

11

B-Trees

Copyright © 2004, Binnur Kurt

Content

- ▶ Introduction to multilevel indexing and B-trees
- ▶ Insertion in B Trees
- ▶ Search and Insert Algorithms
- ▶ Deletion in B Trees

B-Trees 11

File Organization

270

Introduction to Multilevel Indexing and B-Trees

- ▶ Problems with simple indexes that are kept in disk:
 1. Seeking the index is still slow (binary searching):
 - We do not want more than 3/4 seeks for a search
 - So, here $\log_2(N+1)$ is still slow:

N	$\log_2(N+1)$
15 keys	4
1,000	~10
100,000	~17
1,000,000	~20

B-Trees 11

File Organization

271

Introduction to Multilevel Indexing and B-Trees

- ▶ Problems with simple indexes that are kept in disk:
 2. Insertions and deletions should be as fast as searches:
 - In simple indexes, insertion or deletion take $O(n)$ disk accesses (since index should be kept sorted)

B-Trees 11

File Organization

272

Indexing with Binary Search Trees

- ▶ We could use balanced binary search trees:
- ▶ **AVL** Trees
 - Worst-case search is $1.44 \times \log_2(N+2)$
 - 1,000,000 keys \rightarrow 29 Levels
 - Still prohibitive
- ▶ **Paged Binary** Trees
 - Place subtree of size K in a single page
 - Worst-case search is $\log_{K+1}(N+1)$
 - $K=511$, $N=134,217,727$
 - Binary trees: 27 seeks, Paged Binary tree: 3 seeks
 - This is good but there are lots of difficulties in maintaining (doing insertions and deletions) in a paged binary tree

B-Trees 11

File Organization

273

Multilevel Indexing

- ▶ Consider our 8,000,000 example with keysize = 10B
- ▶ Index file size = 80 MB
- ▶ Each record in the index will contain 100 pairs (key,reference)
- ▶ A simple index would contain: 80,000 records:
 - Too expensive to search (~16 seeks)

B-Trees 11

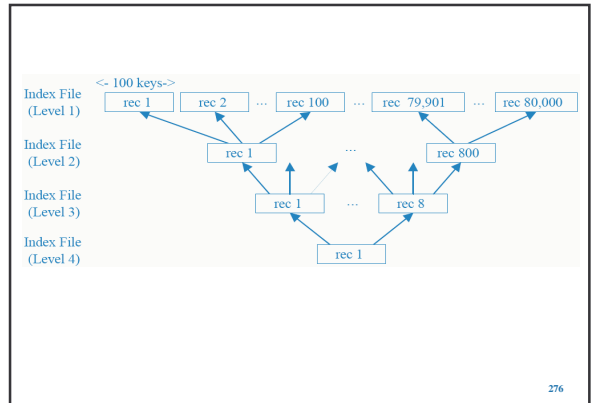
File Organization

274

Multilevel Index

- ▶ Build an index of an index file
- ▶ How?
 - Build a simple index for the file, sorting keys using the method for external sorting previously studied
 - Build an index for this index
 - Build another index for the previous index, and so on
 - The index of an index stores the largest in the record it is pointing to.

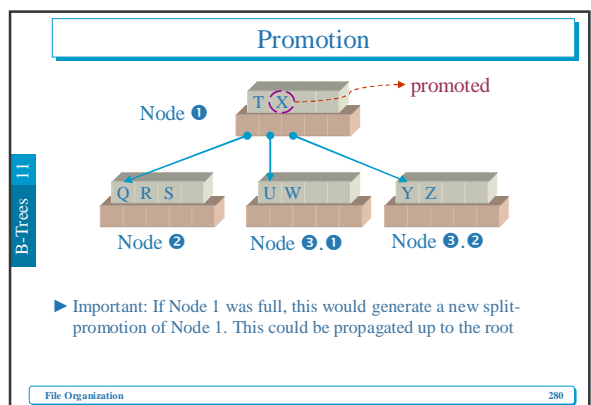
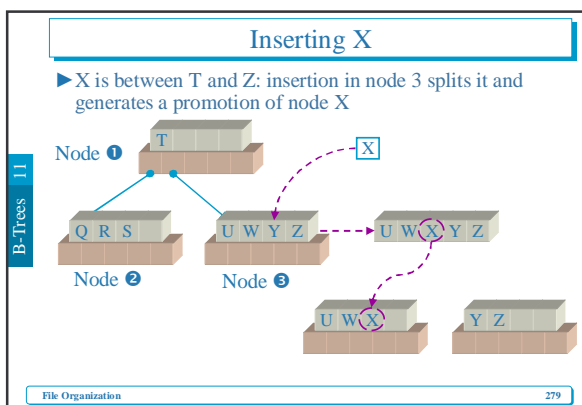
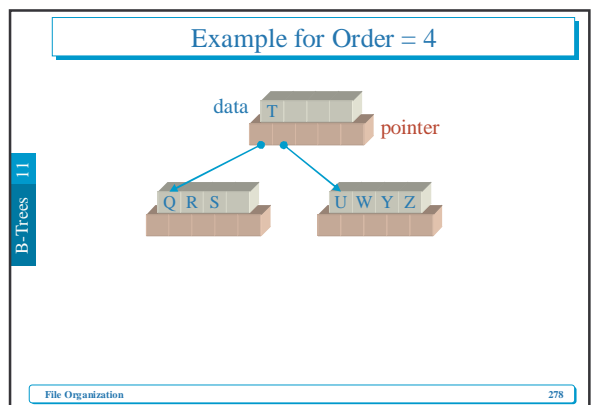
B-Trees 11
File Organization 275



B-Trees

- ▶ Again an index record may contain 100 keys
- ▶ An index record may be half full (each index record may have from 50 to 100 keys)
- ▶ When insertion in an index record causes it to overflow
 - Split record in two
 - “Promote” the largest key in one of the records to the upper level

B-Trees 11
File Organization 277



Example of Insertions

- ▶ Inserting keys: order = 4
- ▶ A,G,F,B,K,D,H,M,J,E,S,I,R,X,C,L,N,T,U,P

B-Trees 11

▶ Inserting K: Split and Promotion

File Organization 281

Insertion Example

- ▶ Inserting D

- ▶ Inserting H

File Organization 282

Insertion Example

- ▶ Inserting M

- ▶ Inserting J: Split and Promotion

File Organization 283

Insertion Example

- ▶ Inserting E

- ▶ Inserting S

File Organization 284

Insertion Example

- ▶ Inserting I

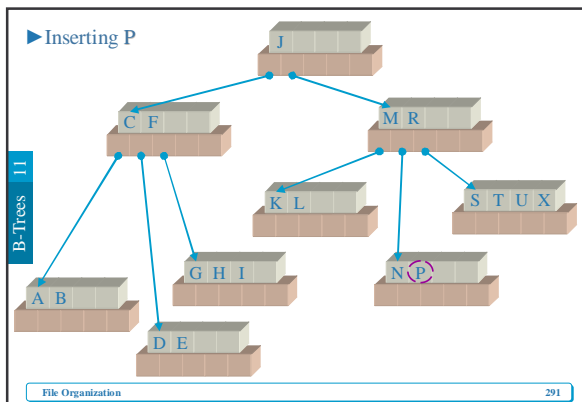
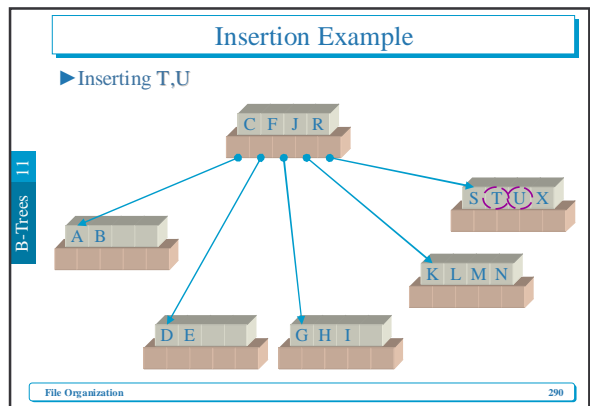
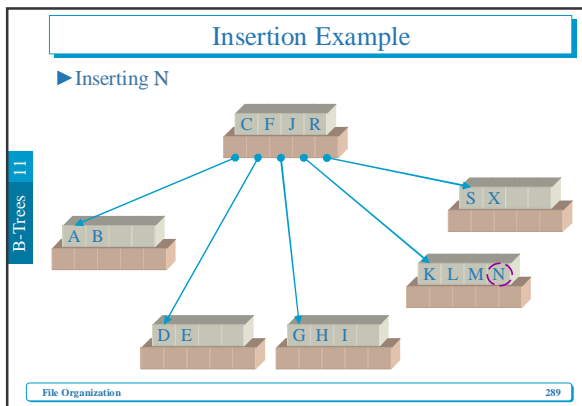
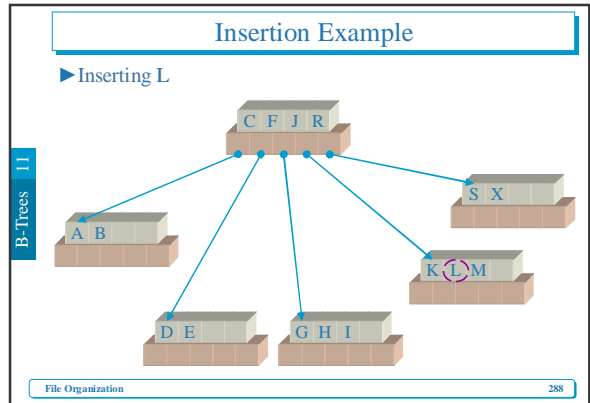
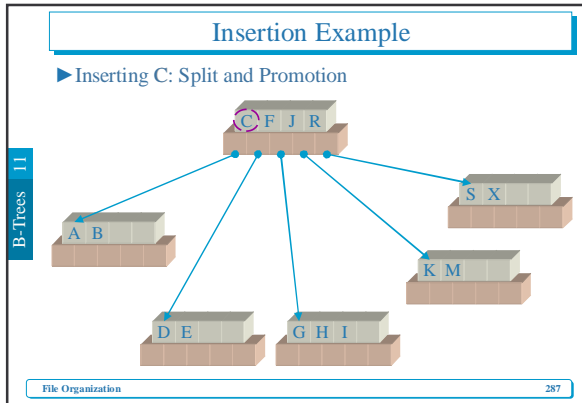
- ▶ Inserting R

File Organization 285

Insertion Example

- ▶ Inserting X: Split and Promotion

File Organization 286



- ### B-Tree Properties
- Properties of a B-tree of order m :
1. Every node has a maximum of m children
 2. Every node, except for the root and the leaves, has at least $m/2$ children
 3. The root has at least two children (unless it is a leaf)
 4. All the leaves appear on the same level
 5. The leaf level forms a complete index of the associated data file
- File Organization 292

Worst-case Search Depth

► The worst-case depth occurs when every node has the minimum number of children

Level	Minimum number of keys (children)
1 (root)	2
2	$2 \cdot \lceil m/2 \rceil$
3	$2 \cdot \lceil m/2 \rceil \cdot \lceil m/2 \rceil = 2 \cdot \lceil m/2 \rceil^2$
4	$2 \cdot \lceil m/2 \rceil^3$
...	...
d	$2 \cdot \lceil m/2 \rceil^{d-1}$

File Organization 293

Example

► Assume that we have N keys in the leaves

So,

$$N \geq 2 \cdot (m/2)^{d-1}$$

$$d \leq 1 + \log_{m/2} (N/2)$$

For $N=1,000,000$ and order $m=512$, we have

$$d \leq 1 + \log_{256} (N/2)$$

$$d \leq 3.37$$

► There is at most 3 levels in a B-tree of order 512 holding 1,000,000 keys.

File Organization 294

Outline of Search Algorithm

► Search (KeyType key)

1. Find leaf: find the leaf that could contain key, loading all the nodes in the path from root to leaf into an array in main memory
2. Search for key in the leaf which was loaded into main memory

File Organization 295

Deletions from B-Tree

► The rules for deleting a key K from a node n :

1. If n has more than the minimum number of keys and K is not the largest key in n , simply delete K from n .
2. If n has more than the minimum number of keys and K is the largest key in n , delete K from n and modify the higher level indexes to reflect the new largest key in n .
3. If n has exactly the minimum number of keys and one of the siblings has “few enough keys”, **merge** n with its sibling and delete a key from the parent node

File Organization 296

Deletions from B-Tree (Con't)

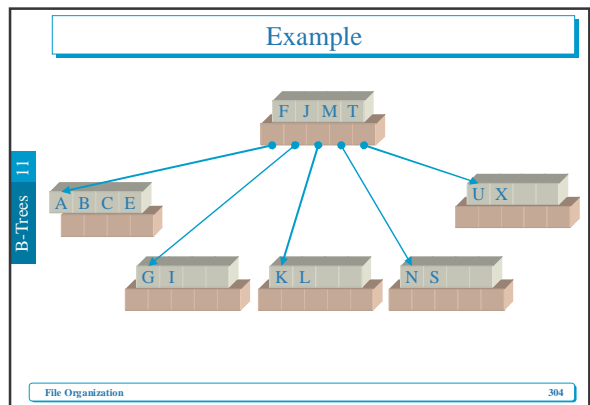
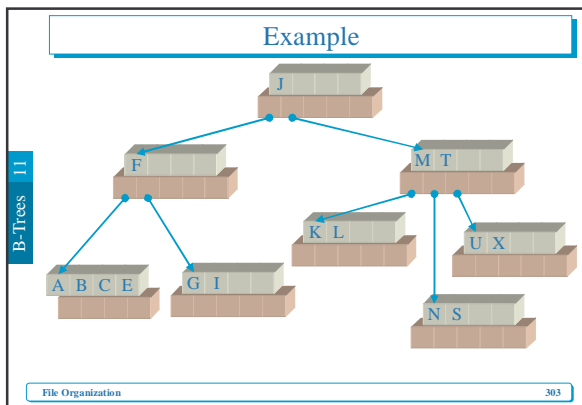
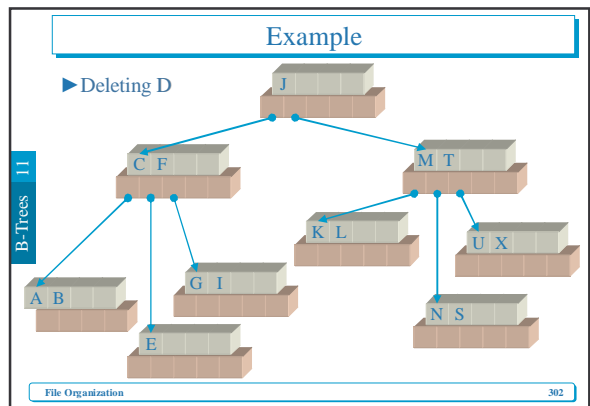
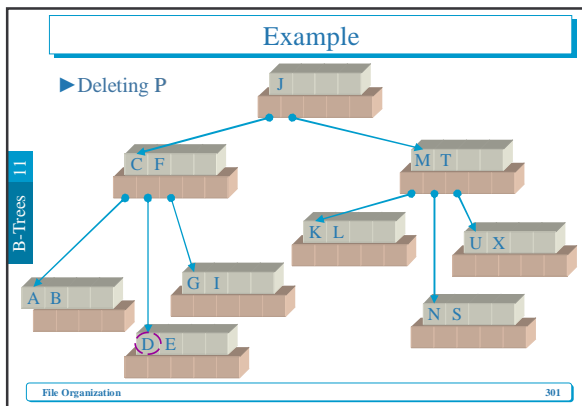
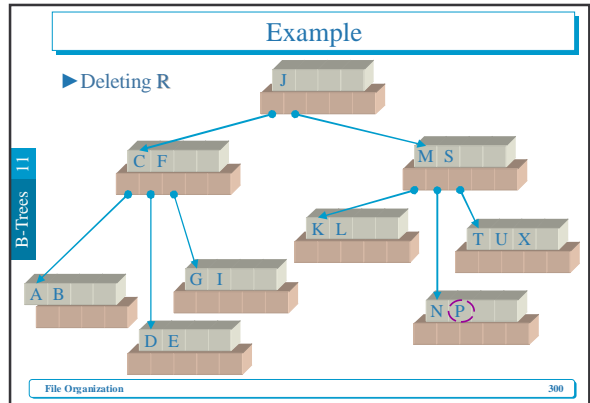
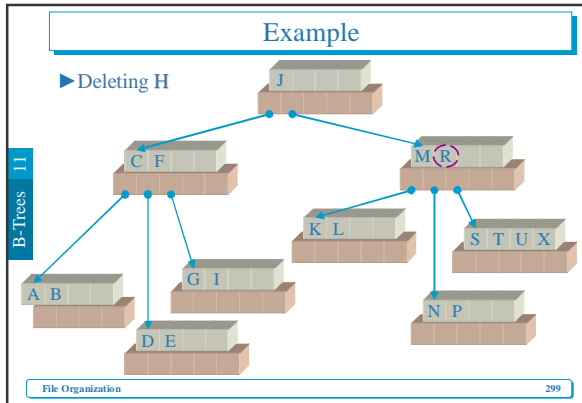
► The rules for deleting a key K from a node n :

4. If n has exactly the minimum number of keys and one of the siblings has extra keys, **redistribute** by moving some keys from a sibling to n , and modify higher levels to reflect the new largest keys in the affected nodes

File Organization 297

Example

File Organization 298



Implementation

```

typedef struct {
    int Count; // Number of keys stored in the current
    ItemType Key[MaxKeys];
    long Branch[MaxKeysPlusOne];
} NodeType;

```

B-Trees 11

File Organization 305

Implementation (Con't)

```

typedef struct {
    int Count; // Number of keys stored in the current
    ItemType Key[MaxKeys];
    long Branch[MaxKeysPlusOne];
    bool Search(KeyFieldType SearchKey,
                ItemType & Item);
} NodeType;

```

B-Trees 11

File Organization 306

bool Search(KeyFieldType SearchKey,...)

```

{
    long CurrentRoot;
    int Location;
    bool Found;
    Found = false;
    CurrentRoot = Root;
    while ((CurrentRoot != -1L) && (!Found)) {
        fseek(hFile, CurrentRoot * NodeSize, SEEK_SET);
        fread((unsigned char *)&CurrentNode, NodeSize, 1, hFile);
        if (SearchNode(SearchKey, Location)) {
            Found = true;
            Item = CurrentNode.Key[Location];
        } else CurrentRoot = CurrentNode.Branch[Location + 1];
    }
    return Found;
}

```

B-Trees 11

File Organization 307

bool SearchNode(KeyFieldType Target,...)

```

bool SearchNode(const KeyFieldType Target, int & Location) const {
    bool Found=false;
    if (strcmp(Target, CurrentNode.Key[0].KeyField)<0) Location=-1L;
    else
    {
        Location = CurrentNode.Count - 1;
        while ((strcmp(Target, CurrentNode.Key[Location].KeyField) < 0)
            && (Location > 0))
            Location--;
        if (strcmp(Target, CurrentNode.Key[Location].KeyField) == 0)
            Found = true;
    }
    return Found;
}

```

B-Trees 11

File Organization 308