B-Trees

B+ Trees

Dr. Thomas Hicks
Computer Science Department
Trinity University
# define M 5

B Trees Nodes

1] Contain M - 1 Records
# define M 5

B Trees Nodes

1] Contain M - 1 Records
2] Contain M Pointers To Other B Tree Nodes
# define M 5

B Trees Nodes

1] Contain M - 1 Records
2] Contain M Pointers To Other B Tree Nodes
3] Contain NoRecords Counter
# define M 5

template <class InfoType>
class BTreeNode {

public:
    BTreeNode(void);

private:
    InfoType Info[M-1];
    NodePtr Ptrs[M];
    long int NoRecords;

};

Assume that sizeof(InfoType) = 9,996   sizeof(BTreeNode) = 40,008
# define M 11

```cpp
template <class InfoType>
class BTreeNode {
public:
    BTreeNode(void);

private:
    InfoType Info[M-1];
    NodePtr Ptrs[M];
    long int NoRecords;
};
```

Assume that sizeof(InfoType) = 9,996

```
sizeof(BTreeNode) = 100,008
```
# define M 5

**About B Trees**

1] Nodes Contain M - 1 Records
2] Nodes Contain M Pointers To Other B Tree Nodes
3] Nodes Contain NoRecords Counter
4] Every Node, Except The Root, Shall Always Be At Least Half Full
BTree <InfoType>
Tree1(5);

Tree1.Inplace(1, 200);

Chose To Be Unique Keys

Records Are Sorted In Ascending Order By Key Within Each Node

Tree1.Inplace(1, 100);

Tree1.Inplace(1, 1400);
BTree <InfoType>
Tree1(5);

Tree1.Inplace(1,400);

Tree1.Inplace(1,500);

Tree1.Inplace(1,300);

Chose To Be Unique Keys
Tree1.Inplace(1, 300);

Chose To Be Unique Keys

Middle Record Goes Up One Level - Create Node If Need! 300
Tree1.Inplace(1, 600);

Chose To Be Unique Keys

Diagram of a tree with nodes containing values 300, 200, 100, 500, and 400.
Tree1.Inplace(1,600) - done

Tree1.Inplace(1,700);
Tree1.Inplace(1,700) - done

Tree1.Inplace(1,50);
Tree1.Inplace(1,50) - done

Tree1.Inplace(1,250);
Tree1.Inplace(1,250) - done

Tree1.Inplace(1,800);
Tree1.Inplace(1,800);

Split 1/3

Middle Record Goes Up One Level - Create Node If Need! 600

Overflow Container
Tree1.Inplace(1,800) - cont

Split 2/3
Tree1.Inplace(1,800) - done

Split 3/3

[1] 250
  200
  100
  50
  / / / / / 4

[2] 600
  300
  1 3 4 / / 2

[3] 500
  400
  / / / / / 2

[4] 800
  700
  / / / / / 2
Tree1.Inplace(1, 150);
Tree1.Inplace(1,150)

Split 1/3

Middle Record Goes Up One Level - Create Node If Need! 150
Tree1.Inplace(1,150) - cont

Split 2/3
Tree1.Inplace(1,150) - done
Split 3/3
Tree1.Inplace(1,550);
Tree1.Inplace(1, 550) - done

Tree1.Inplace(1, 450);
Tree1.Inplace(1,450) - done

Tree1.Inplace(1,350);
Tree1.Inplace(1,350)

Split 1/3

[1]

[2]

[3]

[4]

[5]

Middle Record Goes Up One Level - Create Node If Need! 450
Tree1.Inplace(1,350) - cont

Split 2/3
Tree1.Inplace(1,350) - done

Split 3/3
Tree1.Inplace(1, 150)
Tree1.Inplace(1,750) - done

Tree1.Inplace(1,850);
Tree1.Inplace(1, 850) - done

Tree1.Inplace(1, 900);
Tree1.Inplace(1,900)

Split 1/8

[1]

[2]

[3]

[4]

[5]

[6]
Tree1.Inplace(1, 900)

Split 2/8
Tree1.Inplace(1,900)

Split 3/8
Tree1.Inplace(1,900)

Split 4/8

Ignore This Portion For The Moment
Tree1.Inplace(1,900)

Split 5/8

Back To The Portion We Ignored!
Tree1.Inplace(1,900)

Split 6/8

[1]

[2]

[3]

[5]

[7]

[8]

[1]

[5]

[3]
Tree1.Inplace(1,900)

Split 7/8
Tree1.Inplace(1,900)

Split 8/8
# define M 5

B+Trees - also called M-Way Trees - A balanced search tree in which every node has between M-1 and 2M-1 children, where M is an arbitrary constant.

This is a good structure if much of the tree is in slow memory (disk), since the height, and hence the number of accesses, can be kept small, say one or two, by picking a large t.
B-Tree

B-Tree-Search(x, k)
i <- 1
while i <= n[x] and k > keyi[x]
    do i <- i + 1
if i <= n[x] and k = keyi[x] then
    return (x, i)
if leaf[x] then
    return NIL
else
    Disk-Read(ci[x])
return B-Tree-Search(ci[x], k)

http://www.public.asu.edu/~peterjn/btree/