[Unit-III]
Part-I: B-Trees

Objective:

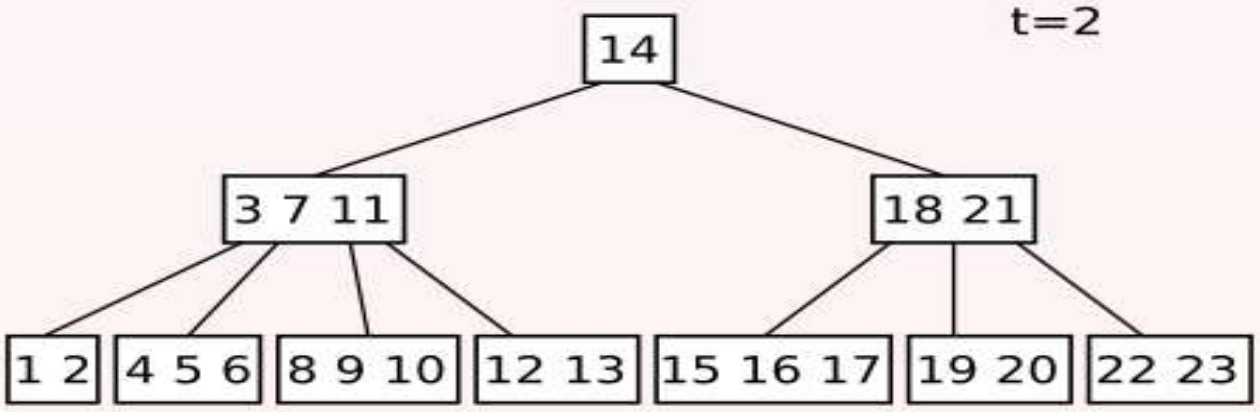
- **To learn importance of B-Tree as data structure**
- **To learn various operations like: Create, Insert, Split, Delete on B-Trees**
- **Develop and analyse algorithms for B-Trees**

B-Tree: Introduction

A B-tree is a rooted tree with the following properties.

1 A node x has the following fields.

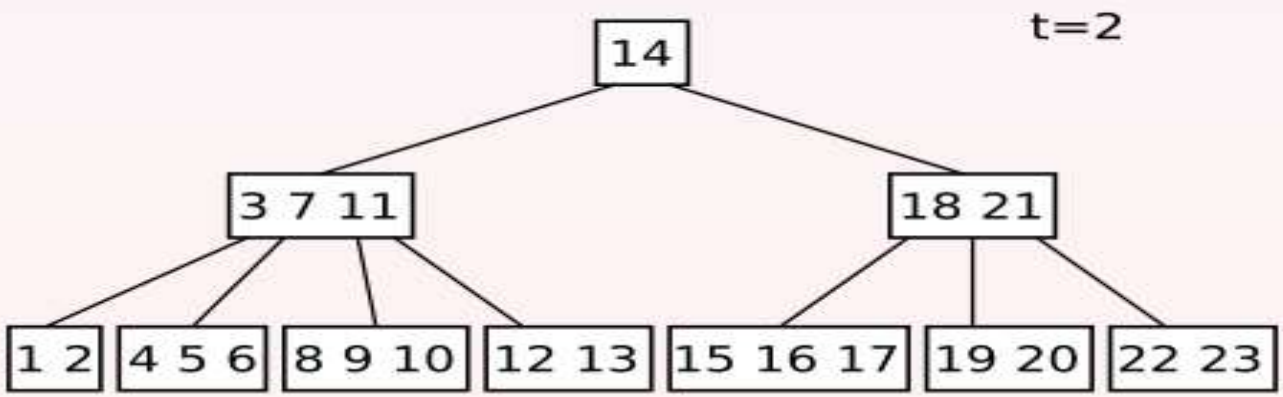
1. $x.n$, the number of keys stored at x.
2. The keys of the elements, $x.\text{key}_1, x.\text{key}_2, \ldots, x.\text{key}_{x.n}$ in increasing order ($x.\text{key}_1 \leq x.\text{key}_2 \leq \cdots$).
3. Pointers to the satellite data of the keys.
4. A boolean value $x.\text{leaf}$ which is TRUE if and only if x is a leaf.



B-Tree: Introduction

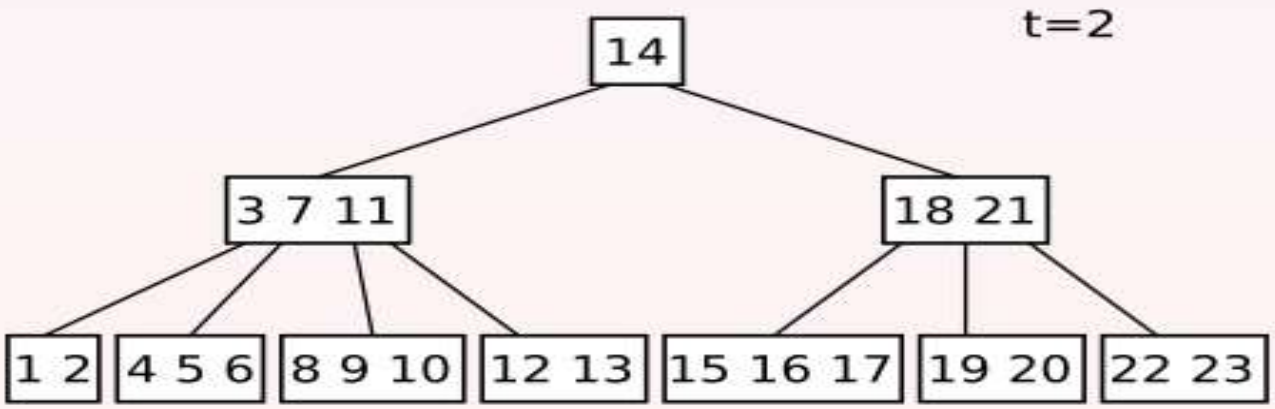
2 An internal nodes x has $x.n + 1$ children. The node x stores pointers to it children $x.c_1, x.c_2, \ldots, x.c_{x.n+1}$.

3 The keys $x.\text{key}_1, \ldots, x.\text{key}_{x.n}$ separate the ranges of keys in the subtrees of x:
If k_i is some key in the subtree of $x.c_i$ then
$k_1 \leq x.\text{key}_1 \leq k_2 \leq x.\text{key}_2 \leq \cdots \leq x.\text{key}_{x.n} \leq k_{x.n+1}$.



B-Tree: Introduction

4. All leaves have the same depth.
5. The tree has a fixed parameter t, called minimum degree, that defines upper and lower bounds on the number of keys stored at each node:
   1. Every node except the root must have at least $t - 1$ keys.
   2. If the tree is not empty, the root must have at least one key.
   3. Every node can have at most $2t - 1$ keys.



B-Tree: Introduction

DEF: A B-Tree of order m is an m-way tree such that

1. All leaf nodes are at the same level.
2. All non-leaf nodes (except the root) have at most m and at least m/2 children.
3. The number of keys is one less than the number of children for non-leaf nodes and at most m-1 and at least m/2 for leaf nodes.
4. The root may have as few as 2 children unless the tree is the root alone.

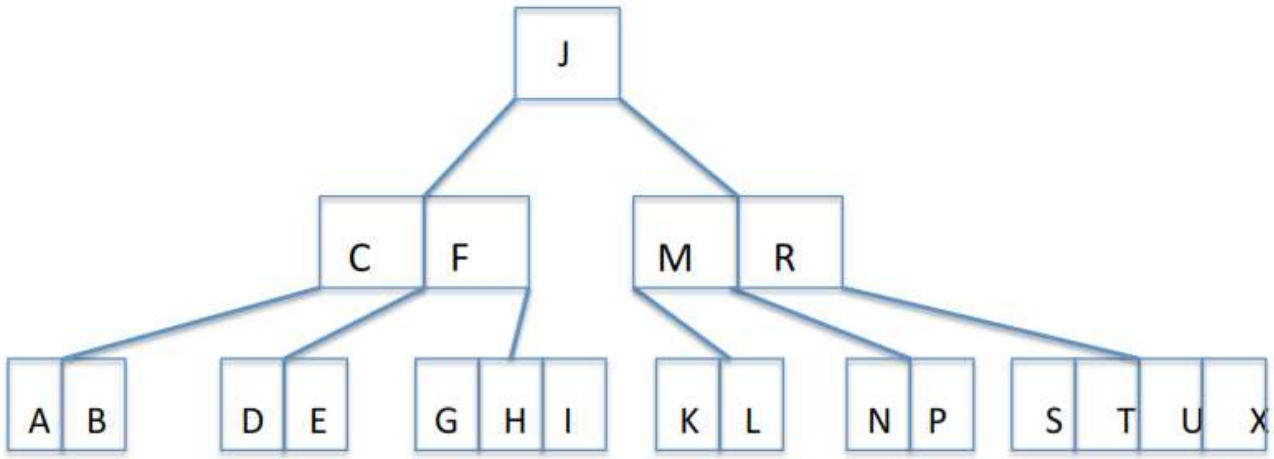
B-Tree: Introduction

DEF: A B-Tree of order 5 is an 5-way tree such that

1. All leaf nodes are at the same level.
2. All non-leaf nodes (except the root) have at most 5 and at least 2 children.
3. The number of keys is one less than the number of children for non-leaf nodes and at most 4 and at least 2 for leaf nodes.
4. The root may have as few as 2 children unless the tree is the root alone.

B-Tree: Introduction

A G F B K D H M J E S I R X C L N T U P



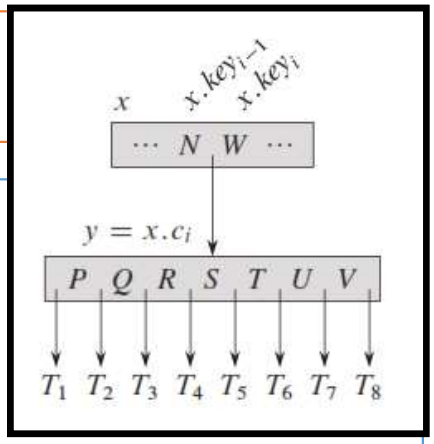
Operation on B Tree

- B-Tree Search
- B-Tree Split
- B-Tree Insert
- B-Tree Delete

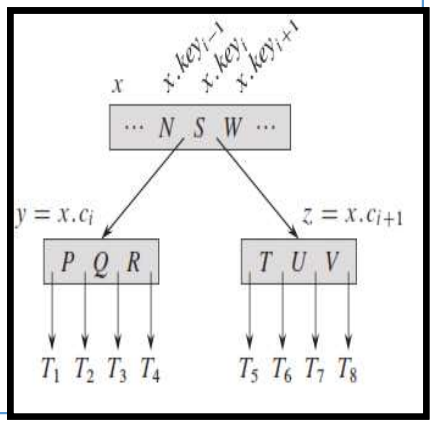
Search operation on B-Tree

```
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)
```

Operation on B Tree : Split



- Address of two nodes is required.
- Address – 1: The node which is full
- Address – 2: Parent of the full node.
- Whenever insert operation is to be performed, every full node should be operated using “Split function”. [t=4] [Node will be considered as full if it has 7 keys, so will need split]
- After split, one node is divided into two nodes
- **<u>Key operations:</u>**
- Pointer management, Data insertion,
- New node creation



Operation on B Tree : Split

- Part-I

```
1  z = ALLOCATE-NODE()
2  y = x.c_i
3  z.leaf = y.leaf
4  z.n = t - 1
```

Get a new node
Y is the node to be split, after split, new node Z is created.
Leaf variable is copied [true/false] in new node
The new node will have “t-1” possible entries.

- Part-II [Transfer data, in new node]

```
5  for j = 1 to t - 1
6      z.key_j = y.key_{j+t}
```

Z.key[j] = Y.key[j+t] [5,6,7]

- Part-III

```
if not y.leaf
    for j = 1 to t
        z.c_j = y.c_{j+t}
y.n = t - 1
```

Z.c[j] = y.c[j+t]
Update y.n=t-1, which was 2t-1

Operation on B Tree : Split

- Part-IV: The new entry removed from child node will be median entry and will be also inserted as median entry in parent node.

```
11  for j = x.n + 1 downto i + 1
12      x.c_{j+1} = x.c_j
13  x.c_{i+1} = z
```

Pointers shifted forward to create space for pointing to new node "Z".
Finally a new pointer is added in parent node pointing to "Z" [new node]

- Part V: Data is shifted forward to create space for data from child node, into parent node.

```
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
```

Move data forward to create space for new data
Increase size by 1

```
18  DISK-WRITE(y)
19  DISK-WRITE(z)
20  DISK-WRITE(x)
```

Rewrite back to disk

B-Tree: Insertion

```
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.c1 = r
 8      B-TREE-SPLIT-CHILD(s, 1)
 9      B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)
```

The split operation can be performed at root, depending on position of insertion. If root contains 2t-1 elements, it is necessary to make newly inserted node as root.
Otherwise, insertion will be carried out using existing root.

B Tree: Insertion

- The new node will be always inserted at leaf. The insertion may or may not require split.

```
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)
```

In B-Tree, insertion will be always at the leaf node.
The leaf node can be also root node.
While travelling to leaf node if any node is already full, then split operation should be performed.
Once reached to leaf node, the data is inserted.