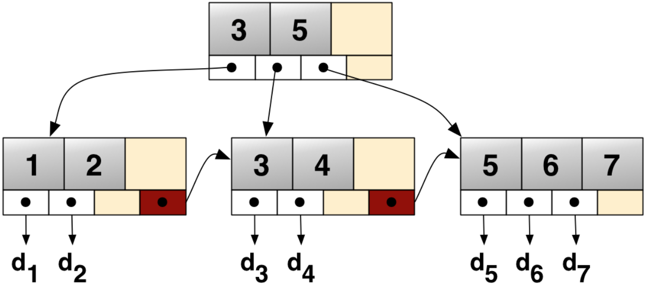
Database Management Systems (CSN-351)
B$^+$ Tree

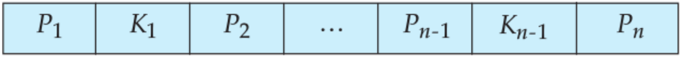
BTech 3rd Year (CS) + Minor

Instructor: **R**anita **B**iswas
Department of Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, Uttarakhand - 247 667, India

B$^+$ Tree



B$^+$ Tree



P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

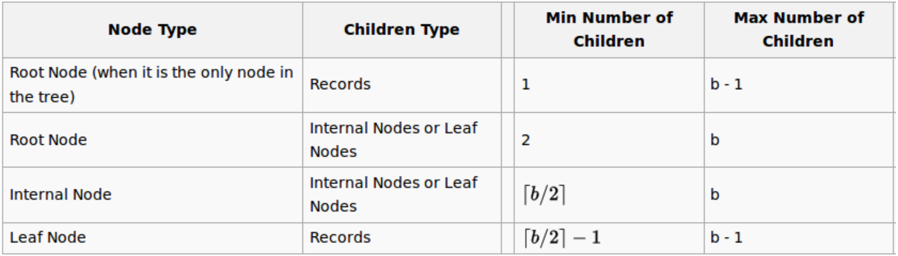
The **order**, or branching factor, b of a B$^+$ tree measures the capacity of nodes (i.e., the number of children nodes) for internal nodes in the tree.

The actual **number of children** for a node, m, is constrained for internal nodes so that $\lceil \frac{b}{2} \rceil \leqslant m \leqslant b$.

The **root** is an exception: it is allowed to have as few as two children.

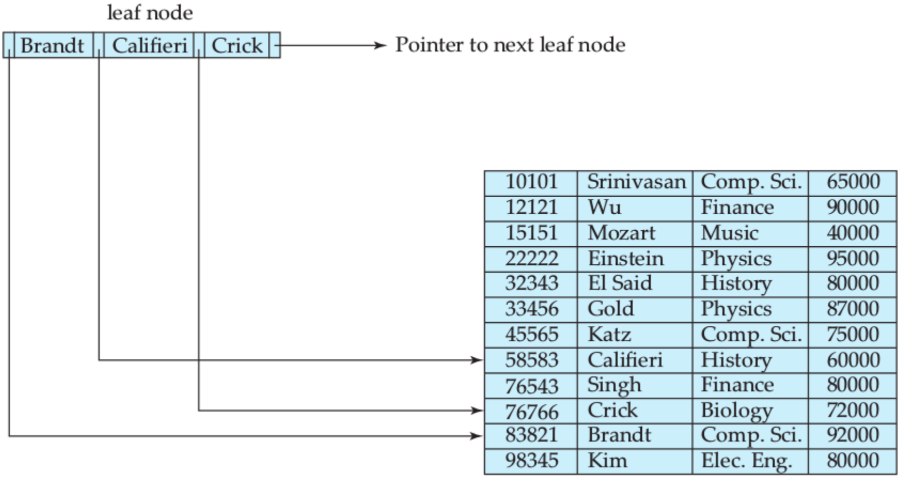
Leaf nodes have no children, but are constrained so that the **number of keys** must be at least $\lceil \frac{b}{2} \rceil - 1$ and at most $b - 1$.

Allowed Children



Node Type	Children Type		Min Number of Children	Max Number of Children
Root Node (when it is the only node in the tree)	Records		1	b - 1
Root Node	Internal Nodes or Leaf Nodes		2	b
Internal Node	Internal Nodes or Leaf Nodes		$\lceil b/2 \rceil$	b
Leaf Node	Records		$\lceil b/2 \rceil - 1$	b - 1

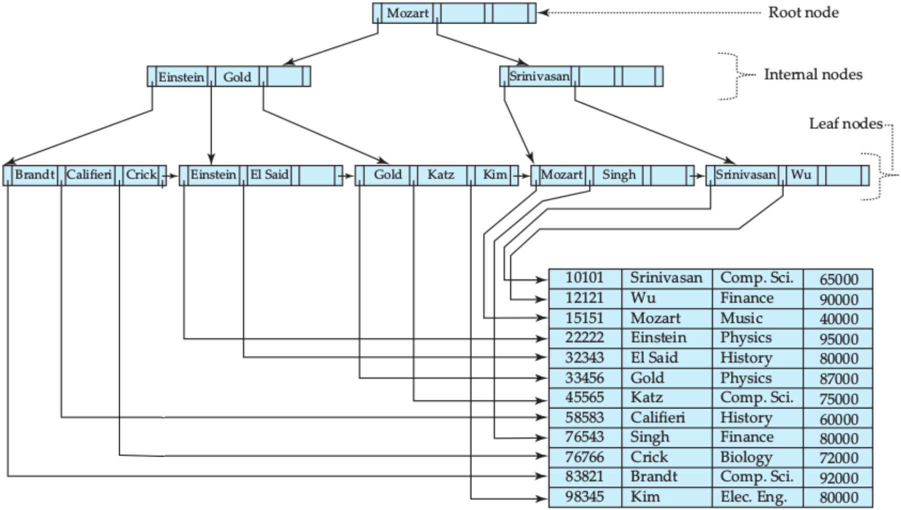
Leaf Nodes



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

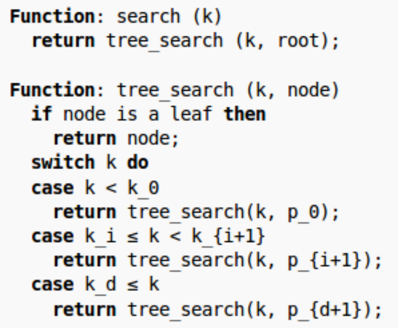
instructor file

Example



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Search



```
Function: search (k)
  return tree_search (k, root);

Function: tree_search (k, node)
  if node is a leaf then
    return node;
  switch k do
  case k < k_0
    return tree_search(k, p_0);
  case k_i ≤ k < k_{i+1}
    return tree_search(k, p_{i+1});
  case k_d ≤ k
    return tree_search(k, p_{d+1});
```

Insertion

Perform a search to determine what bucket the new record should go into.

- If the bucket is not full (at most $b - 1$ entries after the insertion), add the record.
- Otherwise, split the bucket.
    - Allocate new leaf and move half the bucket's elements to the new bucket.
    - Insert the new leaf's smallest key and address into the parent.
    - If the parent is full, split it too.
        - Add the middle key to the parent node.
    - Repeat until a parent is found that need not split.
- If the root splits, create a new root which has one key and two pointers.

B$^+$-trees grow at the root and not at the leaves.

Deletion

Start at root, find leaf L where entry belongs.

Remove the entry.

> If L is at least half-full, done!
>
> If L has fewer entries than it should,
>
> > If sibling (adjacent node with same parent as L) is more than half-full, re-distribute, borrowing an entry from it.
> >
> > Otherwise, sibling is exactly half-full, so we can merge L and sibling.

If merge occurred, must delete entry (pointing to L or sibling) from parent of L.

Merge could propagate to root, decreasing height.

Space and Time Complexities

For a b-order B$^+$ tree with h levels of index:

The maximum number of records stored is $b^h - b^{h-1}$

The minimum number of records stored is $2\left\lceil \frac{b}{2} \right\rceil^{h-1} - 2\left\lceil \frac{b}{2} \right\rceil^{h-2}$

The minimum number of keys is $2\left\lceil \frac{b}{2} \right\rceil^{h-1} - 1$

The maximum number of keys is $b^h - 1$

The space required to store the tree is $O(n)$

Inserting a record requires $O(\log_b n)$ operations

Finding a record requires $O(\log_b n)$ operations

Removing a (previously located) record requires $O(\log_b n)$ operations

Performing a range query with k elements occurring within the range requires $O(\log_b n + k)$ operations

Question 1

Consider a B^+ tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?

Question 1

Consider a B$^+$ tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?

Answer: 2

Question 2

A B^{+} tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place?

Question 2

A B^+ tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place?

Answer: 5

Question 3

The order of an internal node in a B^+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

Question 3

The order of an internal node in a B^+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

Answer: 26

Question 4

The order of a leaf node in a B^+ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

Question 4

The order of a leaf node in a B$^+$ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

Answer: 63

Question 5

A B$^+$ tree used as an index for a large database table has four levels including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the process are

Question 5

A B^+ tree used as an index for a large database table has four levels including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the process are

Answer: 5

Question 6

A B^+ tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 bytes, disk block are size 512 bytes, and index pointers are of size 4 bytes. Given this scenario, what would be the best choice of the degree (i.e. the number of pointers per node) of the B^+ tree?

Question 6

A B$^+$ tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 bytes, disk block are size 512 bytes, and index pointers are of size 4 bytes. Given this scenario, what would be the best choice of the degree (i.e. the number of pointers per node) of the B$^+$ tree?

Answer: 43

Question 7

Consider B^+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is

Question 7

Consider B$^+$ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is

Answer: **50**