Design and Analysis of Algorithms

Unit-2

B-Tree

B-Tree

Definition

A B-tree T is a rooted tree having the following properties:

1. Every node x has the following attributes:

   a. x.n, the number of keys currently stored in node x,

   b. the x.n keys themselves, $x.key_1$, $x.key_2$, ……,$x.key_{x.n}$, stored in non-decreasing order, so that $x.key_1 \le x.key_2 \le \ldots\ldots\ldots \le x.key_{x.\,n}$,

   c. x.leaf , a boolean value that is TRUE if x is a leaf and FALSE if x is an internal node.

2. Each internal node x also contains x.n+1 pointers $x.c_1$; $x.c_2$, ……. ,$x.c_{x.\,n+1}$ to its children. Leaf nodes have no children, and so their c_i attributes are undefined.

3. The keys $x.key_i$ separate the ranges of keys stored in each subtree: if k_i is any key stored in the subtree with root $x.c_i$, then

$k_1 \le x.key_1 \le k_2 \le x.key_2 \le \ldots\ldots\ldots\ldots\ldots\ldots \le X.key_{x.\,n} \le k_{x.\,n+1}$.

B-Tree

4. All leaves have the same depth, which is the tree's height h.

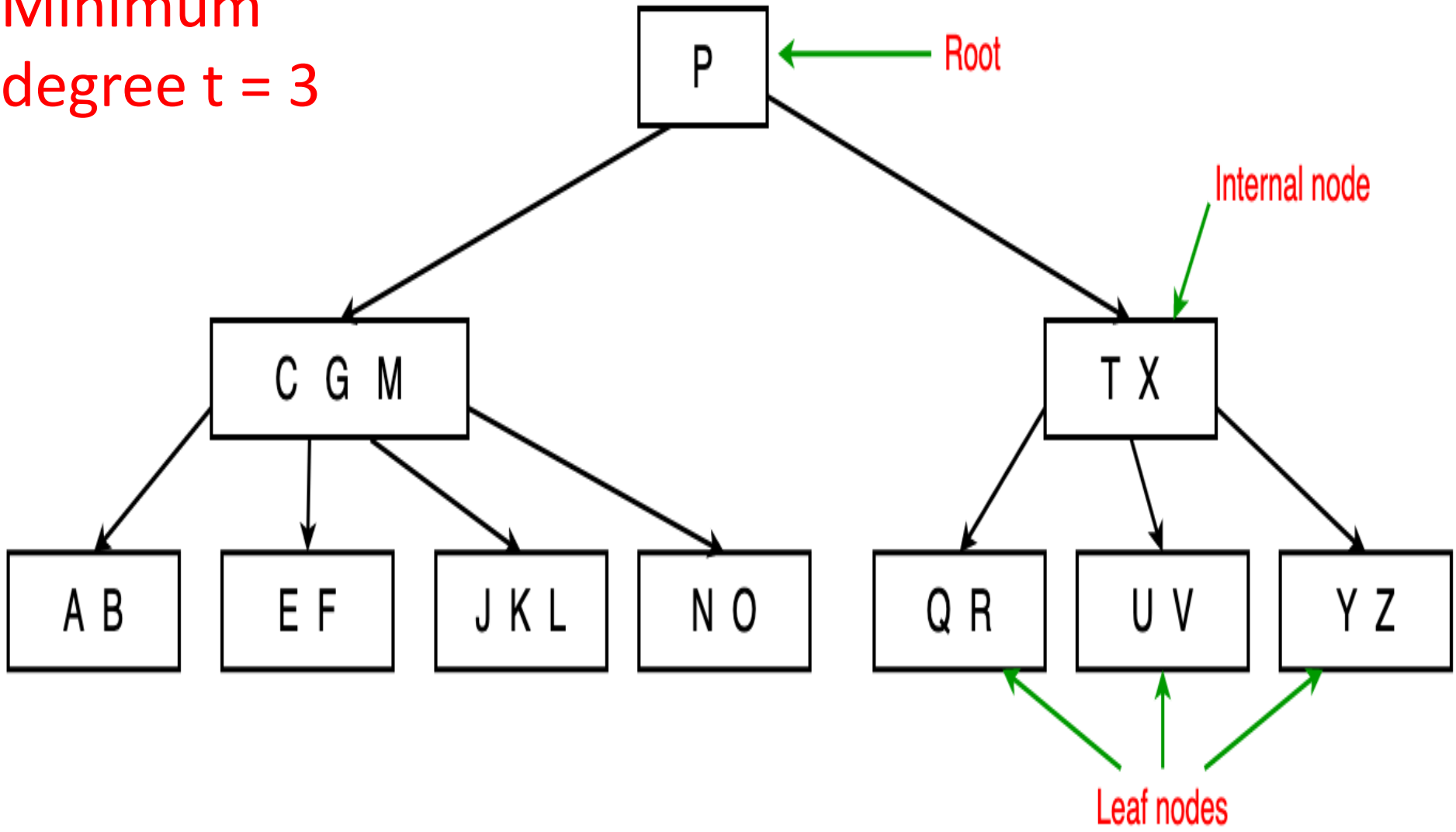
5. Nodes have lower and upper bounds on the number of keys they can contain. We express these bounds in terms of a fixed integer $t \geq 2$ called the minimum degree of the B-tree:

    a. Every node other than the root must have at least t-1 keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.

    b. Every node may contain at most 2t -1 keys. Therefore, an internal node may have at most 2t children. We say that a node is full if it contains exactly2t -1 keys.

B-Tree

Minimum degree t = 3



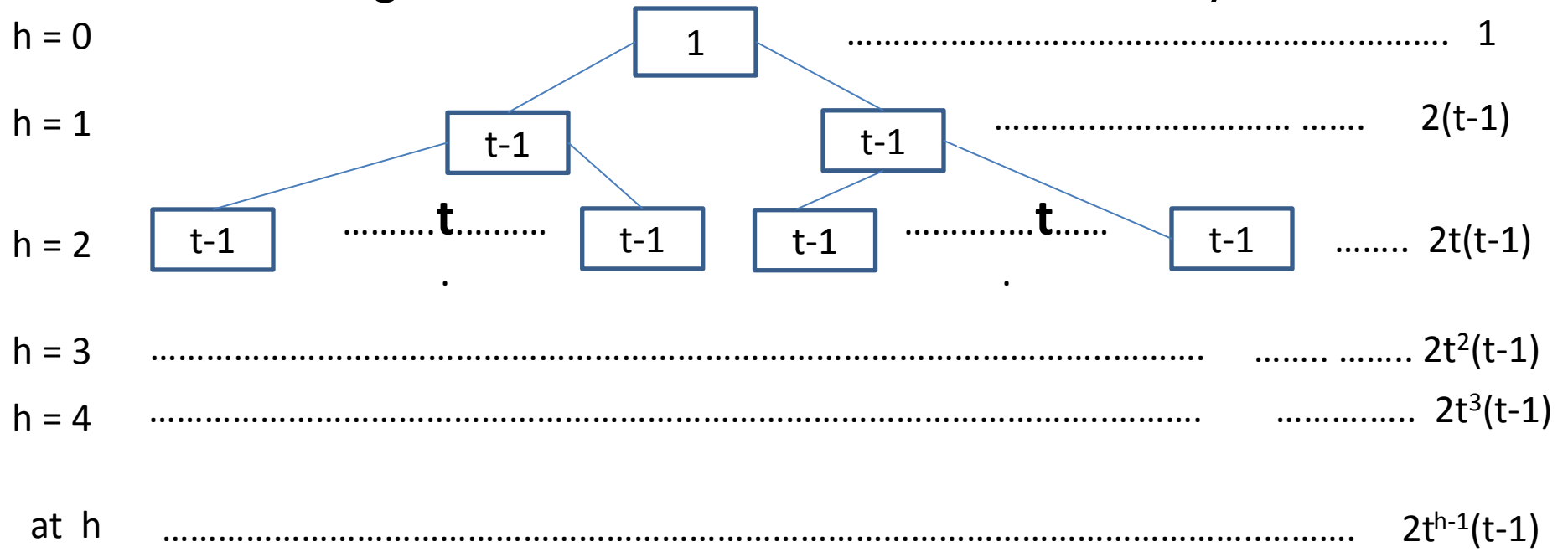
B-Tree

Theorem: If n ≥ 1, then for any n-key B-tree T of height h and minimum degree t ≥ 2,

$$h \leq \log_t \frac{(n+1)}{2}$$

Proof: To prove this, first we will find the minimum number of keys in the B-tree with minimum degree t and height h.

Consider following structure of B-tree with minimum keys:-



B-Tree

Therefore, the minimum number of the keys in B-tree with minimum degree t and height h

$$= 1 + 2(t-1) + 2t(t-1) + 2t^2(t-1) + 2t^3(t-1) + \ldots\ldots + 2t^{h-1}(t-1)$$

$$= 1 + 2(t-1)\left(1 + t + t^2 + t^3 + \ldots\ldots\ldots + t^{h-1}\right)$$

$$= 1 + 2(t-1)\frac{(th-1)}{(t-1)}$$

$$= 1 + 2(t^h-1)$$

$$= 1 + 2t^h-2$$

$$= 2t^h-1$$

Now, since the number of keys in given B-tree is n, therefore

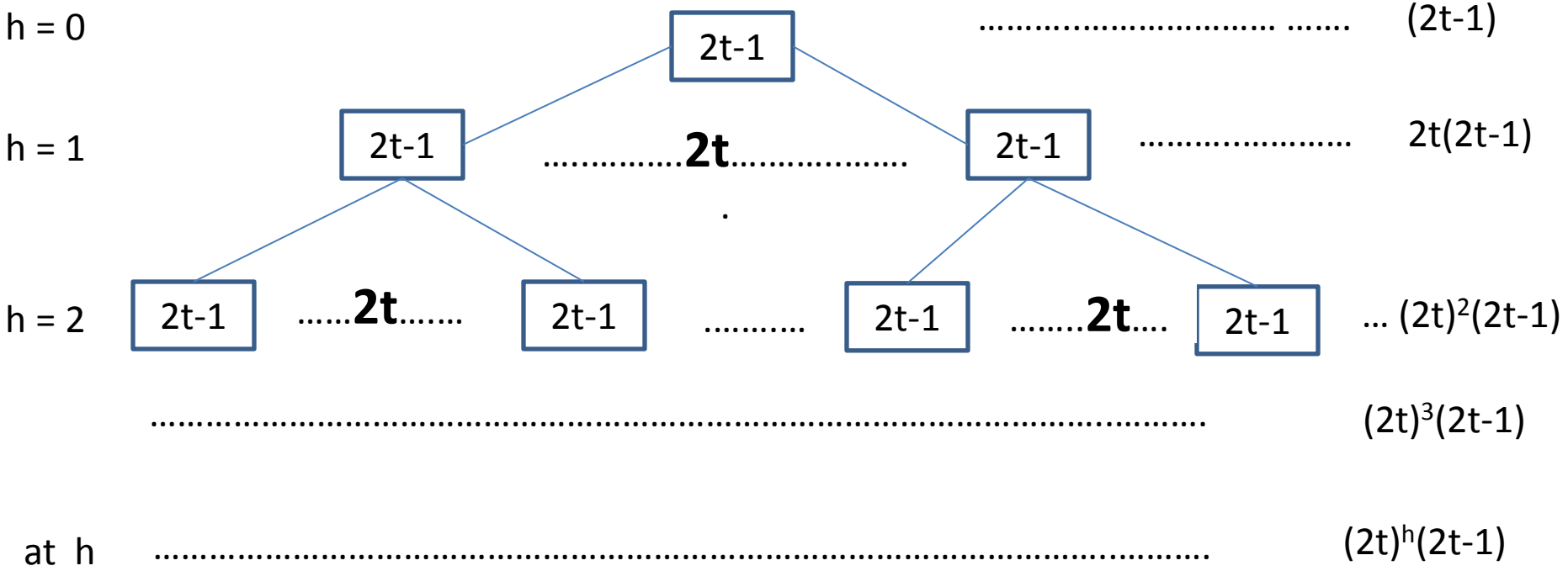
$2t^h-1 \le n \Rightarrow t^h \le \frac{(n+1)}{2} \Rightarrow h \le \log_t \frac{(n+1)}{2}$

It is proved.

B-Tree

Question: As a function of the minimum degree t , what is the maximum number of keys that can be stored in a B-tree of height h?

Solution: Consider following structure of B-tree with maximum keys:-



B-Tree

Therefore maximum number of keys in B-tree with minimum degree t and height h

$= (2t-1) + 2t(2t-1) + (2t)^2(2t-1) + (2t)^3(2t-1) + \ldots\ldots\ldots\ldots + (2t)^h(2t-1)$

$= (2t-1)(1+2t + (2t)^2 + (2t)^3 + \ldots\ldots\ldots\ldots\ldots + (2t)^h)$

$= (2t-1)\frac{((2t)^{h+1}-1)}{(2t-1)}$

$= (2t)^{h+1}-1$

B-Tree

Note:

The simplest B-tree occurs when t = 2. Every internal node then has either 2, 3, or 4 children, and we have a 2-3-4 tree. In practice, however, much larger values of t yield B-trees with smaller height.

B-Tree of order m

B-tree of order m is a tree which satisfies the following properties:

1. Every node has at most m children.
2. Every non-leaf node (except root) has at least ⌈m/2⌉ child nodes.
3. The root has at least two children if it is not a leaf node.
4. A non-leaf node with k children contains k − 1 keys.
5. All leaves appear in the same level and carry no information.

Each internal node's keys act as separation values which divide its subtrees. For example, if an internal node has 3 child nodes (or subtrees) then it must have 2 keys: a1 and a2. All values in the leftmost subtree will be less than a1, all values in the middle subtree will be between a1 and a2, and all values in the rightmost subtree will be greater than a2.

Searching operation in B-Tree

Searching a B-tree is much like searching a binary search tree, except that instead of making a binary, or "two-way," branching decision at each node, we make a multiway branching decision according to the number of the node's children.

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3      i = i + 1
4  if i ≤ x.n and k == x.key_i
5      return (x, i)
6  elseif x.leaf
7      return NIL
8  else DISK-READ(x.c_i)
9      return B-TREE-SEARCH(x.c_i, k)
```

Searching operation in B-Tree

- B-TREE-SEARCH takes as input a pointer to the root node x of a subtree and a key k to be searched for in that subtree.
- The initial call of this algorithm will be

  B-TREE-SEARCH(T.root, k)
- If k is in the B-tree, B-TREE-SEARCH returns the ordered pair (y, i) consisting of a node y and an index i such that $y.key_i = k$. Otherwise, the procedure returns NIL.

B-Tree creation

Example: Create B-tree for the following elements with minimum degree t = 2.

F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E

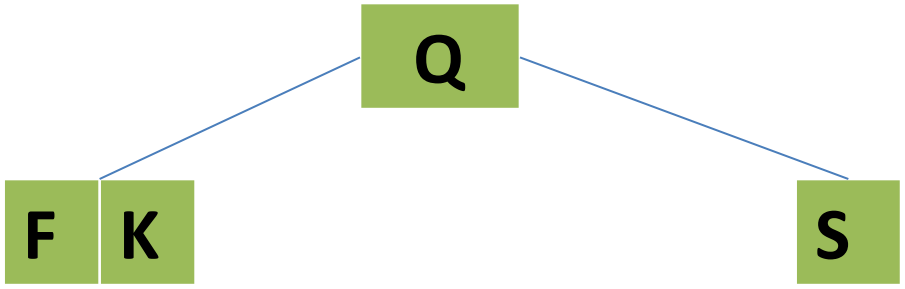
Solution:

Minimum number of keys in a node = t-1 = 1

Maximum number of keys in a node = 2t-1 = 3

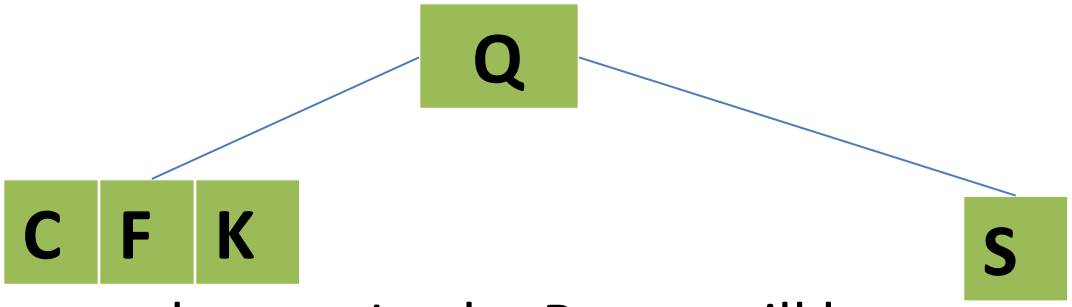
Initial consider 2t-1 elements in the sequence i.e. 3 elements. These are F, S and Q. B-tree for this will be

After inserting next element K, the B-tree will be

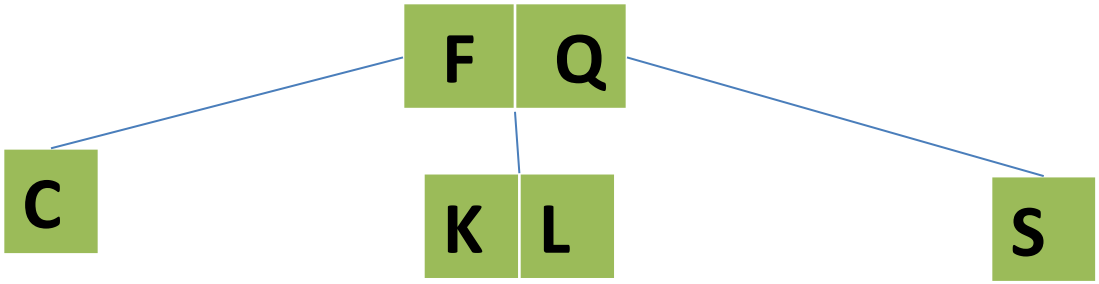


B-Tree

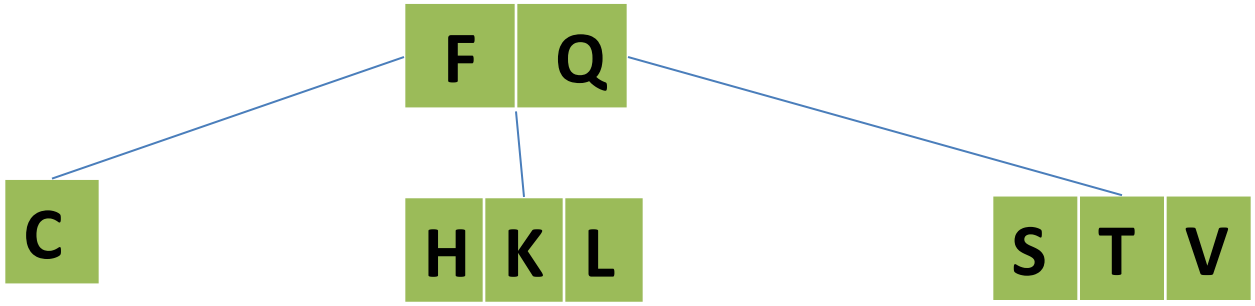
After inserting next element C, the B-tree will be



After inserting next element L , the B-tree will be

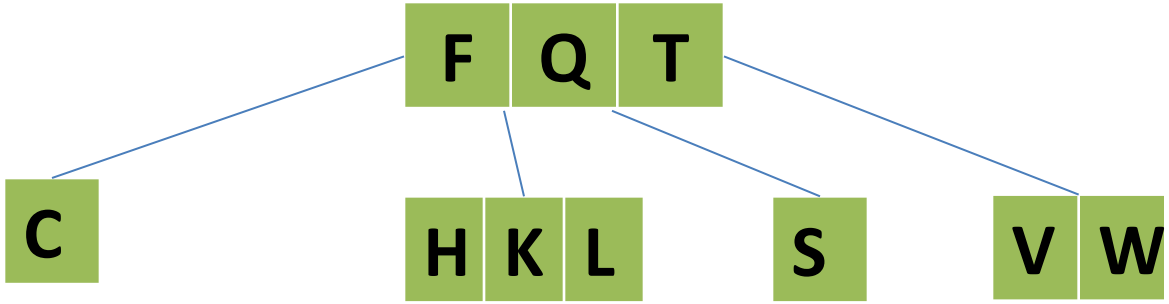


After inserting next element H,T, and V, the B-tree will be

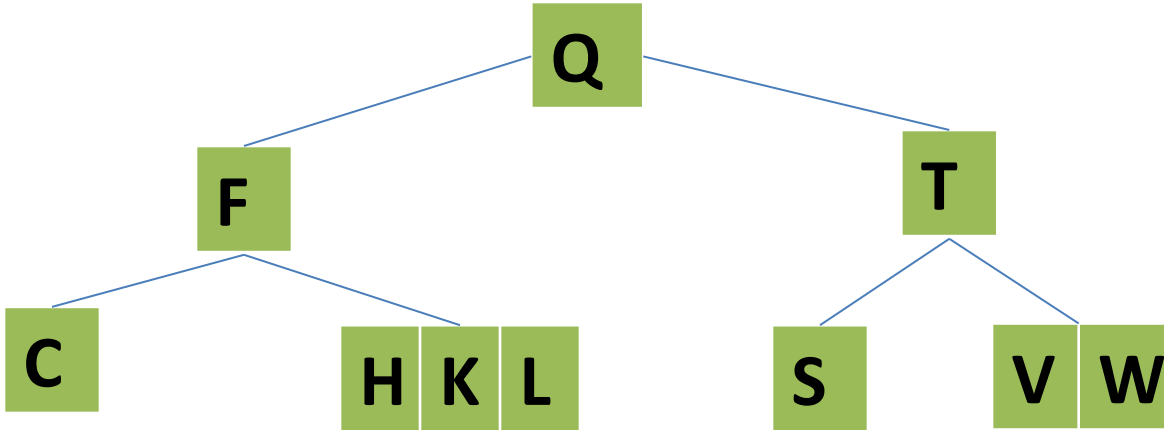


B-Tree

After inserting next element W, the B-tree will be

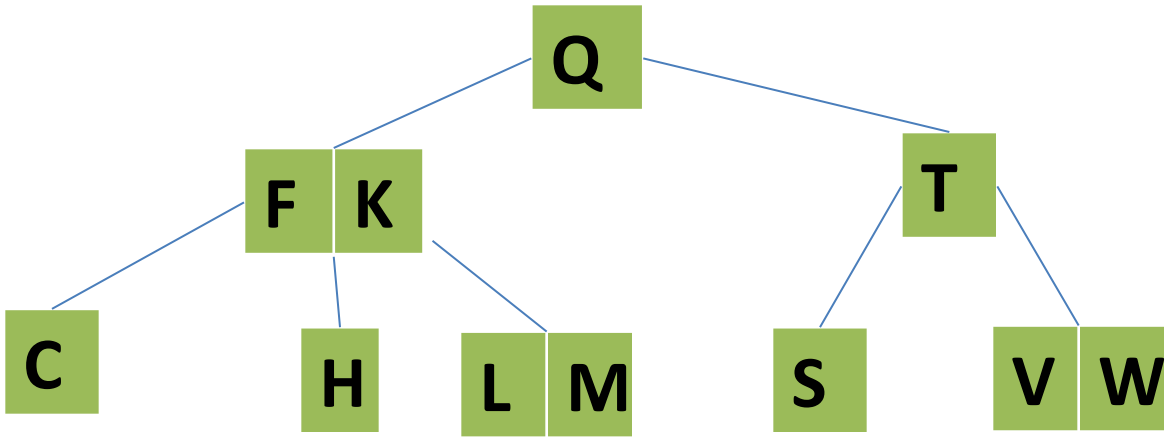


Now, insert element M. Since root node is full, therefore first, we split it.

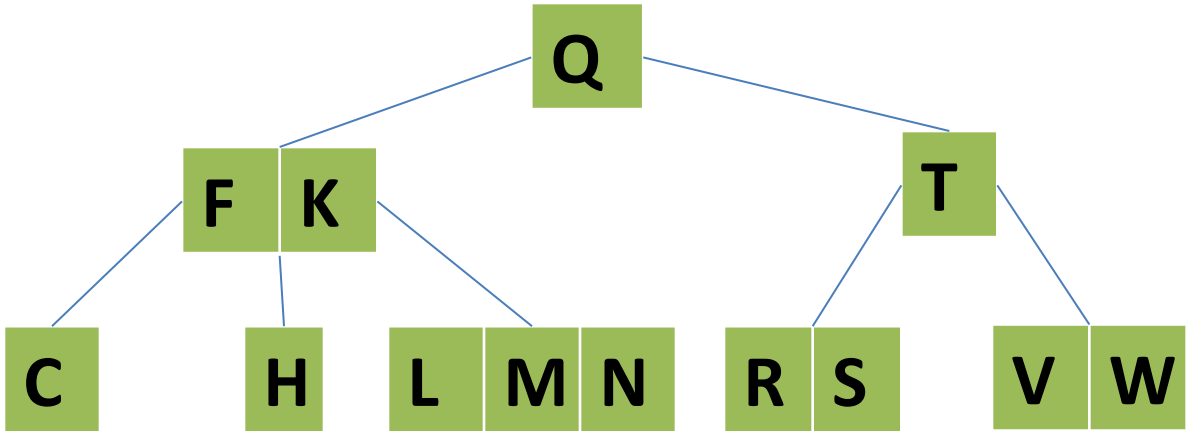


B-Tree

. After splitting and inserting M, B-tree will be

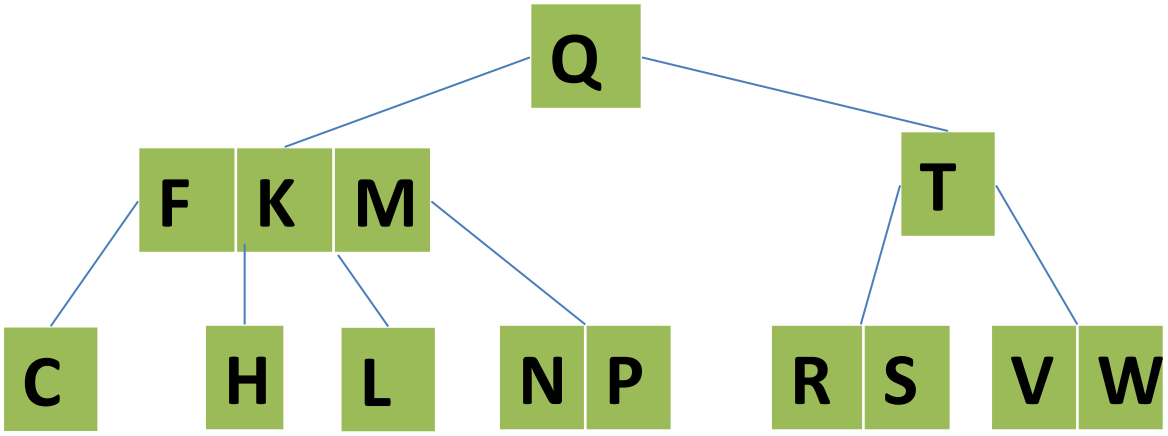


After inserting next element R and N, the B-tree will be

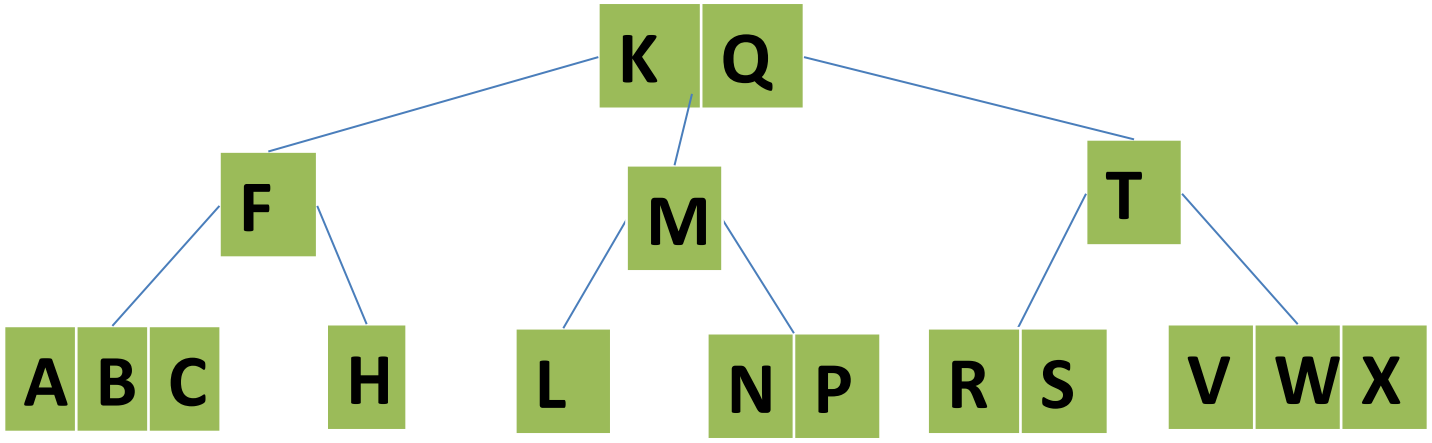


B-Tree

After inserting next element P, the B-tree will be

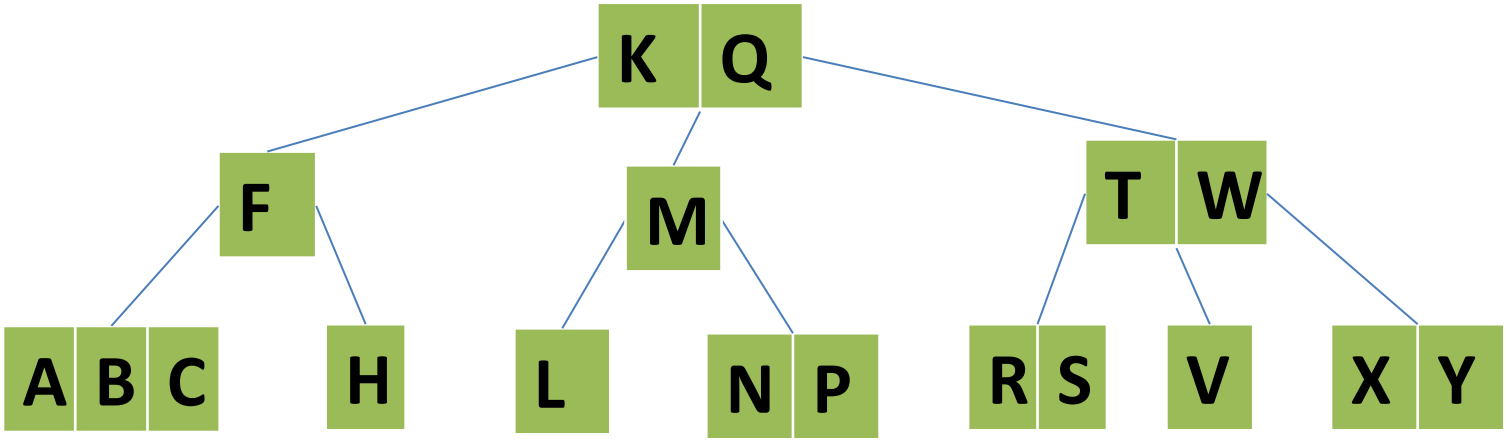


After splitting and inserting next element A, B and X, the B-tree will be

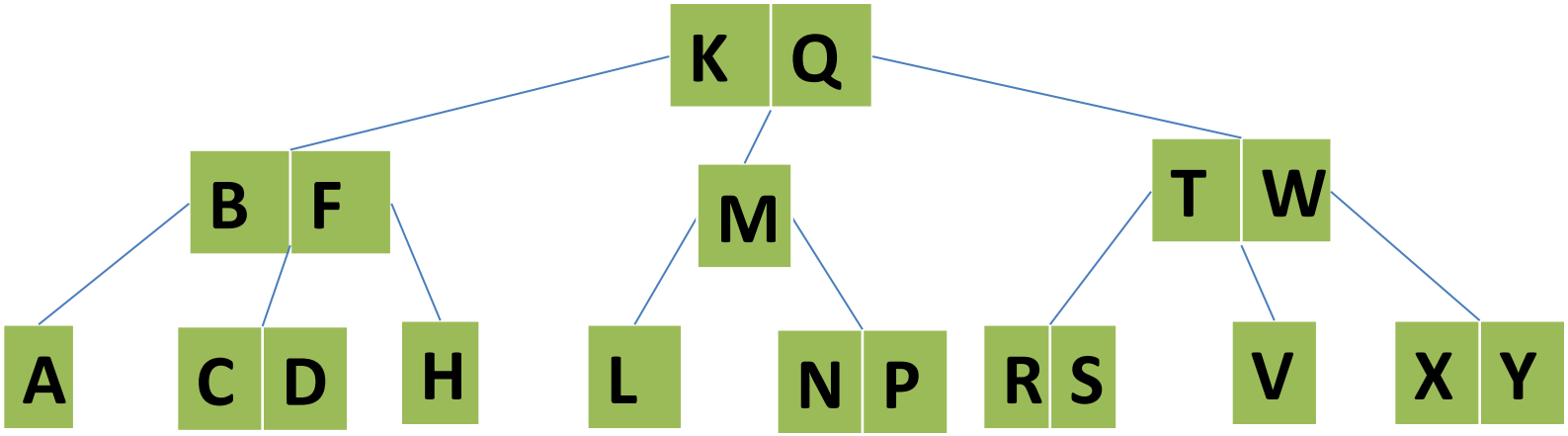


B-Tree

After splitting and inserting next element Y, the B-tree will be

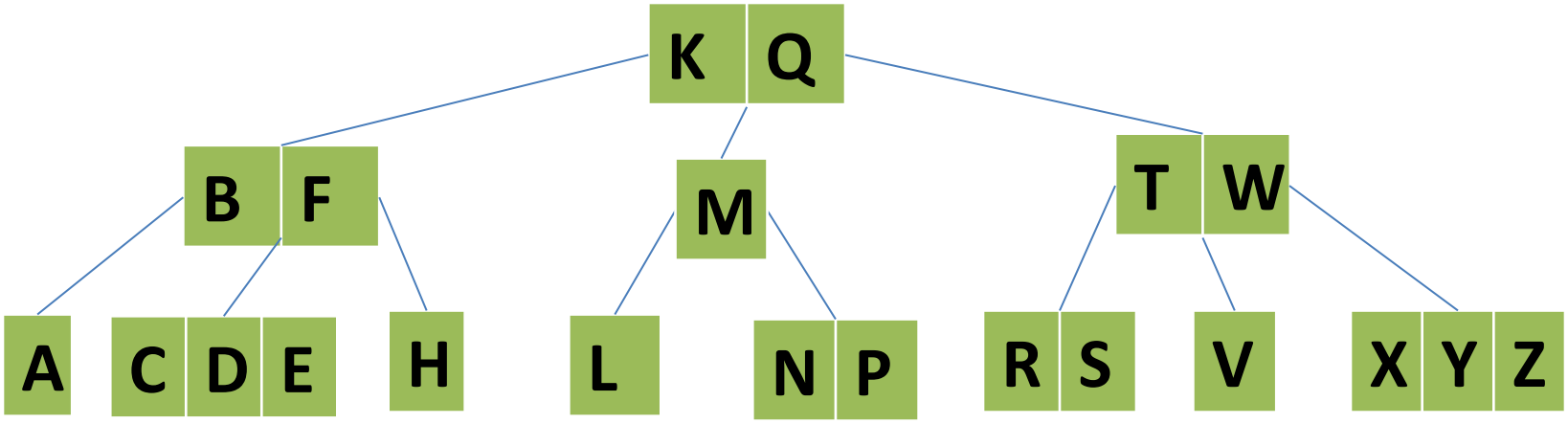


After splitting and inserting next element D, the B-tree will be



B-Tree

After inserting next element Z and E, the B-tree will be



Final B-Tree

B-Tree

Example: Create B-tree for the following elements with minimum degree t = 3.

F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E, G, I.

Example: Insert the following keys in a *2-3-4 B Tree*:

40, 35, 22, 90, 12, 45, 58, 78, 67, 60

Example: Using minimum degree ‘t’ as 3, insert following sequence of integers

10, 25, 20, 35, 30, 55, 40, 45, 50, 55, 60, 75, 70, 65, 80, 85 and 90

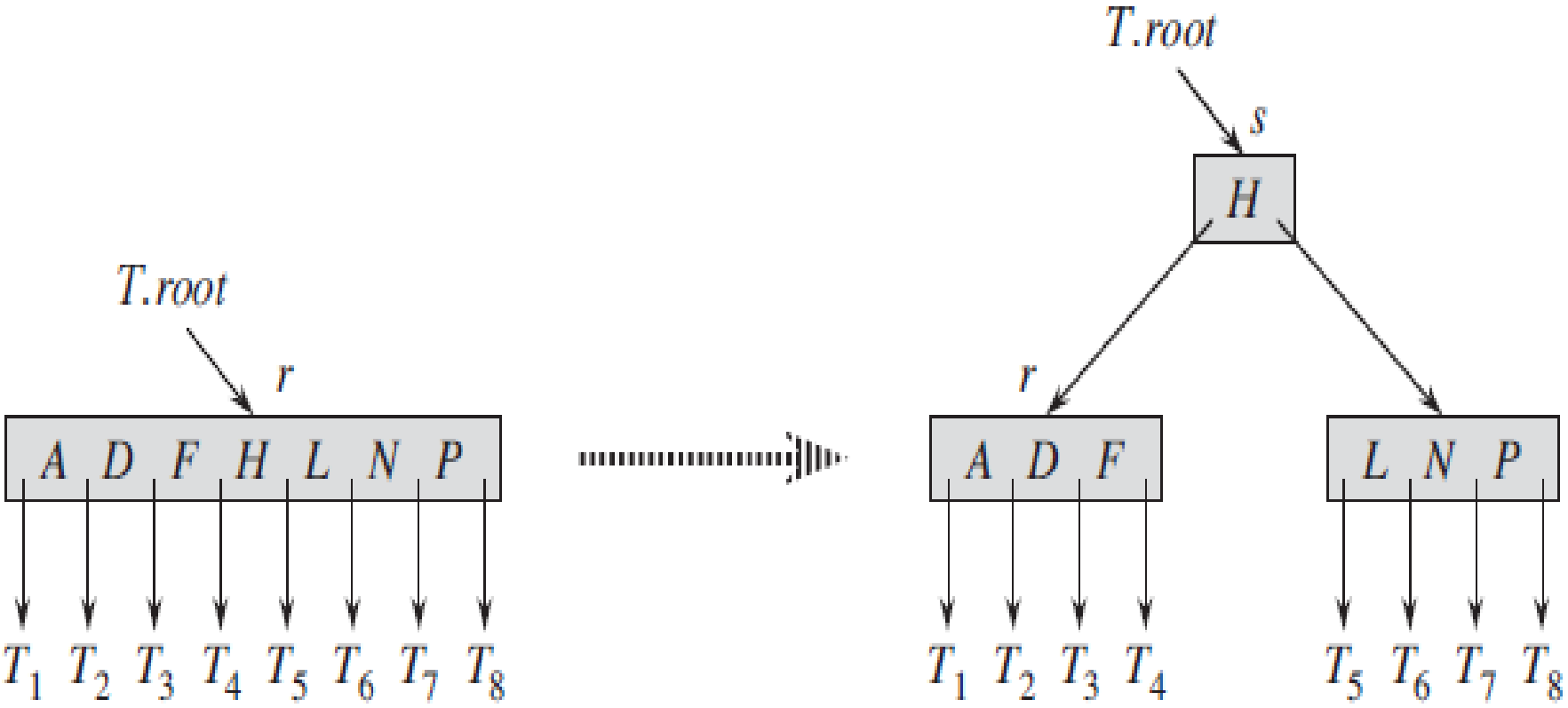
in an initially empty B-Tree. Give the number of nodes splitting operations that take place.

B-Tree Insertion Algorithm

```
B-TREE-INSERT(T, k)
 1  r = T.root
 2  if r.n == 2t − 1
 3      s = ALLOCATE-NODE()
 4      T.root = s
 5      s.leaf = FALSE
 6      s.n = 0
 7      s.c_1 = r
 8      B-TREE-SPLIT-CHILD(s, 1)
 9      B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)
```

B-Tree Insertion Algorithm

Splitting the root with t = 4. Root node r splits in two, and a new root node s is created. The new root contains the median key of r and has the two halves of r as children. The B-tree grows in height by one when the root is split.

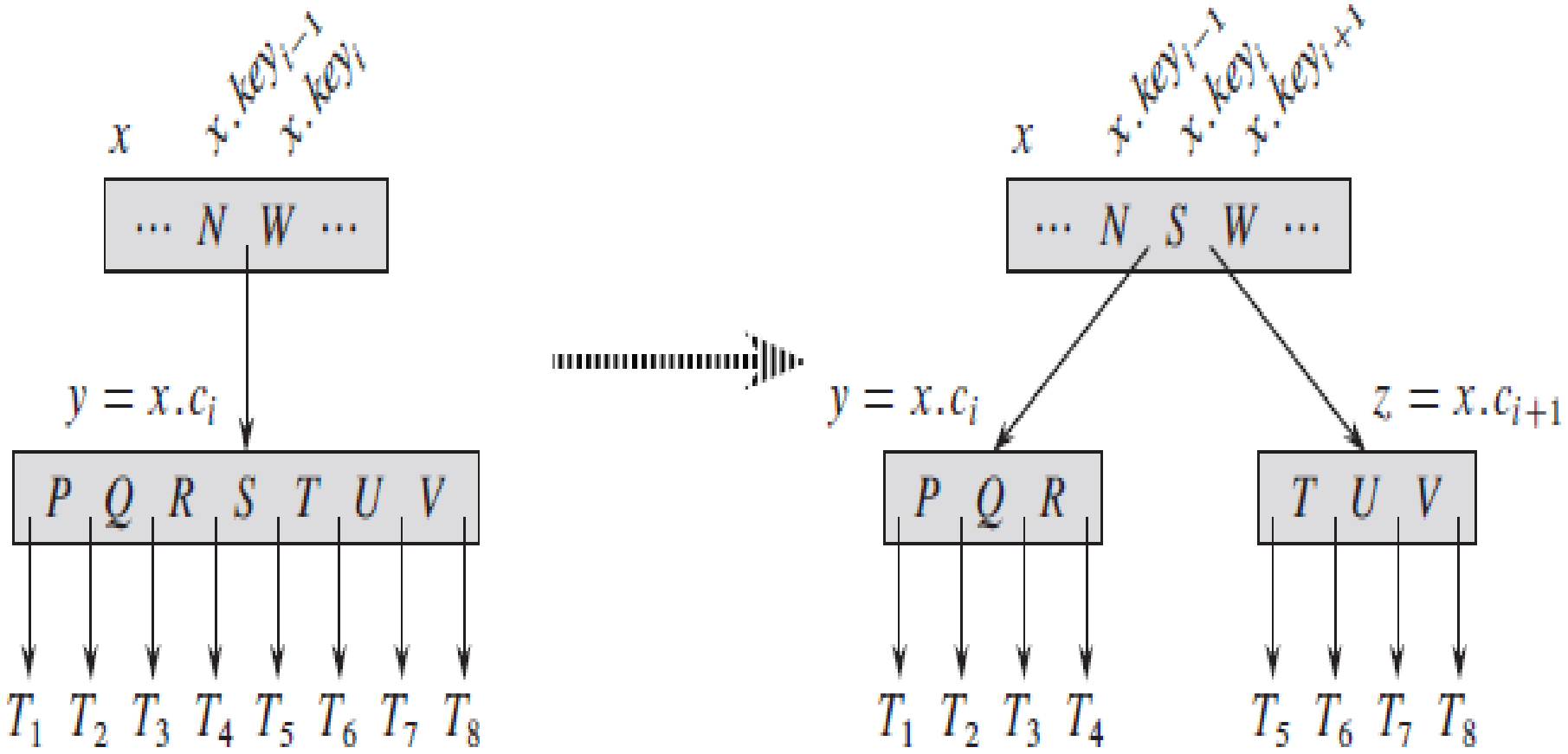


B-Tree Insertion Algorithm

```
B-TREE-SPLIT-CHILD(x, i)
 1  z = ALLOCATE-NODE()
 2  y = x.c_i
 3  z.leaf = y.leaf
 4  z.n = t − 1
 5  for j = 1 to t − 1
 6       z.key_j = y.key_{j+t}
 7  if not y.leaf
 8       for j = 1 to t
 9            z.c_j = y.c_{j+t}
10  y.n = t − 1
11  for j = x.n + 1 downto i + 1
12       x.c_{j+1} = x.c_j
13  x.c_{i+1} = z
14  for j = x.n downto i
15       x.key_{j+1} = x.key_j
16  x.key_i = y.key_t
17  x.n = x.n + 1
18  DISK-WRITE(y)
19  DISK-WRITE(z)
20  DISK-WRITE(x)
```

B-Tree Insertion Algorithm

Splitting a node with t = 4. Node y = $x.c_i$ splits into two nodes, y and z, and the median key S of y moves up into y's parent.



B-Tree Insertion Algorithm

```
B-TREE-INSERT-NONFULL(x, k)
 1  i = x.n
 2  if x.leaf
 3       while i ≥ 1 and k < x.key_i
 4            x.key_{i+1} = x.key_i
 5            i = i − 1
 6       x.key_{i+1} = k
 7       x.n = x.n + 1
 8       DISK-WRITE(x)
 9  else while i ≥ 1 and k < x.key_i
10            i = i − 1
11       i = i + 1
12       DISK-READ(x.c_i)
13       if x.c_i.n == 2t − 1
14            B-TREE-SPLIT-CHILD(x, i)
15            if k > x.key_i
16                 i = i + 1
17       B-TREE-INSERT-NONFULL(x.c_i, k)
```

Time complexity of insertion algorithm

= O(th)

= $O(t \log_t n)$

Deletion operation

Procedure: Suppose we want to delete key k from the B-tree with minimum degree t. Then we use following steps for this purpose:-

1. If the key k is in node x and x is a leaf, then delete the key k from x.

2. If the key k is in node x and x is an internal node, then we do the following:

    2a. If the child y that precedes k in node x has at least t keys, then find the predecessor k’ of k in the subtree rooted at y. Recursively delete k’, and replace k by k’ in x.

Deletion operation

2b. If y has fewer than t keys, then, symmetrically, examine the child z that follows k in node x. If z has at least t keys, then find the successor k’ of k in the subtree rooted at z. Recursively delete k’ , and replace k by k’ in x.

2c. Otherwise, if both y and z have only t-1 keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains 2t-1 keys. Then free z and recursively delete k from y.

Deletion operation

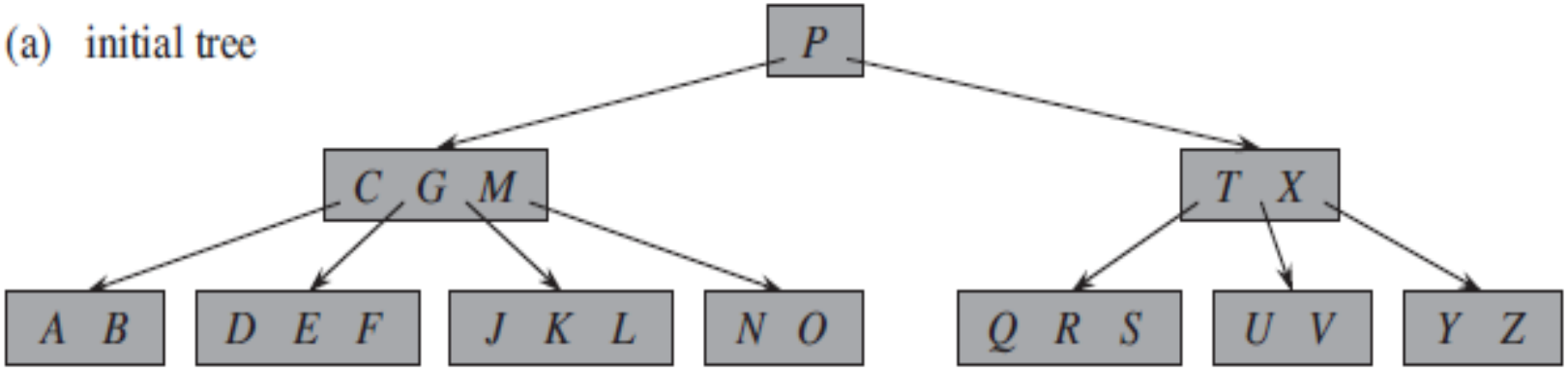
3. If the key k is not present in internal node x, then determine the root $x.c_i$ of the appropriate subtree that must contain k, if k is in the tree at all. If $x.c_i$ has only t-1 keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys. Then finish by recursing on the appropriate child of x.

3a. If $x.c_i$ has only t-1 keys but has an immediate sibling with at least t keys, give $x.c_i$ an extra key by moving a key from x down into $x.c_i$, moving a key from $x.c_i$'s immediate left or right sibling up into x, and moving the appropriate child pointer from the sibling into $x.c_i$.

3b. If $x.c_i$ and both of $x.c_i$'s immediate siblings have t-1 keys, merge $x.c_i$ with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.

Deletion operation

Example: Consider the following B-tree with minimum degree t = 3.



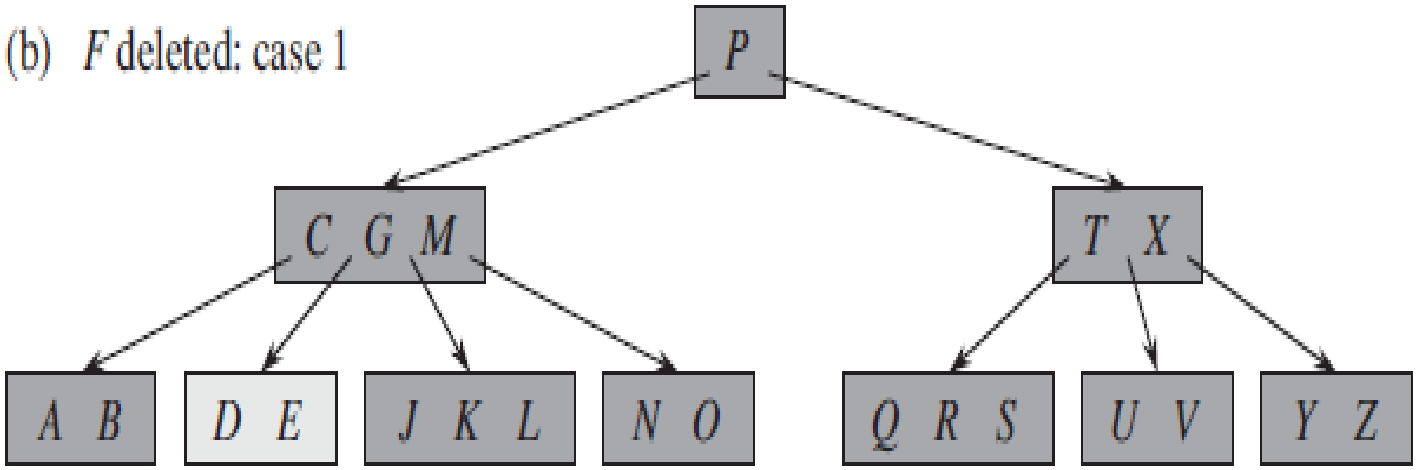
Delete the following elements from this B-tree in-order.

F, M, G, D and B.

Deletion operation

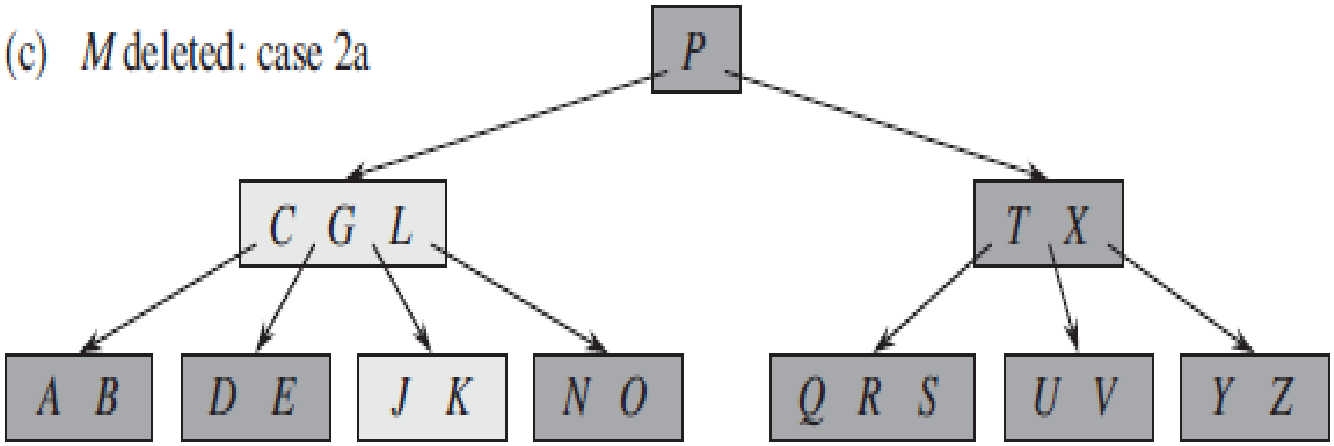
Solution:

Deletion of F



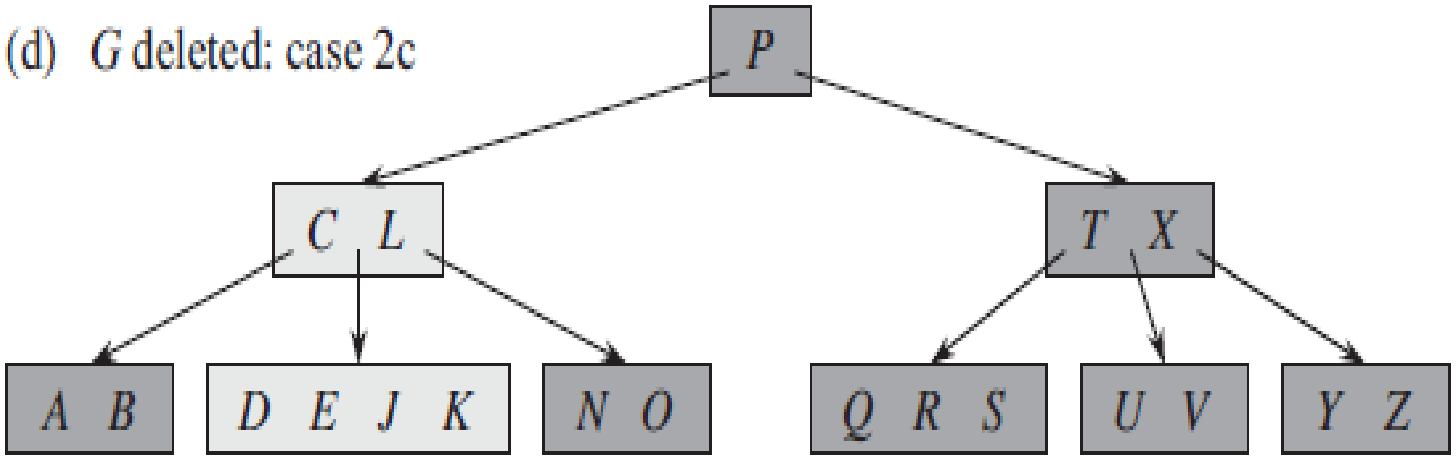
Deletion operation

Deletion of M



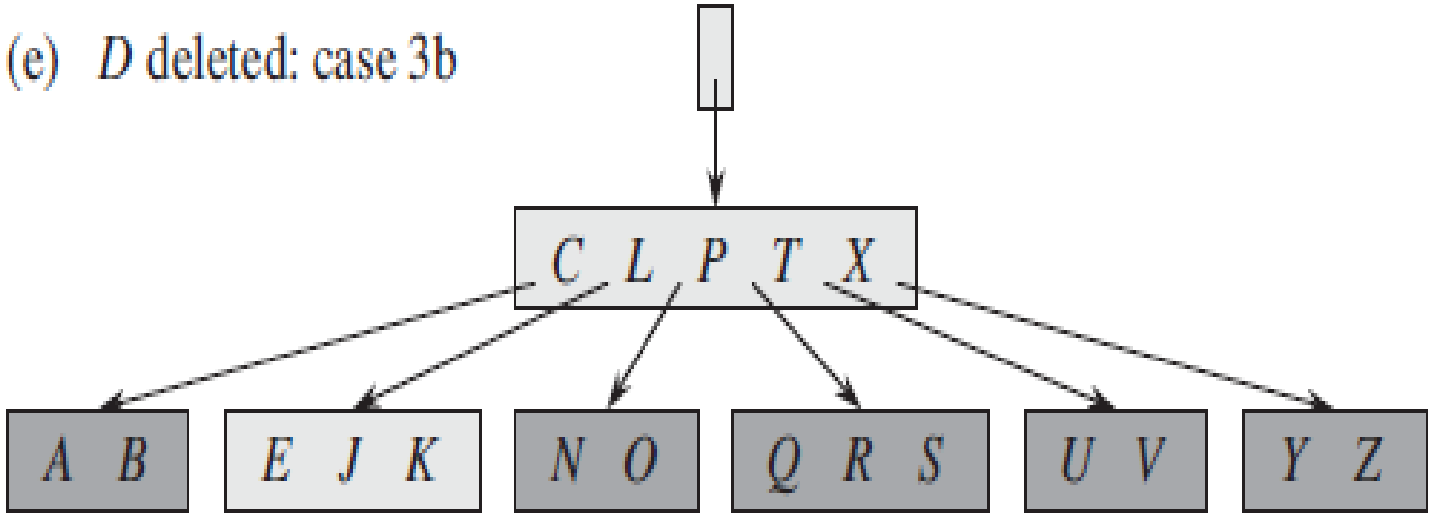
Deletion operation

Deletion of G



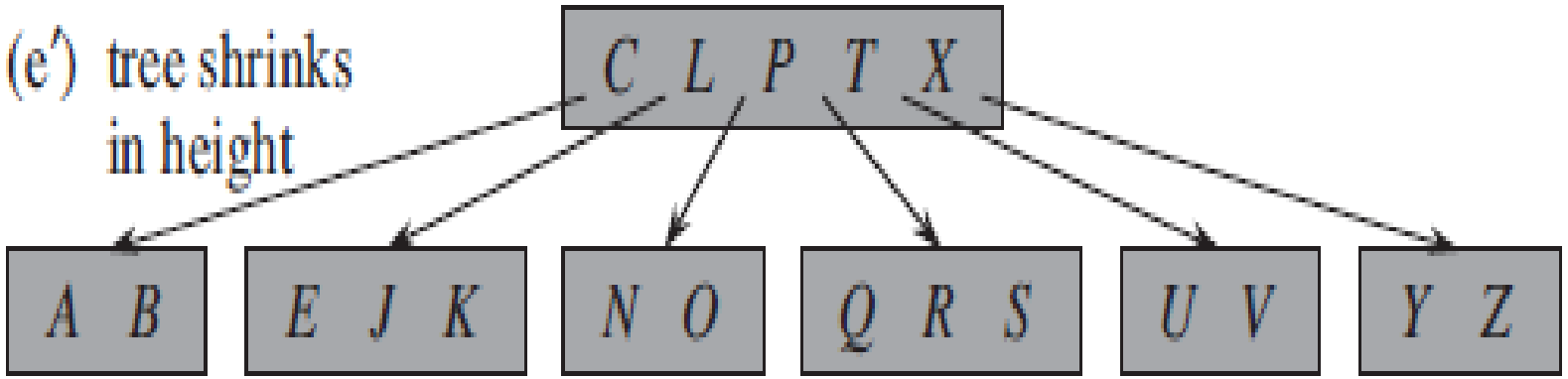
Deletion operation

Deletion of D



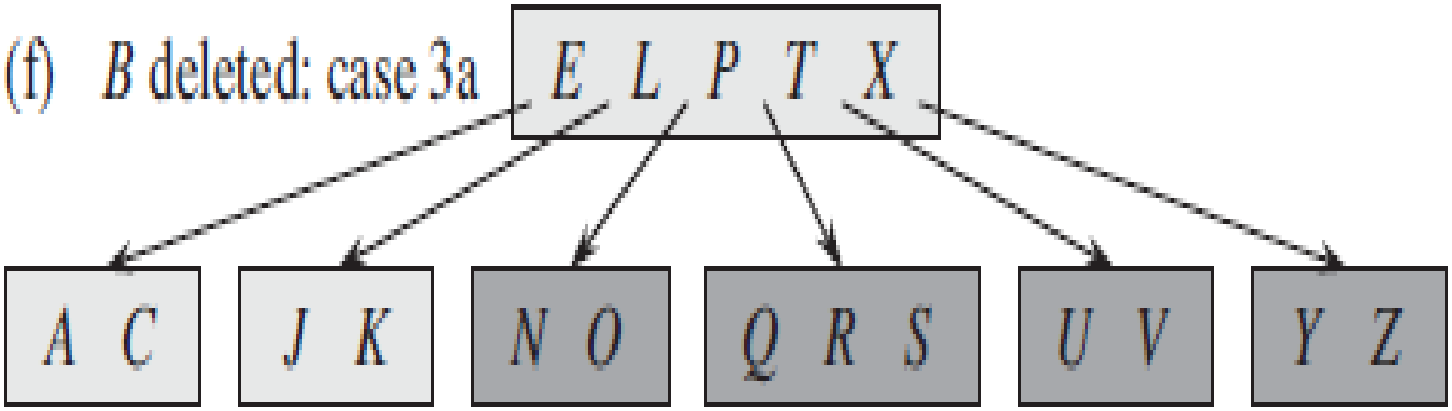
Deletion operation

Deletion of D



Deletion operation

Deletion of B



Final B-tree

Time complexity of deletion operation is O(th) i.e.

$O(th) = O(t \log_t n)$

AKTU Questions

1. Discuss the advantages of using B-Tree. Insert the following Information 86, 23, 91, 4, 67, 18, 32, 54, 46, 96, 45 into an empty B-Tree with degree t = 2 and delete 18, 23 from it.
2. Define a B-Tree of order m. Explain the searching operation in a B-Tree.
3. Using minimum degree ‘t’ as 3, insert following sequence of integers 10, 25, 20, 35, 30, 55, 40, 45, 50, 55, 60, 75, 70, 65, 80, 85 and 90 in an initially empty B-Tree. Give the number of nodes splitting operations that take place.
4. Insert the following information F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E, G, I. Into an empty B-tree with degree t=3.
5. Prove that if n>=1, then for any n-key B-Tree of height h and minimum degree t >=2, h<=log t ((n +1)/2).
6. Insert the following keys in a 2-3-4 B Tree: 40, 35, 22, 90, 12, 45, 58, 78, 67, 60 and then delete key 35 and 22 one after other.

Thank You.