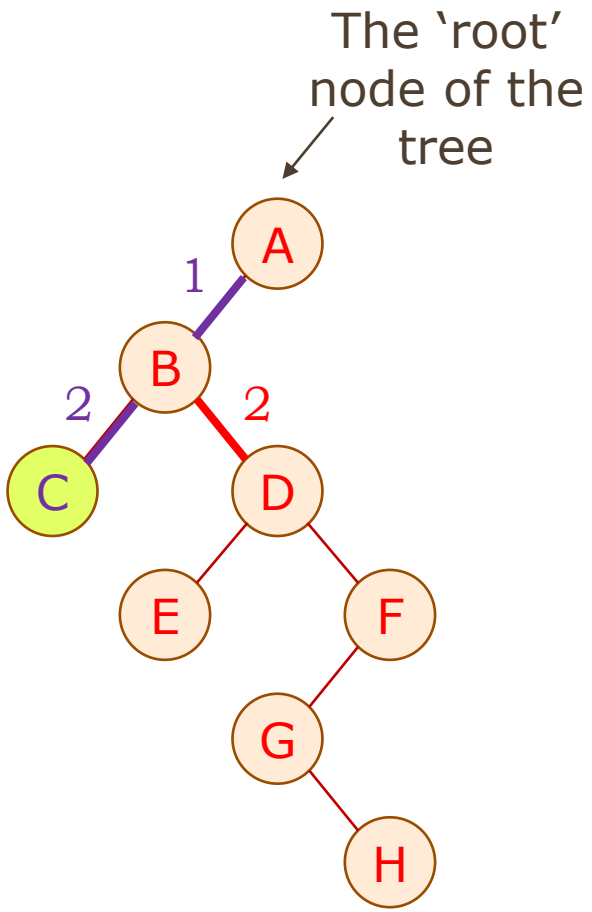
  The level of the node D = 2.



Tree

- The level of a node is equal to the number of edges along the unique path between the node and the root node of the tree.

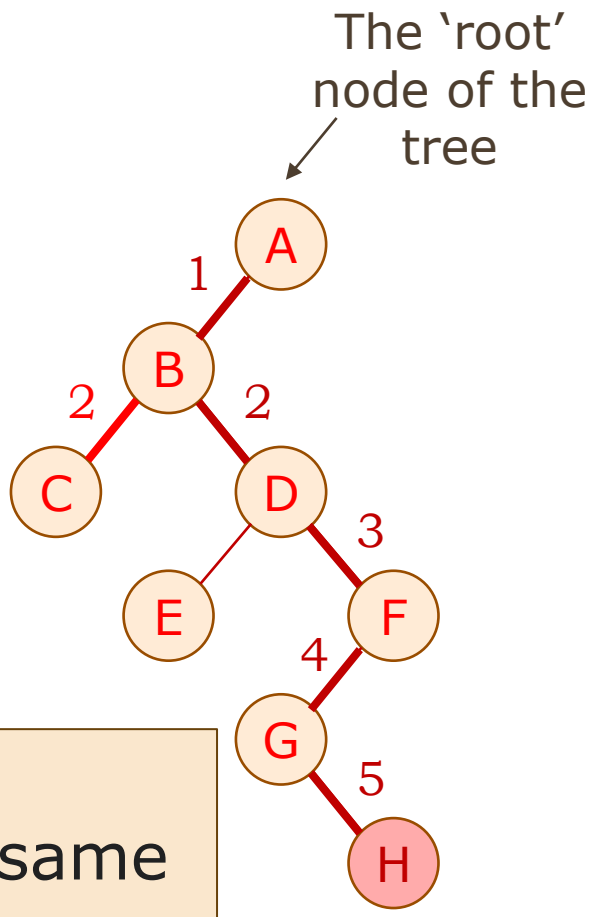
  The level of the node A = 0.
  The level of the node B = 1.
  The level of the node D = 2.
  The level of the node C = 2.



Tree

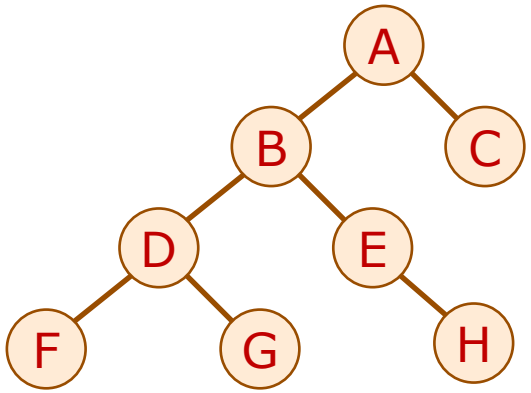
- The level of a node is equal to the number of edges along the unique path between the node and the root node of the tree.

The level of the node A = 0.
The level of the node B = 1.
The level of the node D = 2.
The level of the node C = 2.
The level of the node H = 5.



Note:
- The depth of a node will be the same as the level of a node.

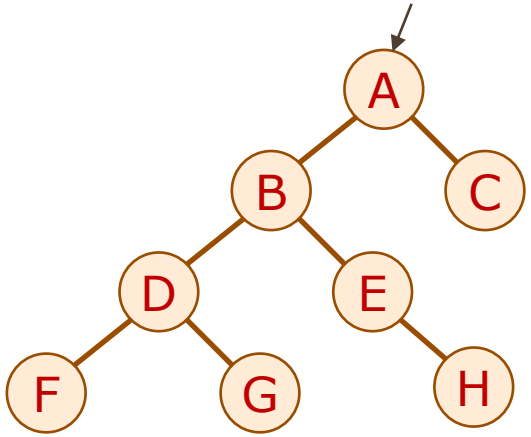
Tree



Node	Height	Depth	Level
A			
B			
C			
D			
E			
F			
G			
H			

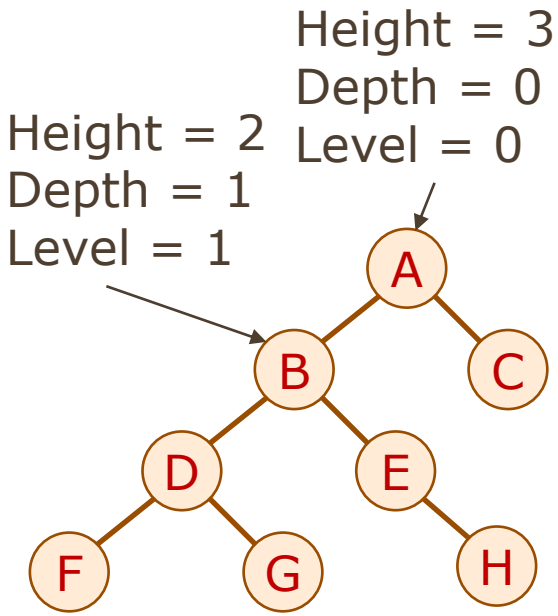
Tree

Height = 3
Depth = 0
Level = 0



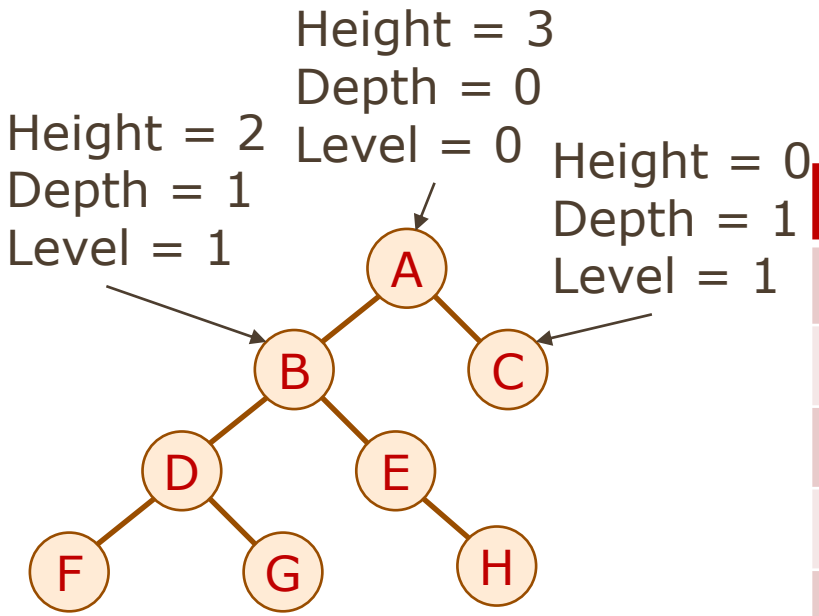
Node	Height	Depth	Level
A	3	0	0
B			
C			
D			
E			
F			
G			
H			

Tree



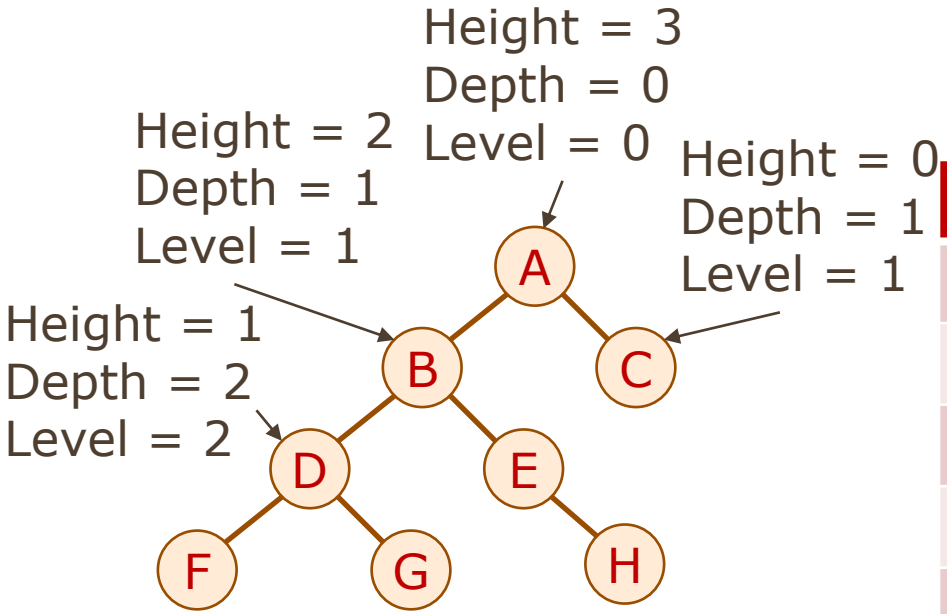
Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C			
D			
E			
F			
G			
H			

Tree



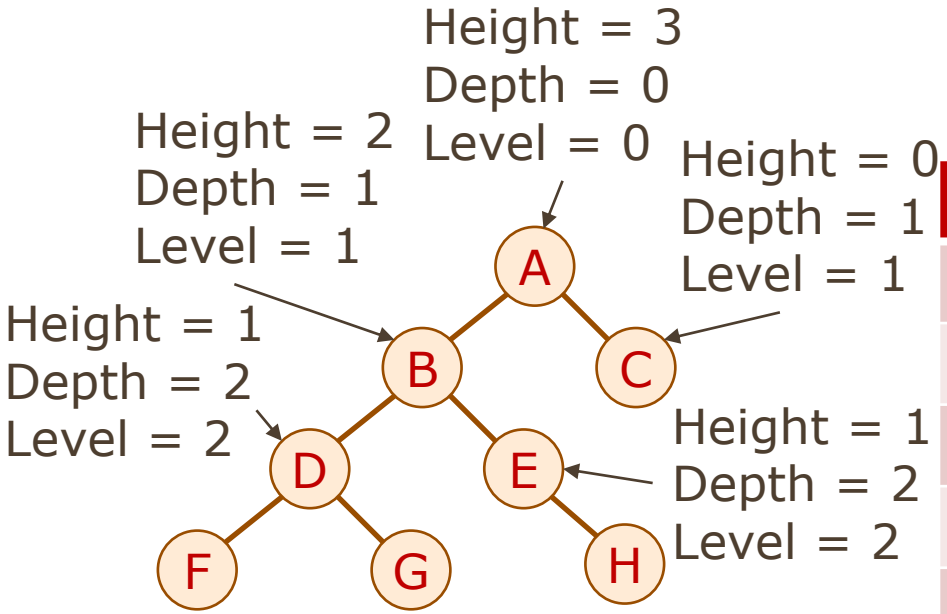
Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C	0	1	1
D			
E			
F			
G			
H			

Tree



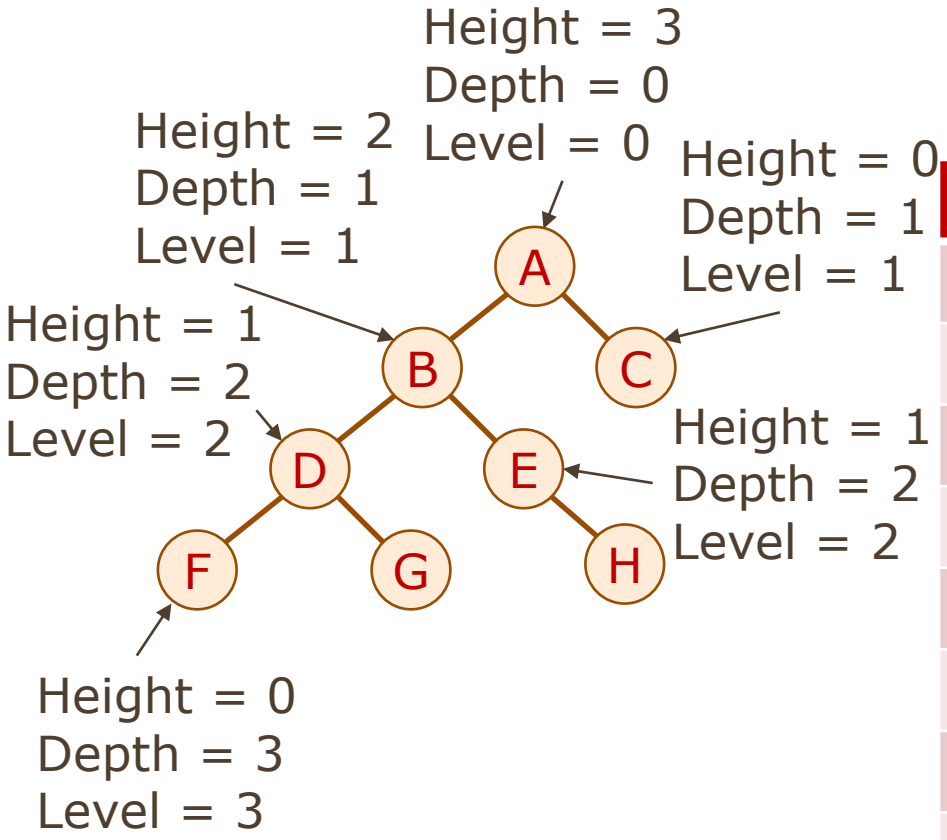
Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C	0	1	1
D	1	2	2
E			
F			
G			
H			

Tree



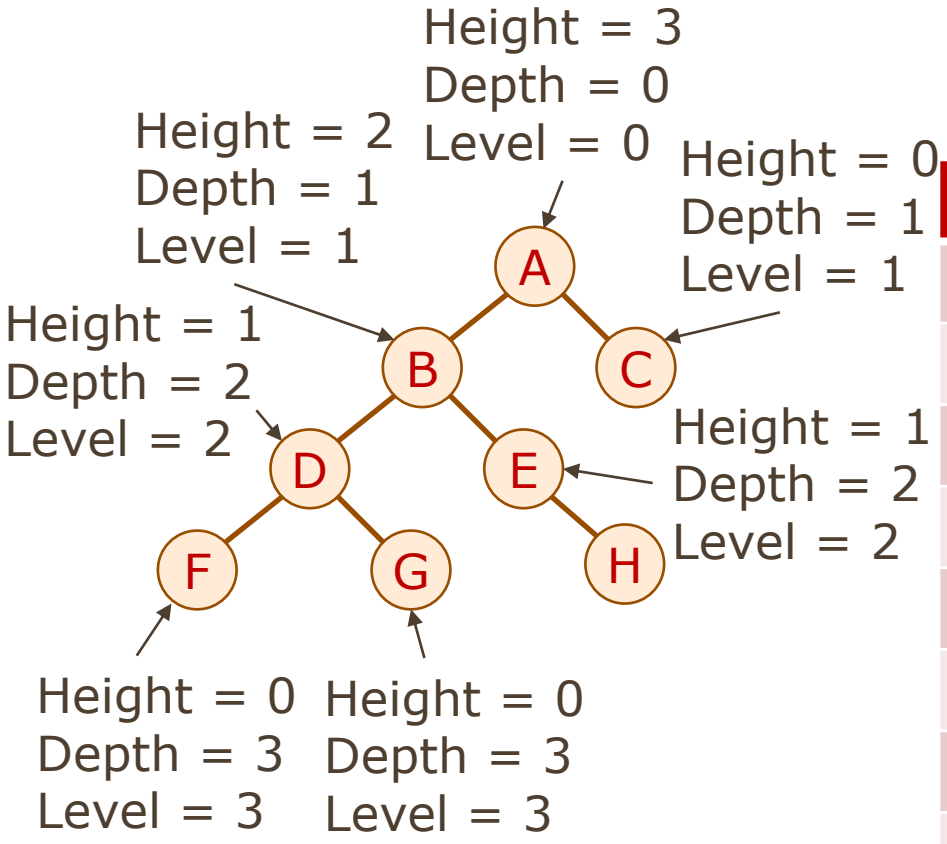
Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C	0	1	1
D	1	2	2
E	1	2	2
F			
G			
H			

Tree



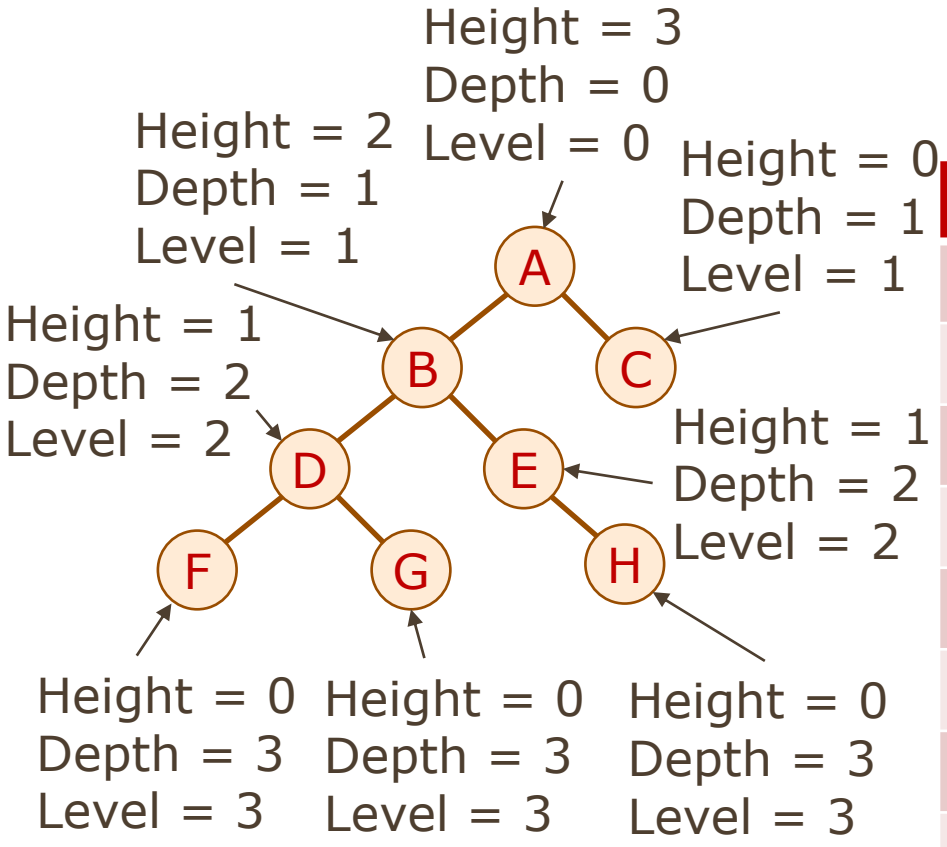
Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C	0	1	1
D	1	2	2
E	1	2	2
F	0	3	3
G			
H			

Tree



Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C	0	1	1
D	1	2	2
E	1	2	2
F	0	3	3
G	0	3	3
H			

Tree

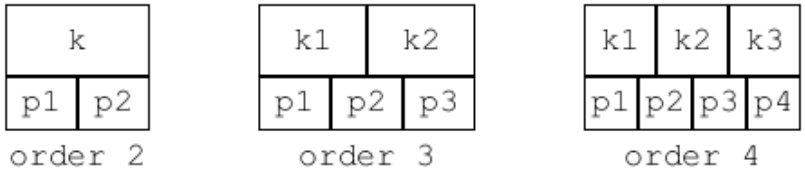


Node	Height	Depth	Level
A	3	0	0
B	2	1	1
C	0	1	1
D	1	2	2
E	1	2	2
F	0	3	3
G	0	3	3
H	0	3	3

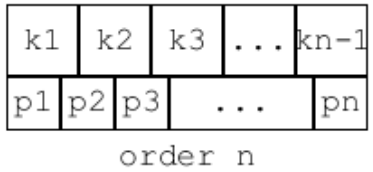
m-way Tree

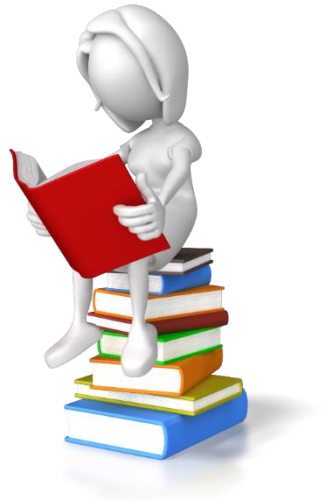
m-Way (multi-way) Tree

- An $m - way$ tree is a search tree in which each node can have from 0 to m sub-trees, where m is defined as the order of the tree.



Order n tree, has up to $n - 1$ keys in each node.



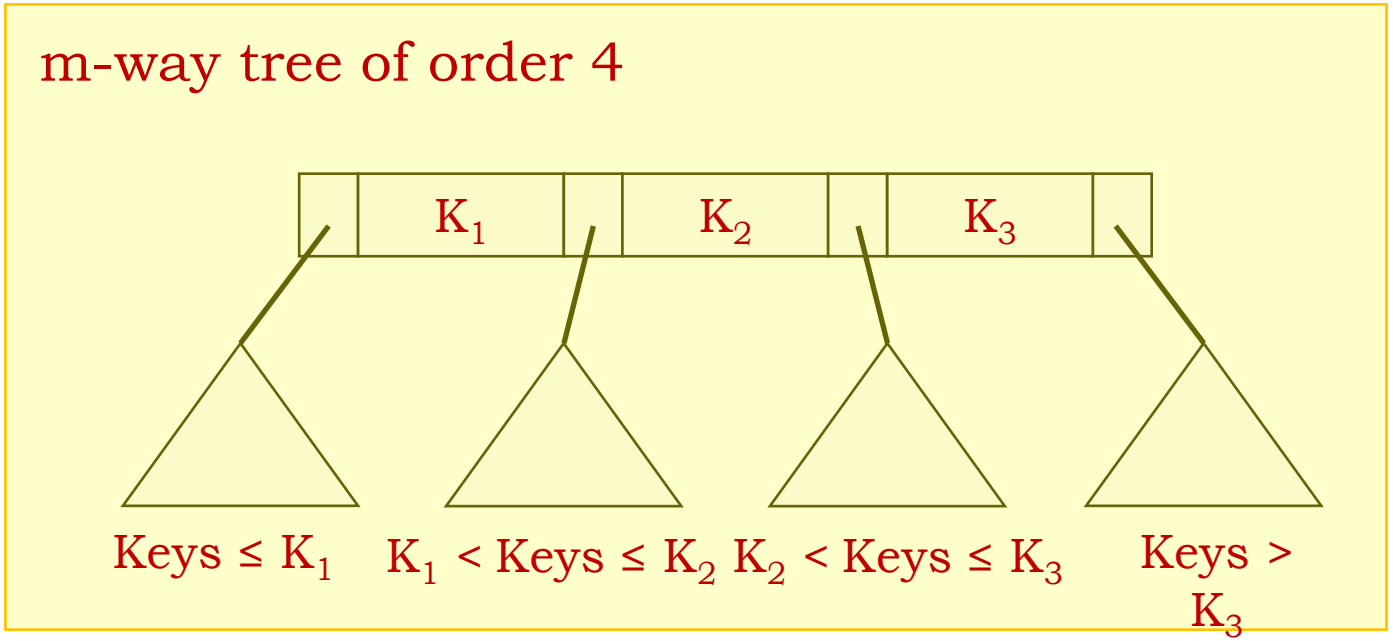


m-Way Tree

Properties:

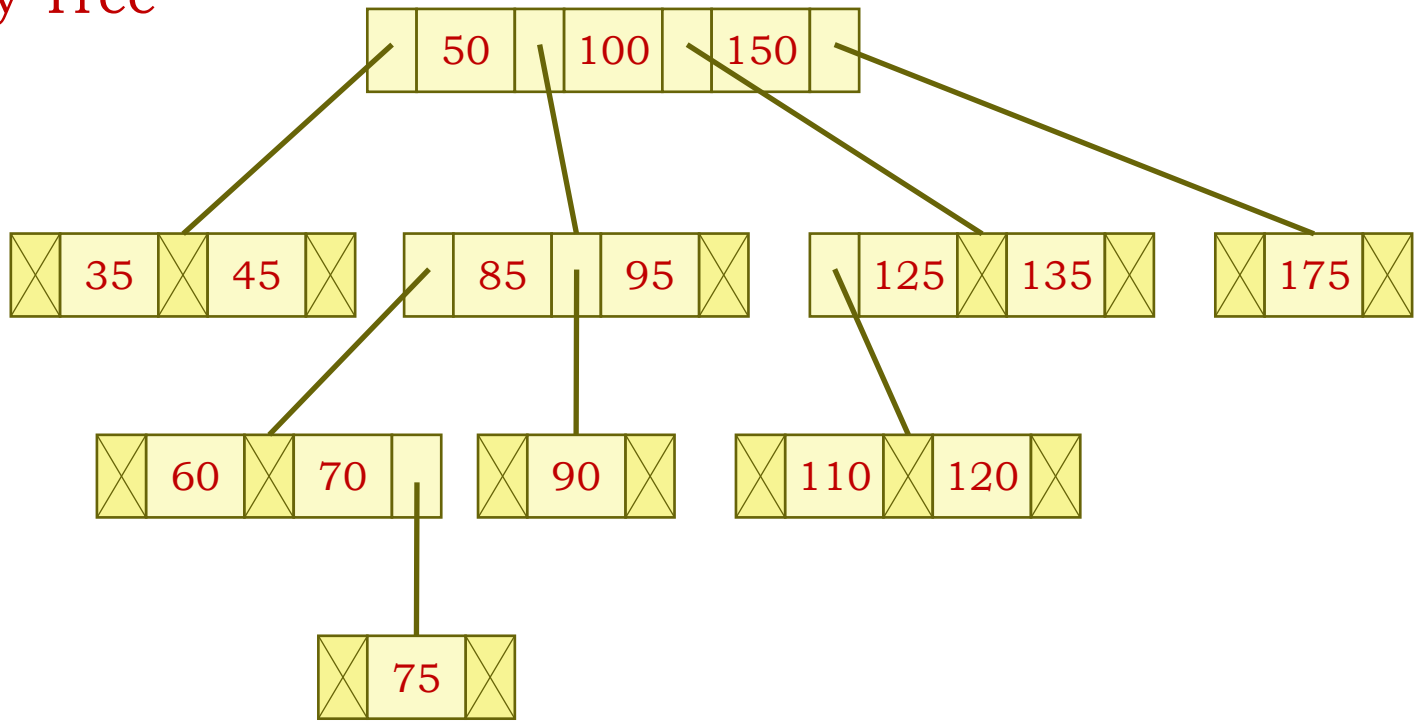
1. Each node has 0 to m sub-trees
2. A node with $k < m$ sub-trees contains k sub-tree pointers, some of which may be $null$, and $k - 1$ data entries (keys)
3. The key values in the first sub-tree are all less than the key value in the first entry: the key values in the other sub-trees are all greater than or equal to the key value in their parent entry
4. The keys of the data entries are ordered $key_1 \leq key_2 \leq \cdots \leq key_k$
5. All sub-trees are themselves multiway trees

m-Way Tree


m-way tree of order 4
K1
K2
K3
Keys ≤ K1
K1 < Keys ≤ K2
K2 < Keys ≤ K3
Keys > K3

m-Way Tree

Four-way Tree



m-Way Tree

- $m - way$ search tree has the same structure as the binary search tree:
  - sub-trees to the left of an entry contain data with keys that are less than the entry's key, and
  - sub-trees to the right of an entry contain data with keys that are greater than or equal to the entry's key


Balance Tree
(B-Tree)

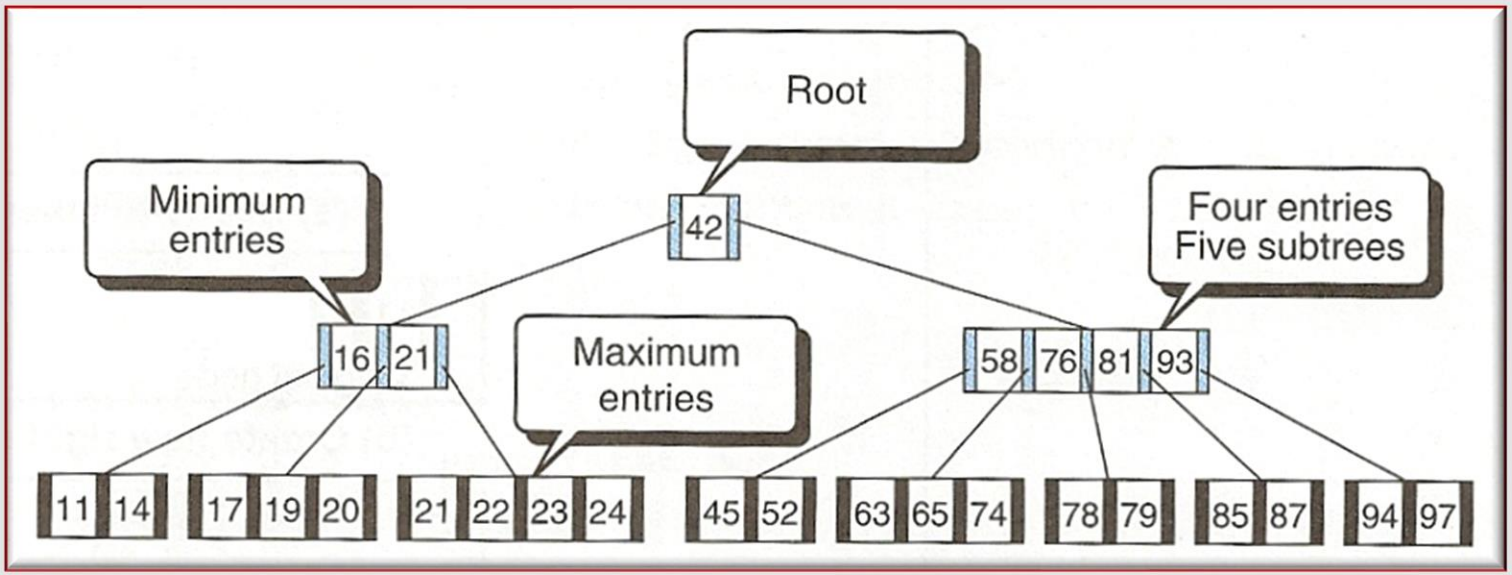
B-Tree

- A B-tree is an m-way search tree with the following properties:
  1. The root is either a leaf or it has $2..m$ sub-trees.
  2. All internal nodes have at least $\lceil m/2 \rceil$ non-null sub-trees and at most m non-null sub-trees.
  3. All leaf nodes are at the same level, that is, the tree is perfectly balanced.
  4. A leaf node has at least $\lceil m/2 \rceil - 1$ and at most $m - 1$ entries (keys).

B-Tree

Order	Number of sub-trees	
	Minimum	Maximum
3	2	3
4	2	4
5	3	5
6	3	6
…	…	…
m	$\lceil m/2 \rceil$	m

B-Tree


Root
Minimum entries
42
Four entries
Five subtrees
16
21
Maximum entries
58
76
81
93
11
14
17
19
20
21
22
23
24
45
52
63
65
74
78
79
85
87
94
97

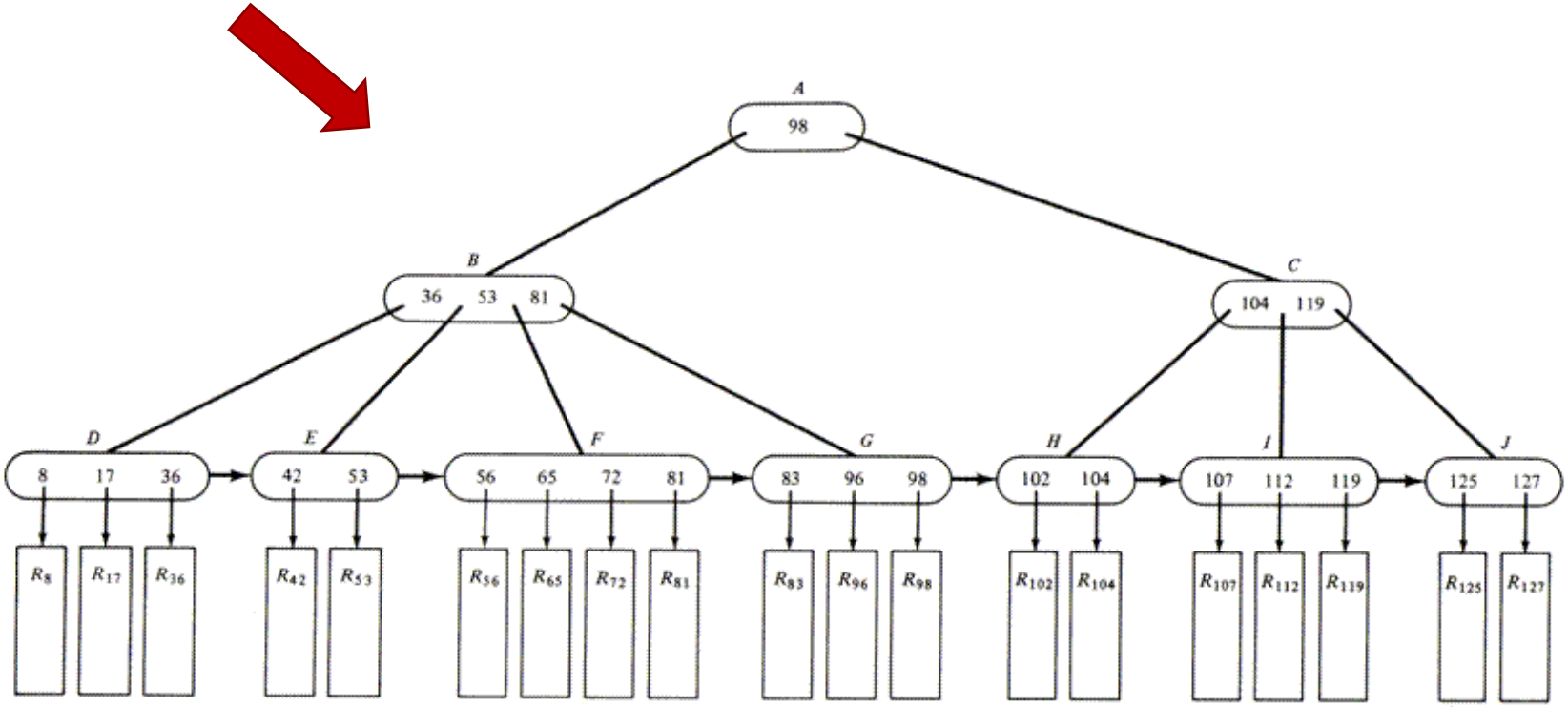

B*-Tree

B*-Tree

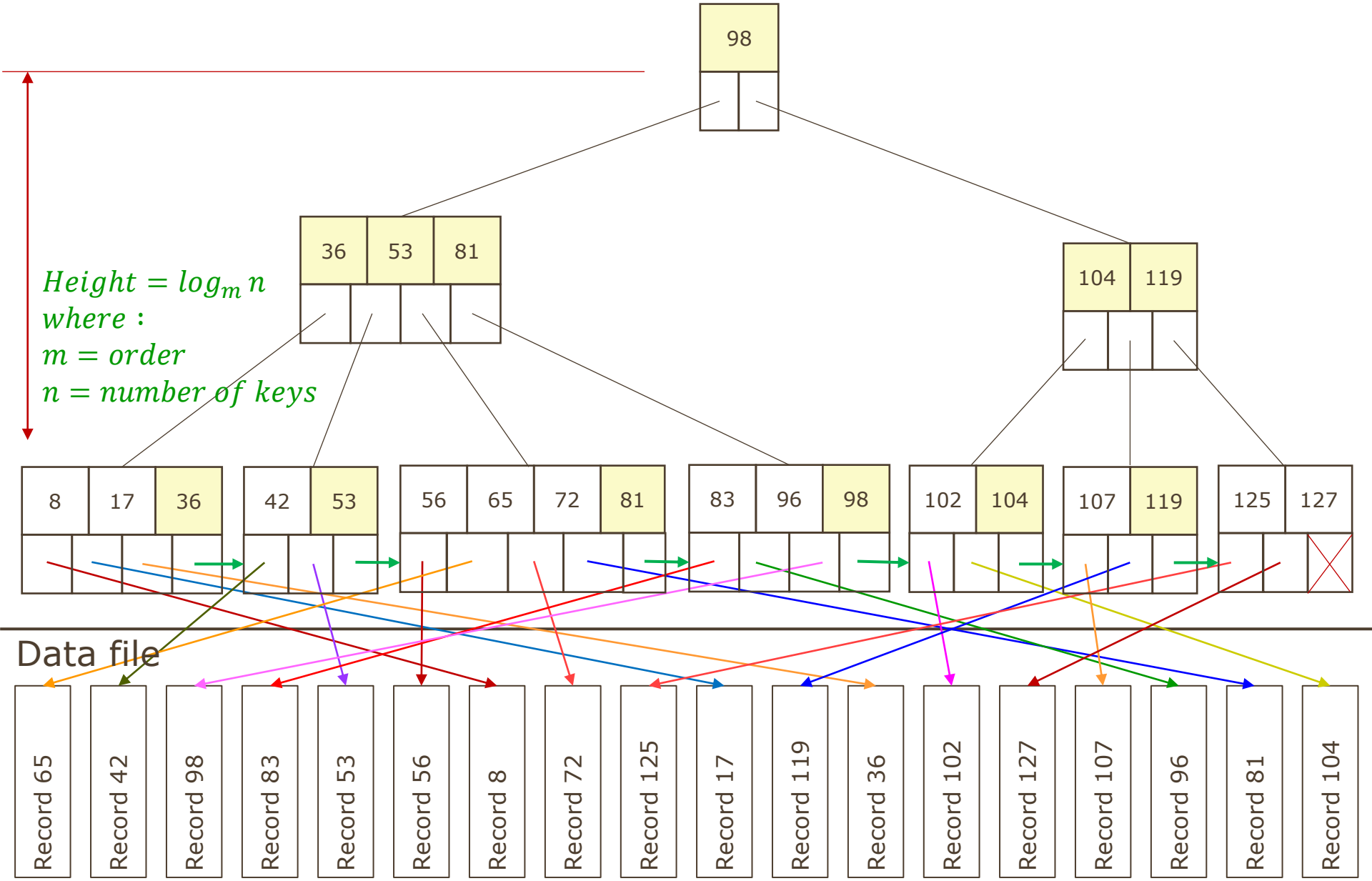
- A $B^* - tree$ index takes the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length.
- The leave nodes are linked together to allow horizontal scan of the nodes at the leave level of the tree.
- B*-trees are commonly used in database systems to implement indexes, allowing for fast retrieval of data based on a key value.

B*-Tree

$B^* - tree$


A
98
B
36 53 81
C
104 119
D
8 17 36
E
42 53
F
56 65 72 81
G
83 96 98
H
102 104
I
107 112 119
J
125 127
R_8 R_{17} R_{36} R_{42} R_{53} R_{56} R_{65} R_{72} R_{81} R_{83} R_{96} R_{98} R_{102} R_{104} R_{107} R_{112} R_{119} R_{125} R_{127}

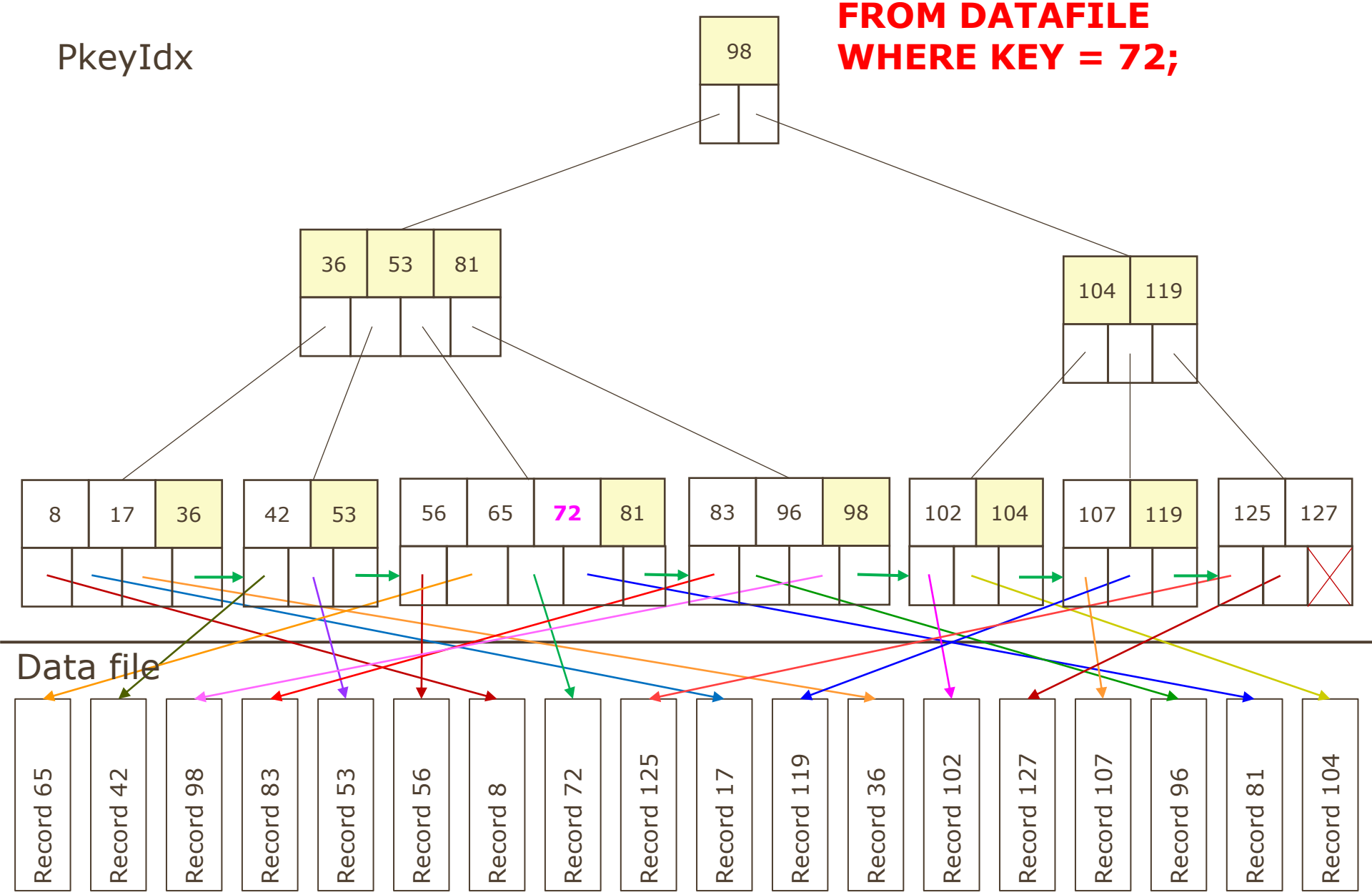
Index (B*-Tree)


98
36 53 81
104 119
$Height = log_m n$
$where:$
$m = order$
$n = number\ of\ keys$
8 17 36
42 53
56 65 72 81
83 96 98
102 104
107 119
125 127
Data file
Record 65
Record 42
Record 98
Record 83
Record 53
Record 56
Record 8
Record 72
Record 125
Record 17
Record 119
Record 36
Record 102
Record 127
Record 107
Record 96
Record 81
Record 104

Index (B*-Tree)

PkeyIdx

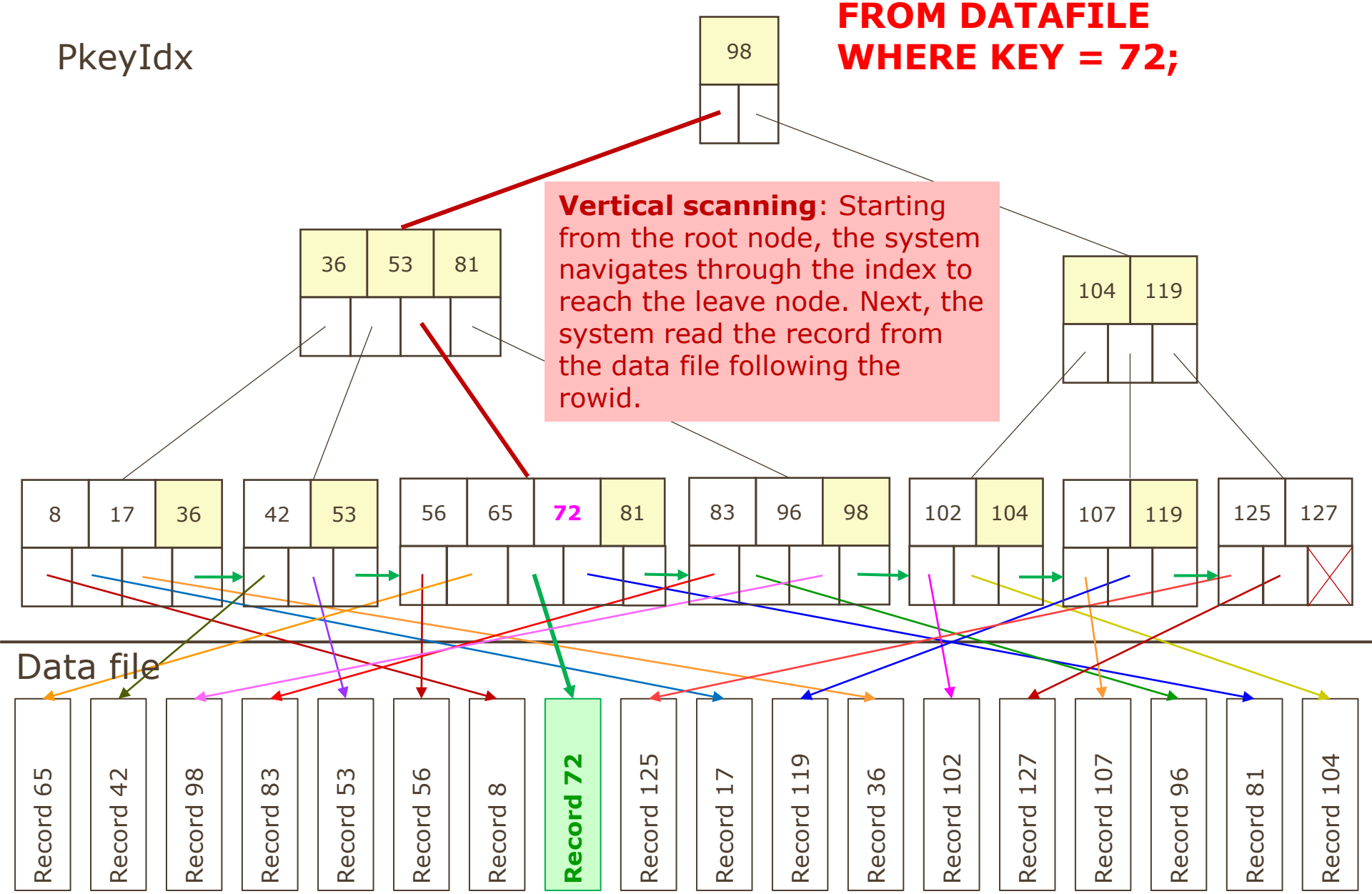
**SELECT *
FROM DATAFILE
WHERE KEY = 72;**



Index (B*-Tree)

PkeyIdx

**SELECT *
FROM DATAFILE
WHERE KEY = 72;**

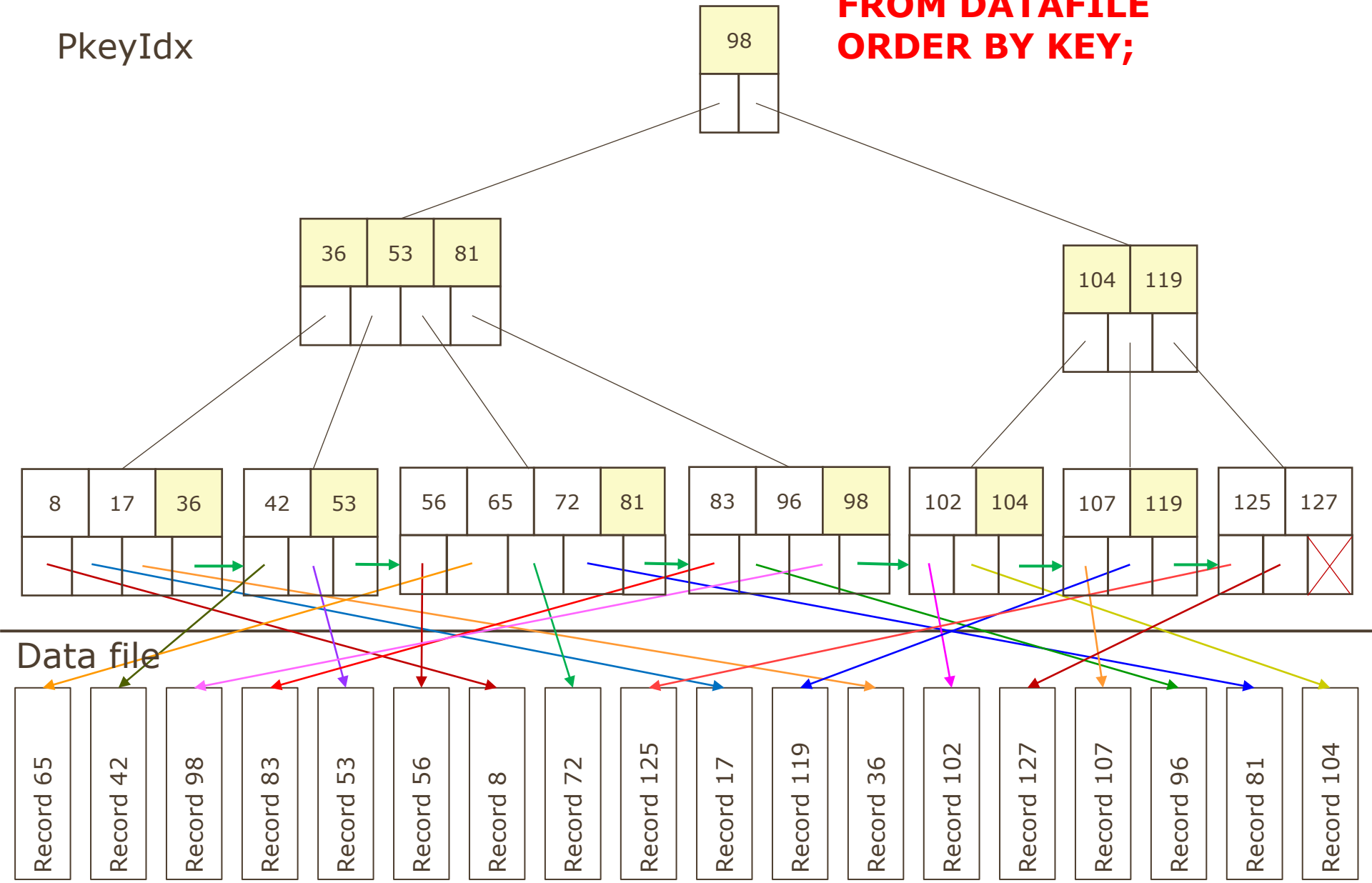


Vertical scanning: Starting from the root node, the system navigates through the index to reach the leave node. Next, the system read the record from the data file following the rowid.

Index (B*-Tree)

PkeyIdx

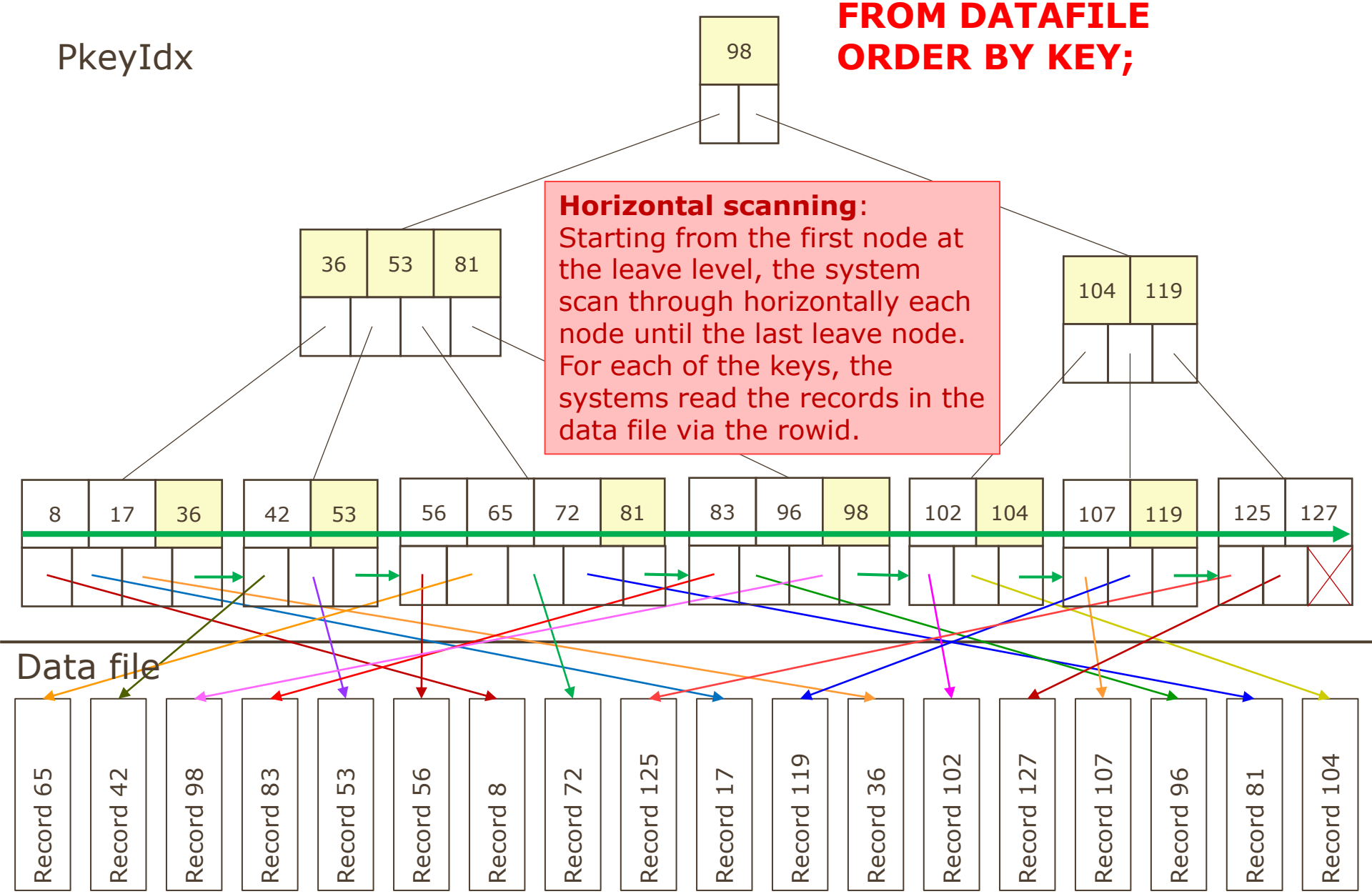
**SELECT *
FROM DATAFILE
ORDER BY KEY;**



Index (B*-Tree)

PkeyIdx

**SELECT *
FROM DATAFILE
ORDER BY KEY;**

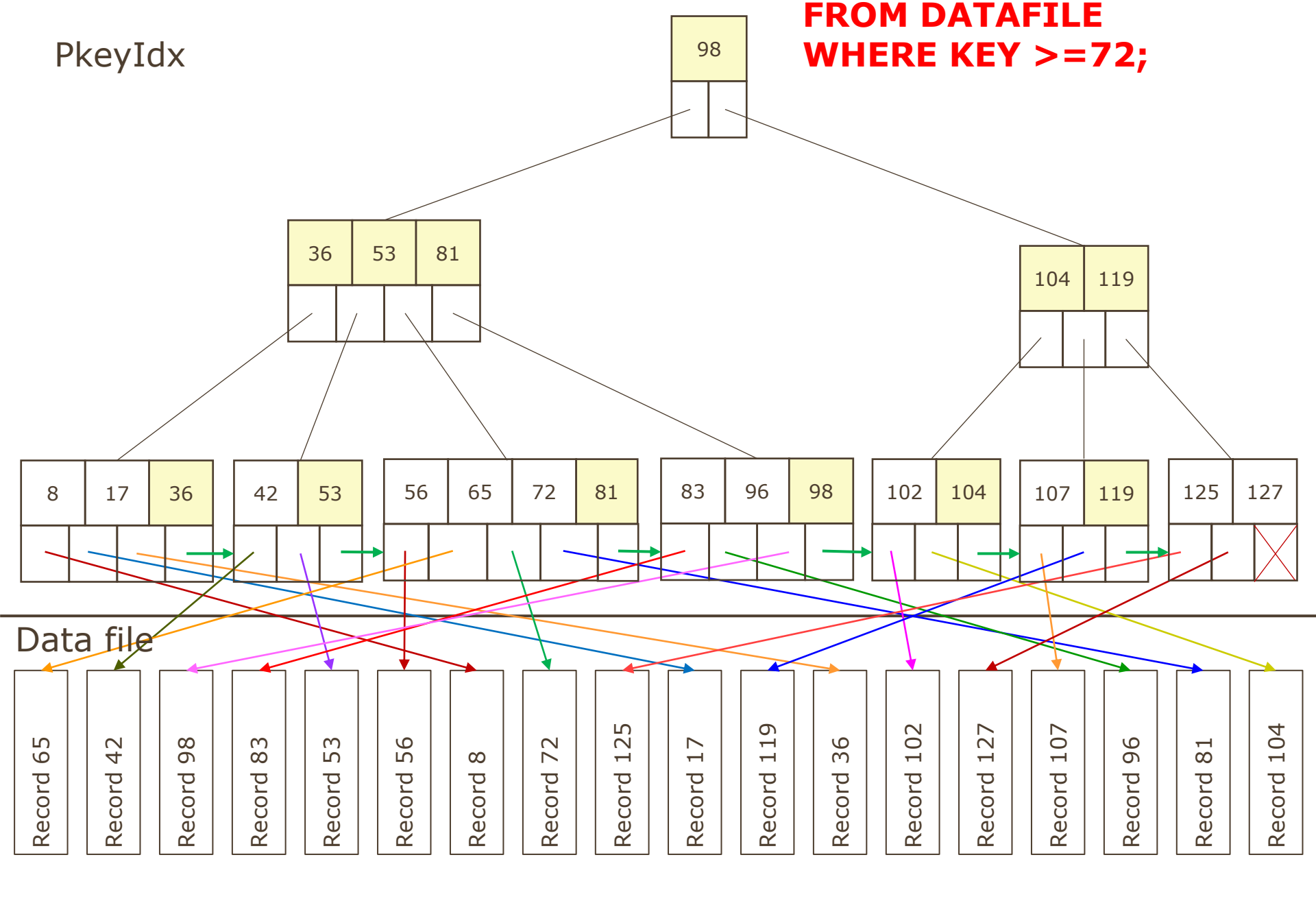


Horizontal scanning:
Starting from the first node at the leave level, the system scan through horizontally each node until the last leave node. For each of the keys, the systems read the records in the data file via the rowid.

Index (B*-Tree)

PkeyIdx

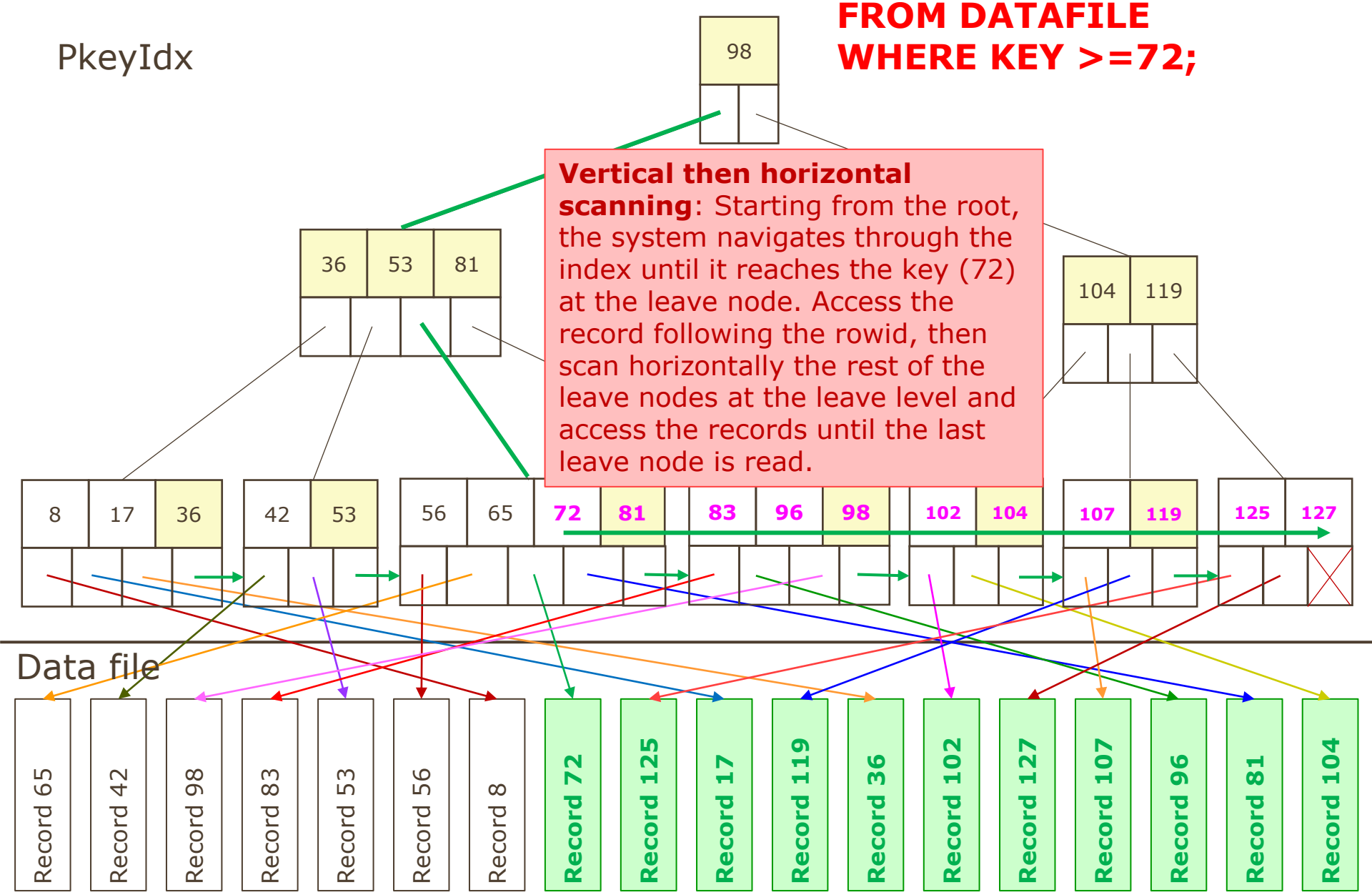
**SELECT *
FROM DATAFILE
WHERE KEY >=72;**



Index (B*-Tree)

PkeyIdx

**SELECT *
FROM DATAFILE
WHERE KEY >=72;**



Vertical then horizontal scanning: Starting from the root, the system navigates through the index until it reaches the key (72) at the leave node. Access the record following the rowid, then scan horizontally the rest of the leave nodes at the leave level and access the records until the last leave node is read.