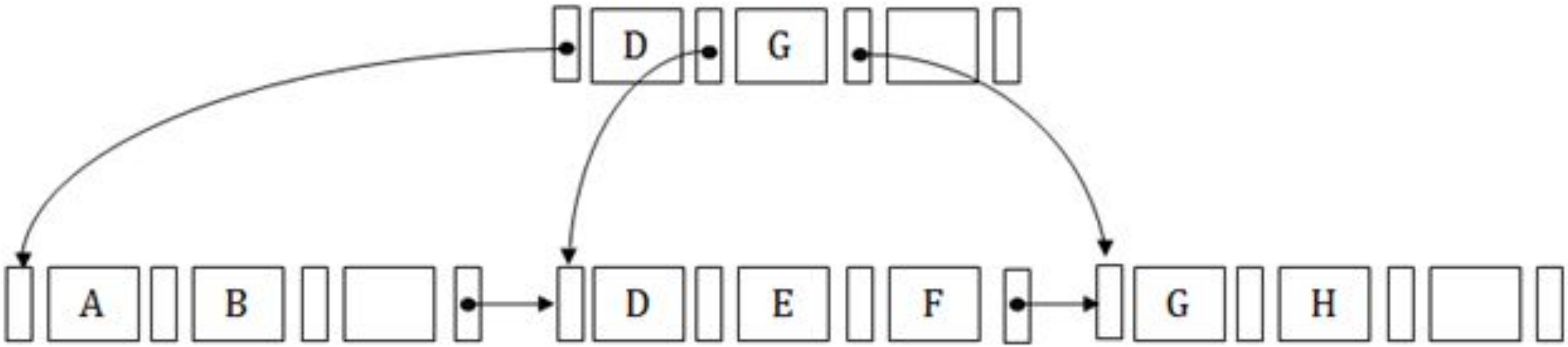
- The B+-tree index structure is the most widely used of several index structures that maintain their efficiency despite insertion and deletion of data.
- A B+-tree index takes the form of a balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length.

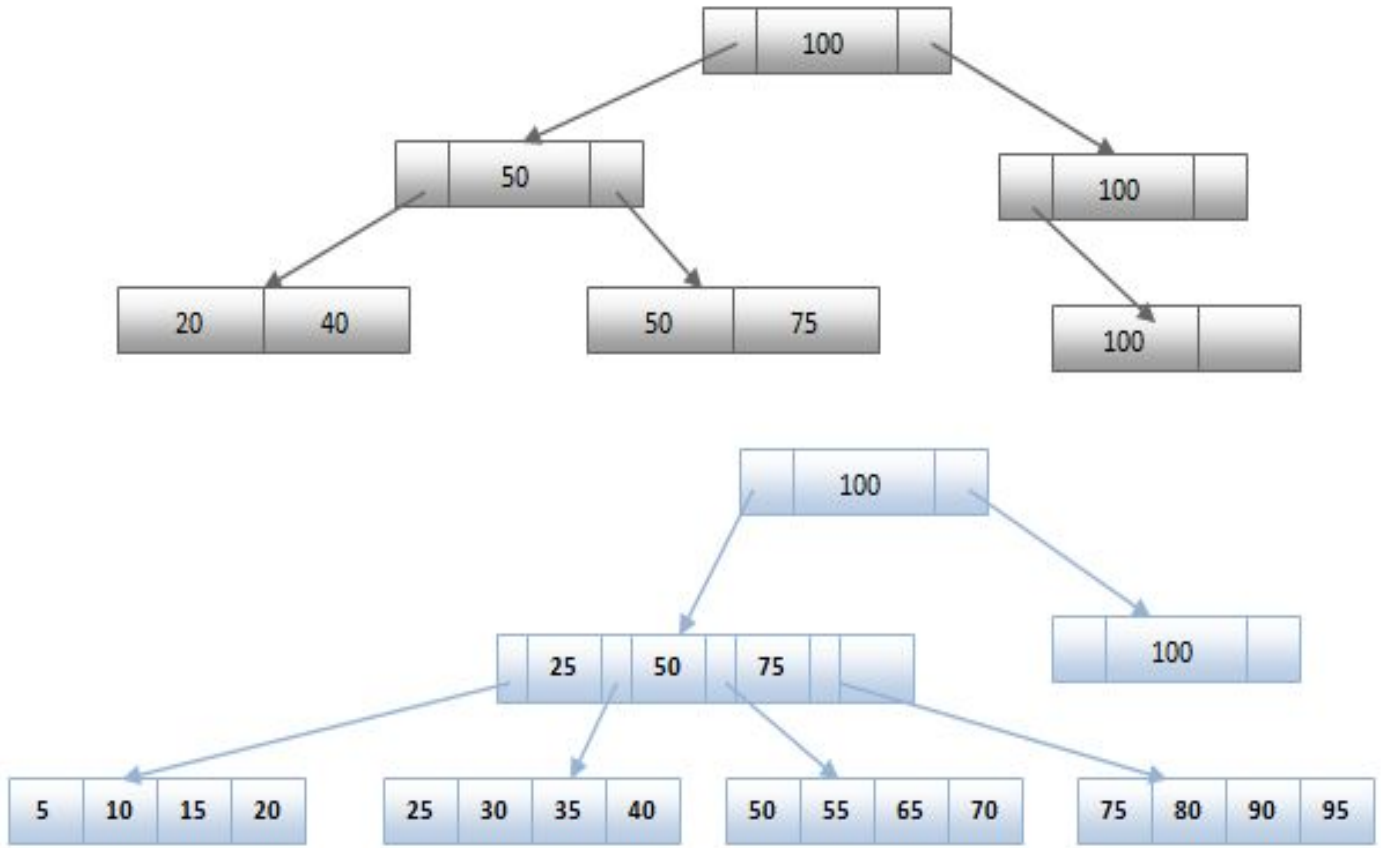
What is B+tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.
- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node

Simple B+tree



With n =3


100
50
100
20
40
50
75
100
100
25
50
75
100
5
10
15
20
25
30
35
40
50
55
65
70
75
80
90
95

B+ tree

- Leaves are used to store data records.
- It stored in the internal nodes of the Tree.
- If a target key value is less than the internal node, then the point just to its left side is followed.
- If a target key value is greater than or equal to the internal node, then the point just to its right side is followed.

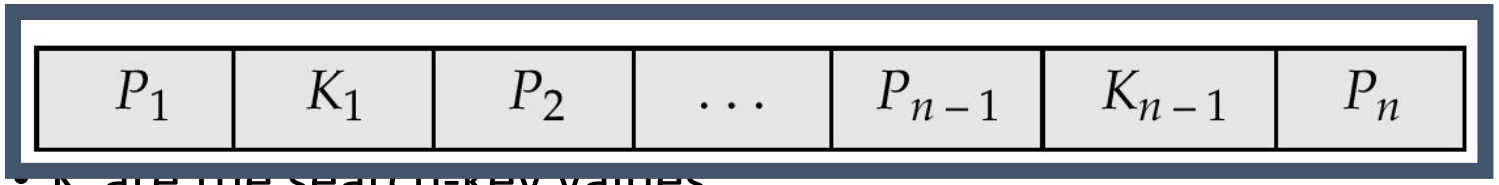
B^+-Tree Index Files (Cont.) properties/structure

- B^+-tree is a rooted tree satisfying the following properties:
  - All paths from root to leaf are of the same length
  - Each node that is not a root or a leaf has between $[n/2]$ to n children.
  - A leaf node has between $[(n–1)/2]$ and $n–1$ values

  - If n = 4 then
  - Min keys in leaf node is (n-1)/2 = 3/2 = 1
  - Max key in leaf node is (n-1) = 3
  - Internal node has maximum n/2 to n children that is 2 to 4 childern

B^+-Tree Node Structure

- Typical node



  - K_i are the search-key values
  - P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered

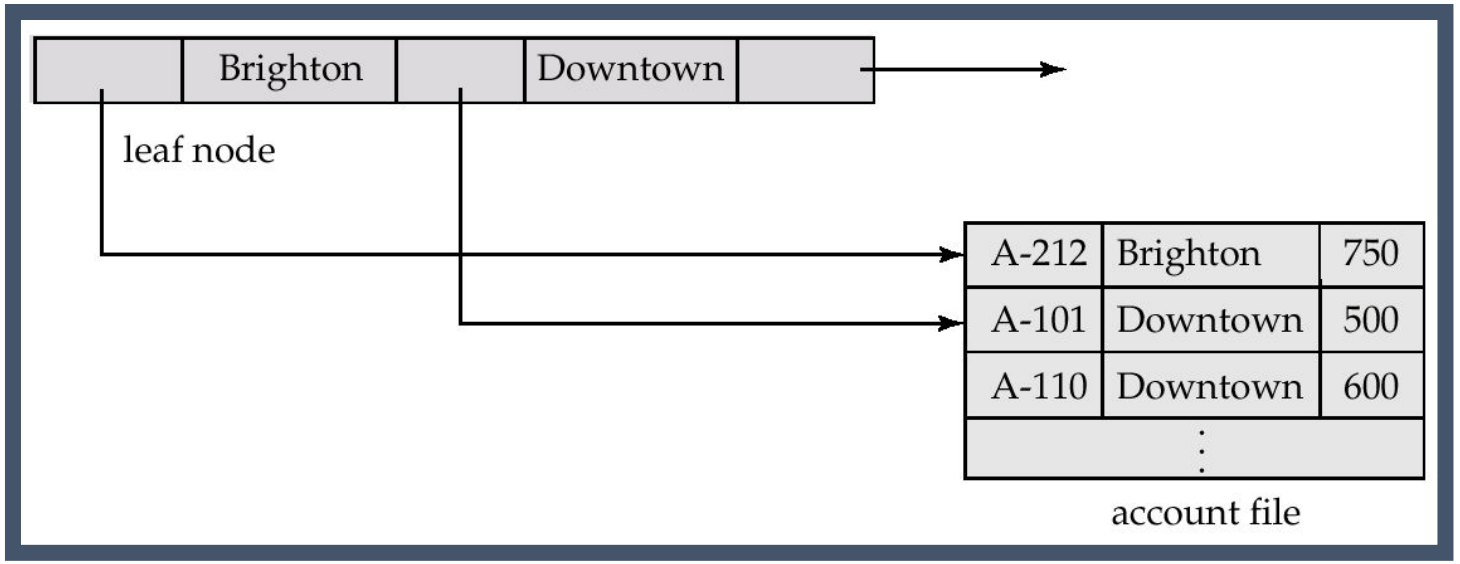
$$K_1 < K_2 < K_3 < \ldots < K_{n-1}$$

(key are in sorted order)

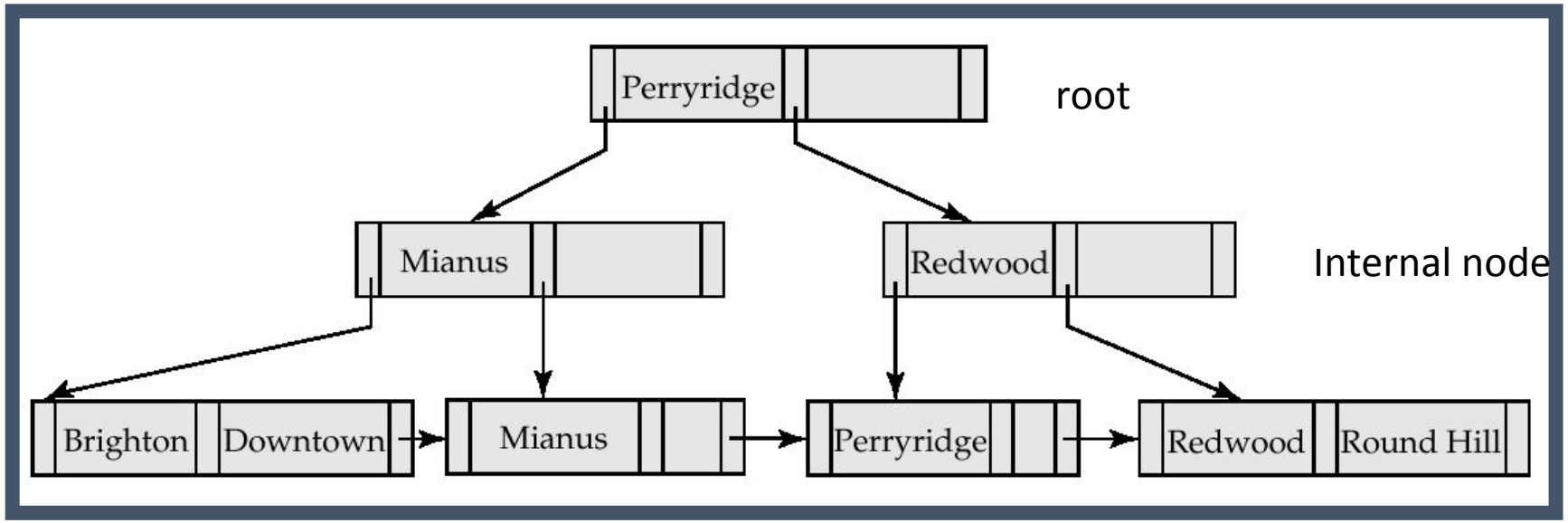
Leaf Nodes in B^+-Trees

Properties of a leaf node:

- I L_j are leaf nodes and $i < j$, L_i's search-key values are less than L_j's search-key values
- P_n points to next leaf node in search-key order



Example of a B^+-tree



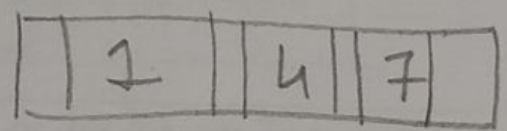
B^+-tree for *account* file (*n* = 3)

e.G for explanation

- Construct the B+ tree for given key
  - 1,4,7,10,17,21,31,25 order is 4

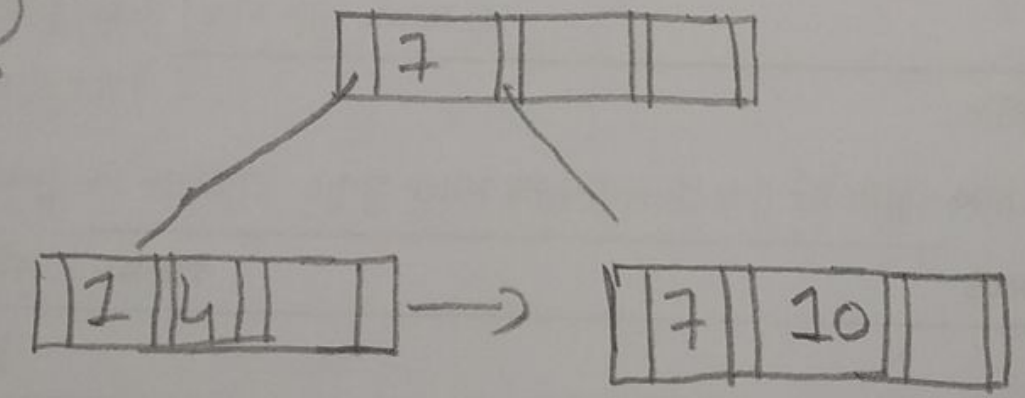
  - Order(m) =4
  - Max children = 4
  - Min children = m/2 =2
  - Max keys = m -1 =3
  - Min keys = [m/2] -1 =1

(1)



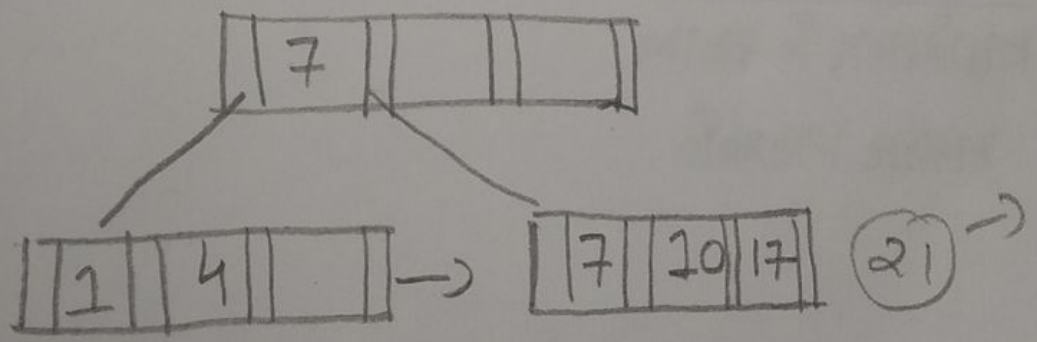
10 → cannot insert 10 as max key = 3
so split the node & create link from upper level node.

(2)



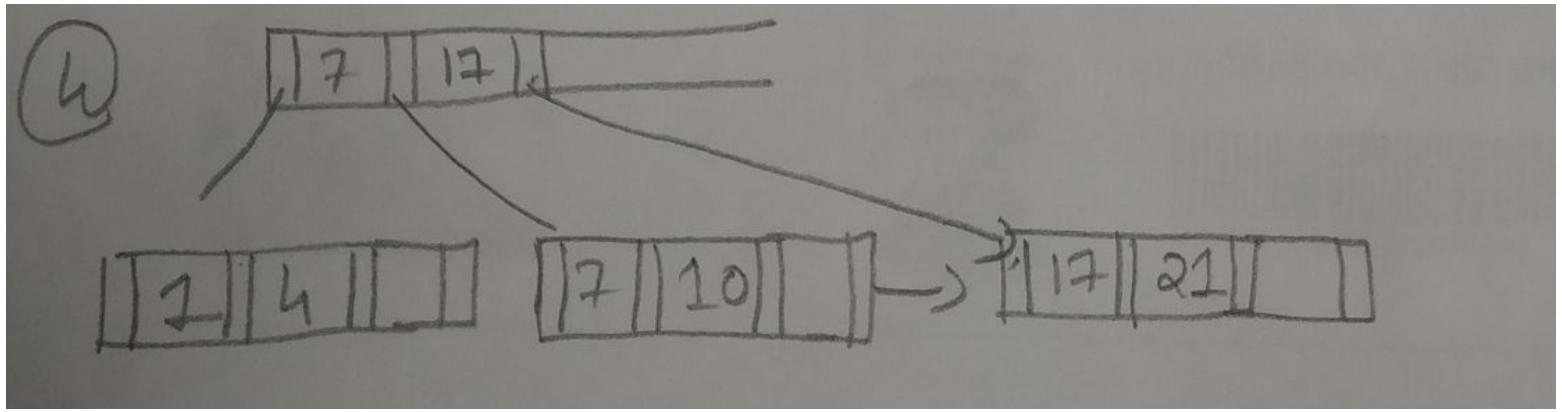
Inserting 10

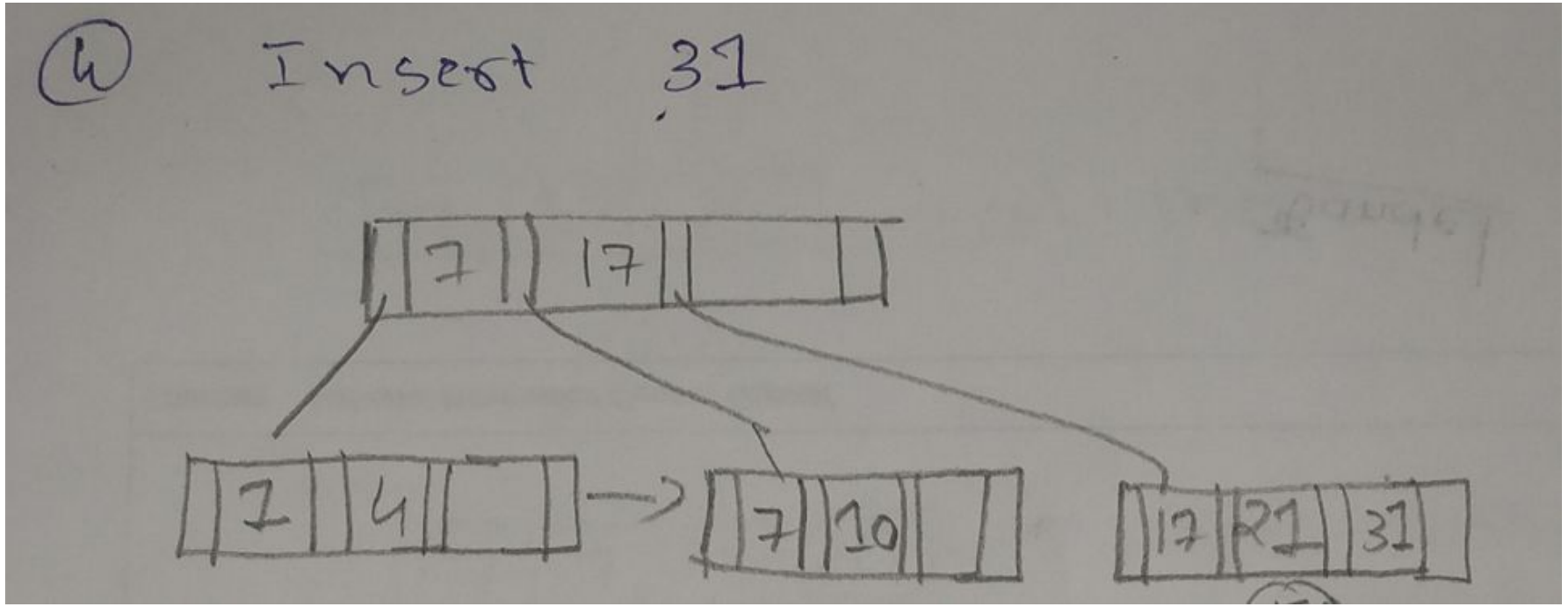
(3) Insert 17, 21

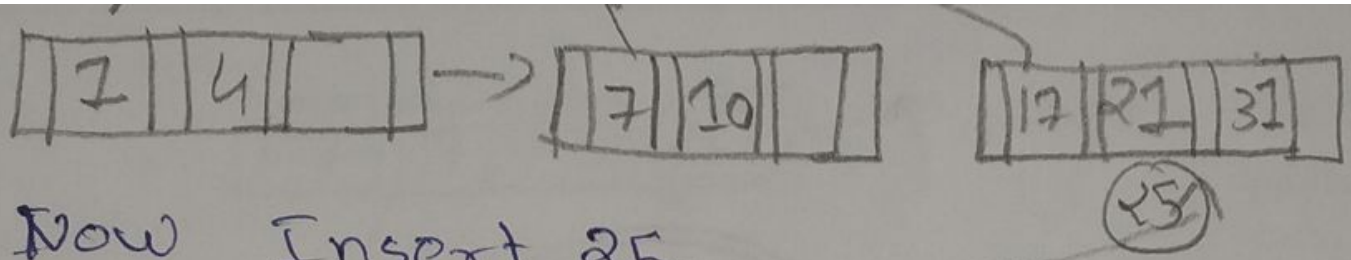


21 key dont have space has maximum keys are 3. so split the Node.

leaf

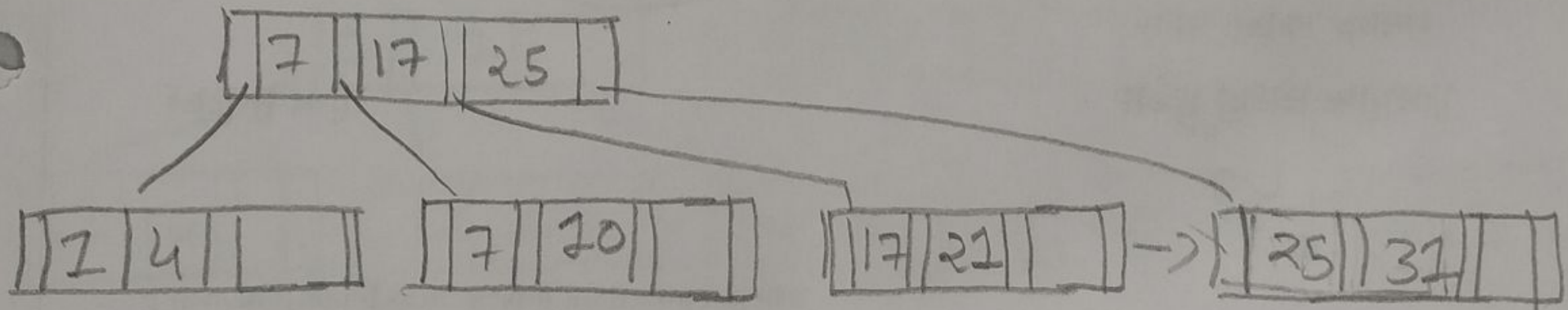

4
7
17
2
4
7
10
17
21


4
Insert 31
7
17
2
4
7
10
17
21
31

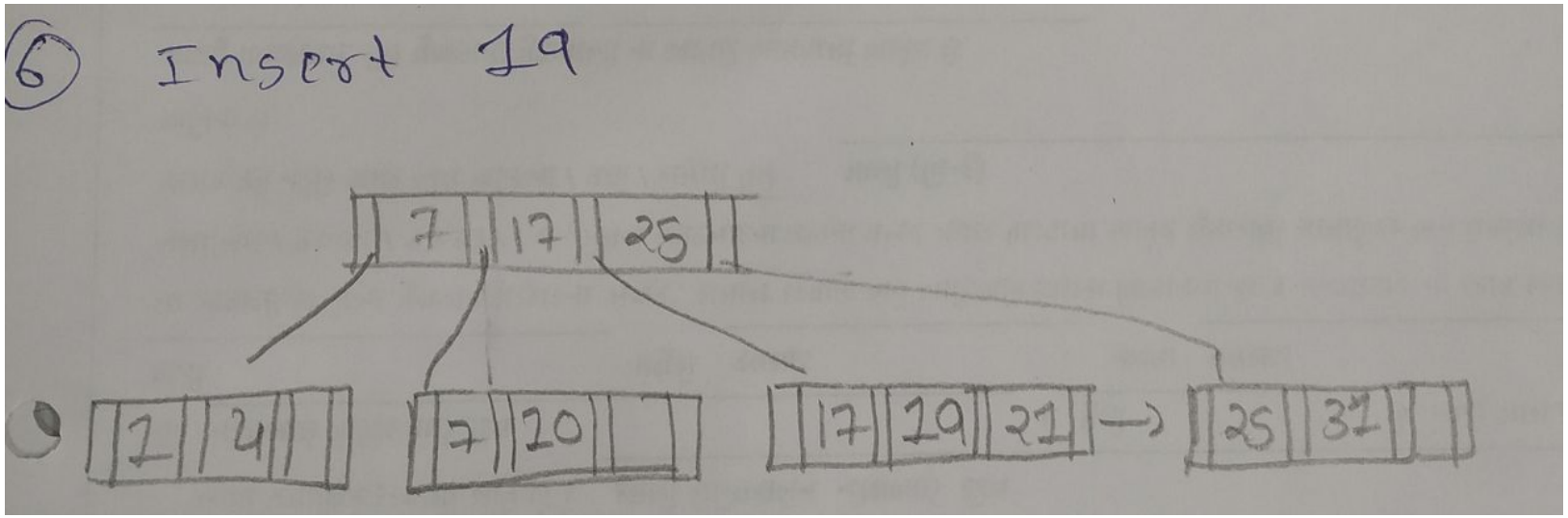


(5) Now Insert 25

→ 25 must be between 21 & 31 but can not insert as max keys are 3 which are filled. So split the Node.



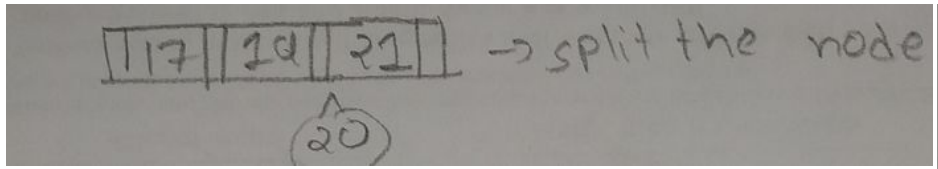
(6) Insert 19

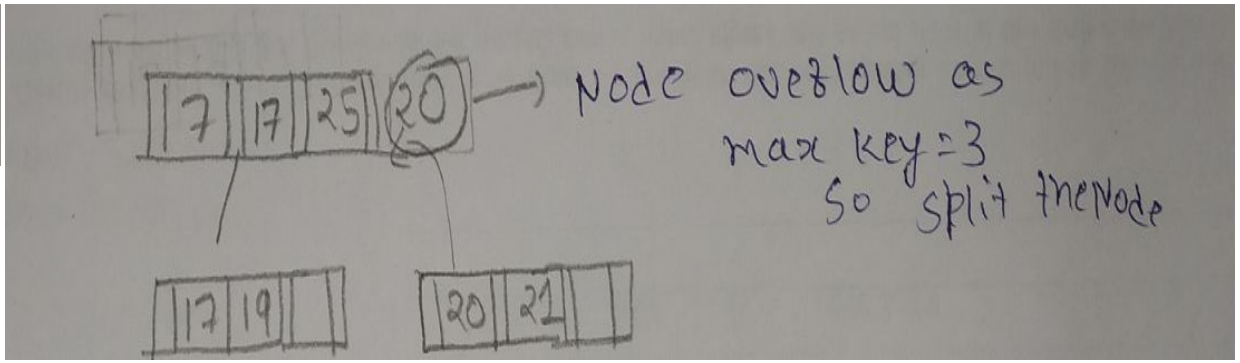


(7) Insert 20.

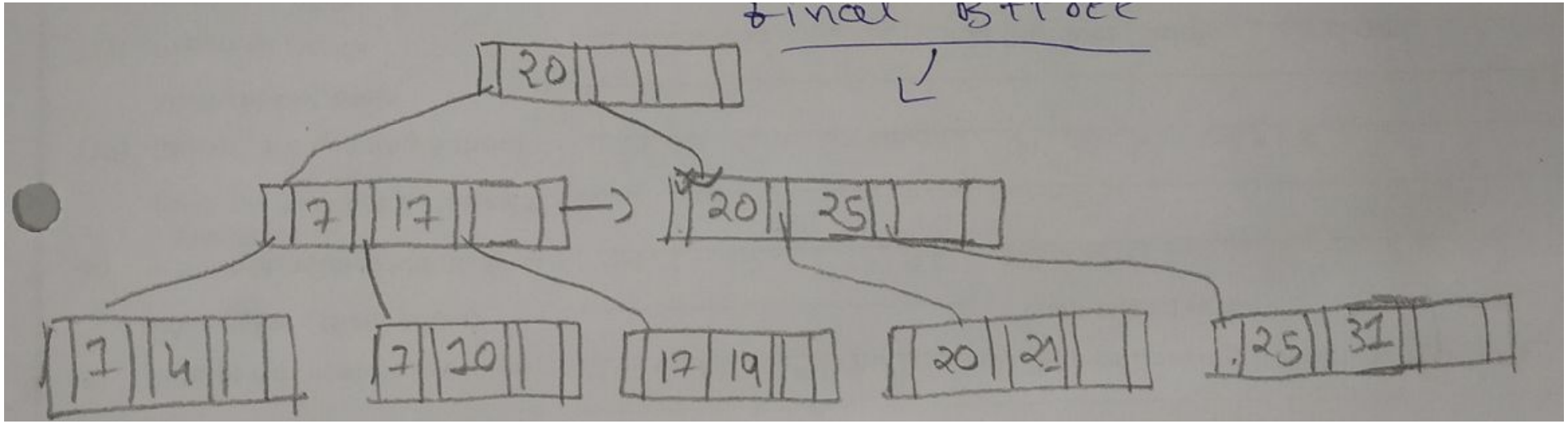
-> ~~In~~ Leaf node is overflow so leaf node is splited.

-> Upper level node is also overflow so split Interal node & create one upper level


17
19
21
20
→ split the node


7
17
25
20
→ Node overflow as
max key = 3
So split the Node
17
19
20
21

Final tree


20
7
17
20
25
1
4
7
10
17
19
20
21
25
31