Indexing

Swarup Kr Ghosh

@SKG 1

Indexing

- Auxiliary Access Structure
  - Used to speed up the retrieval of records in response to certain search condition
- Secondary Access Path
  - Alternative way of accessing the records without affecting the physical placement of records
  - Enables efficient access of records based on indexing fields

@SKG 22

Types of Indexing

- Indexing
  - **Single Level Indexing**
    - **Primary Index** — on the key field of ordered file
    - **Clustering Index** — on the non-key field of ordered file
    - **Secondary Index** — on the non-ordered file (Key/ Non-key)
  - **Multi- Level Indexing**
    - **Static**
    - **Dynamic** — using B-Trees and B+ Trees

Indexes on Multiple Keys

@SKG 33

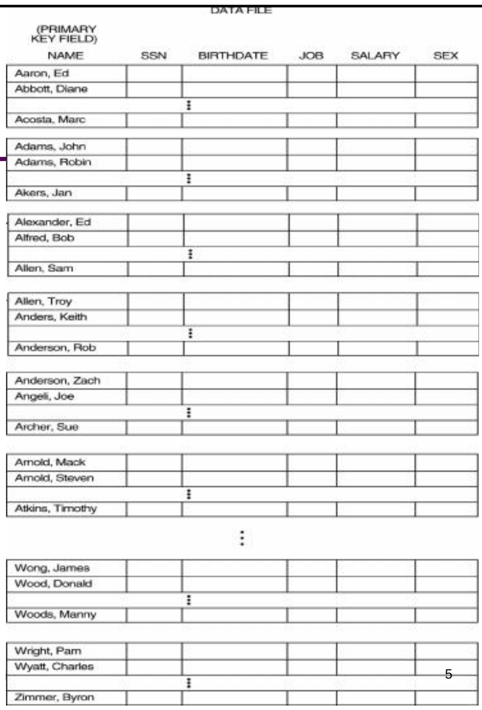
Indexing

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries **<field value, pointer to record>,** which is ordered by field value
- The index is called an *access path*on the field

@SKG 44

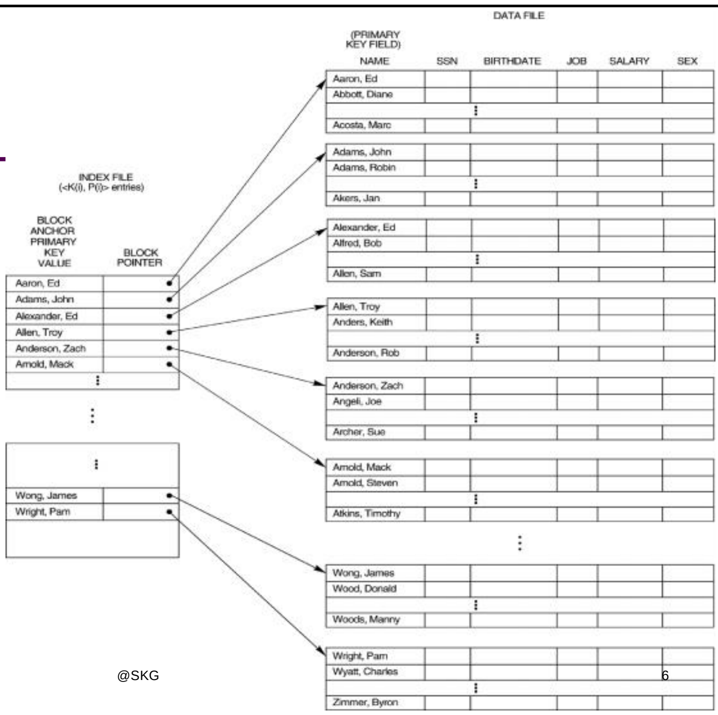
Primary Index

- on the key field of ordered file
- Searching a data based on primary index



@SKG 5

Primary Index



@SKG 6

Primary Index

- Defined on an ordered data file
- The data file is ordered on a *key field*
- Primary index is an ordered file ⊠records are of fixed length wh two fields
  - First field is same as the data type of the ordering key field.PK of the data file
  - Second field is a pointer to a disk block (block address)
  - <K(i),P(i)>
- One index entry for in the index file for each block in the data file
- Each index entry has the value of the primary key field for the first record in a block and the pointer to that block

@SKG 7

Primary Index

- Includes one index entry *for each block*in the data file; the index entry has the key field value for the *first record*in the block, which is called the *block anchor/ anchor record*
- A similar scheme can use the *last record*in a block
  - A primary index is a non-dense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value

How to search the data using primary index ?

@SKG 8

Primary Index: Problem

Example: Consider the following data file:

EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)

Problem1: Suppose, we have an ordered file with 30,000 records stored on a disk with block size 1024 Bytes. File records are of fixed size and un-spanned with record length of 100 Bytes.

Calculate-

- The blocking factor of the file.
- The number of blocks needed to store the file.
- The number of block access needed to access a data from the data file (Assume that binary search has been implemented as a search scheme).

Primary Index: Solution

Example Ans.: Given the following data file:

Suppose that: r=30000 records, record size R=100 bytes

block size B=1024 bytes

Then, we get:

Blocking factor of the file : $bfr = \left\lfloor \frac{B}{R} \right\rfloor = \left\lfloor \frac{1024}{100} \right\rfloor = 10 \text{ records/block}$

Number of file blocks: $b = \left\lceil \frac{r}{bfr} \right\rceil = \left\lceil \frac{30000}{10} \right\rceil = 3000 \text{ blocks}$

Total number of block access: $\lceil \log_2 b \rceil = \lceil \log_2 3000 \rceil = 12 \text{ block access}$

Primary Index: Problem

Problem2: Now assume that the file is ordered w.r.t. the key field of the file having 9 bytes long. The size of block pointer is 6 bytes and we have constructed a primary index for the file. Calculate-

i) The blocking factor of the index file.

ii) The total number of blocks needed to store the index file.

iii) The number of block access needed to access a data from the data file (Assume binary search has been implemented as a search scheme)

@SKG 11

Primary Index: Solution

Example Ans.: Given the following data file:

Suppose that: Ordering key field size V=9 bytes,

Block Pointer P=6 bytes,

Then, we get: Size of each index entry: $R_i = (9+6) = 15$ bytes

Blocking factor of the Index : $bfr_i = \left\lfloor \frac{B}{R_i} \right\rfloor = \left\lfloor \frac{1024}{15} \right\rfloor = 68$ records/block

Number of blocks to store the index: $b_i = \left\lceil \frac{r_i}{bfr_i} \right\rceil = \left\lceil \frac{3000}{68} \right\rceil = 45$ blocks

Total number of block access: $\lceil \log_2 b_i \rceil + 1 = \lceil \log_2 45 \rceil + 1 = 7$ block access

@SKG 12

Primary Index

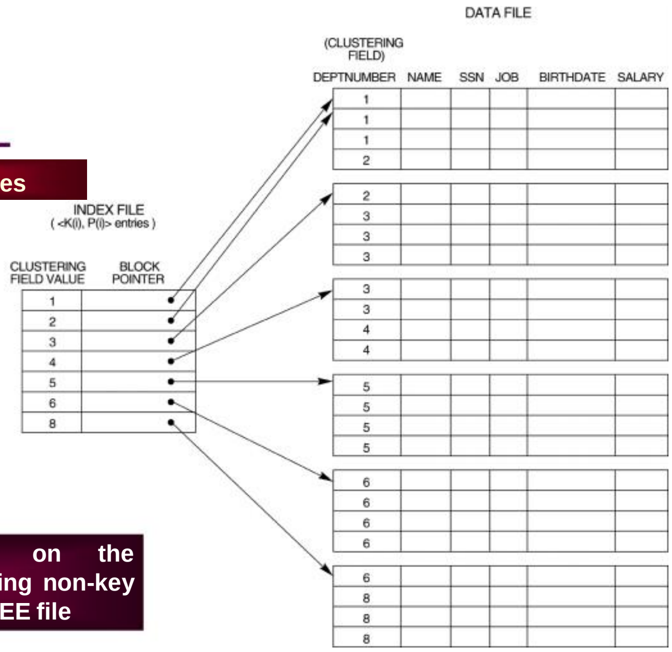
- Advantages
- Disadvantages
- Insertion/ Deletion

Clustering Index

- Records of the file are physically ordered w.r.t non-key field ☒ clustering field
- Creating index on clustering field clustering index
- Clustering index to speed up the retrieval of data
- Clustering index is an ordered file with two fields ☒ The first field is same type as the clustering field second field is a block pointer

Clustering Index

Nondense Indexes



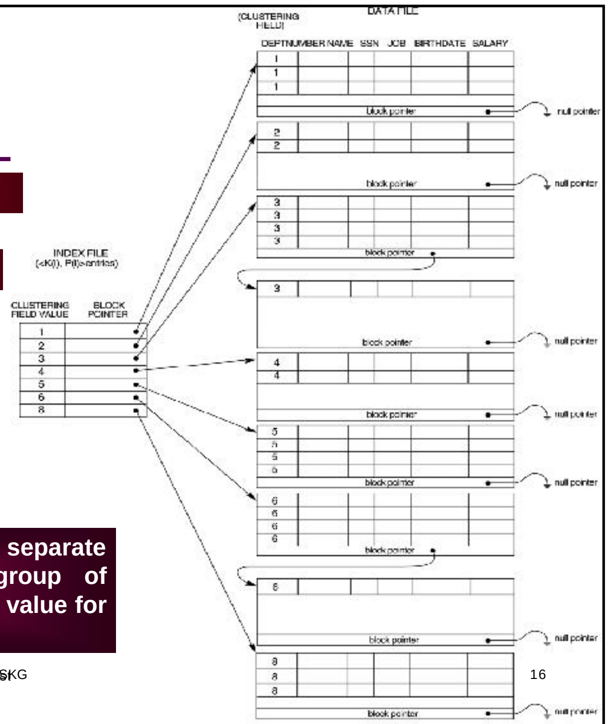
Clustering index on the DEPTNUMBER ordering non-key field of an EMPLOYEE file

@SKG 15

Clustering Index

Nondense Indexes

How to search ?



Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

9/9/2010 @SKG 16

Clustering Index

- Advantages
- Disadvantages
  - ✓ Insertion/ Deletion

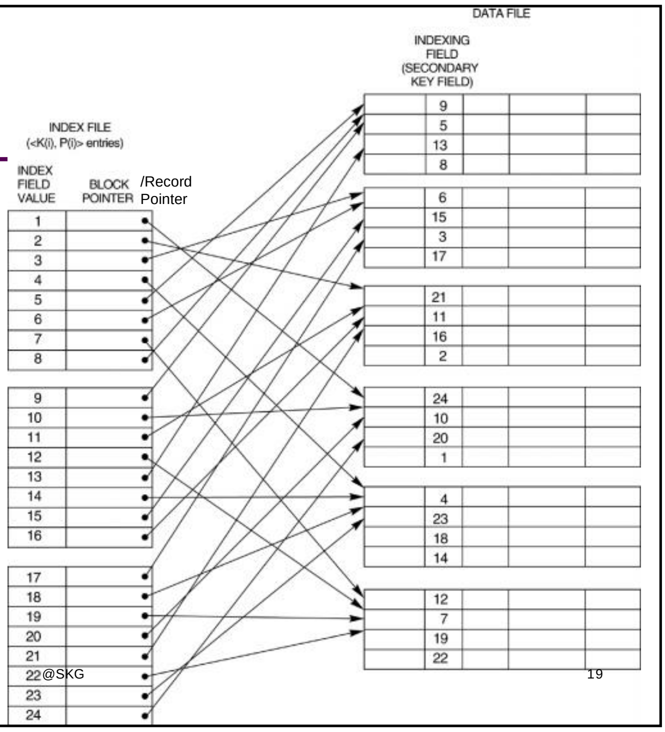
Secondary Index

- Provides a secondary means of accessing a file for which some primary access already exist
- Index on non-ordered key/ non-key field
- Index is an ordered file with two fields → The first field is same type a the non-ordering field of the data file…. second field is a block pointer/ record pointer
- Secondary index structure on a key filed
- field is called secondary key → Indexing is dense index

Secondary Index

How to search?

Dense secondary index (with block pointers) on a non-ordering key field of a file



Secondary Index: Problem

Example: Given the following data file:

EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)

Problem3: Consider problem 1 with r= 30,000 records stored on a disk with block size B= 1024 bytes. File records are of fixed size and un-spanned with record length of R=100 Bytes.

Calculate the amount of block access needed to retrieve a data from the file with linear search

Ans: Total number of block access: b/2 = 3000/2

=1500 block access on the average

Secondary Index : Problem

Problem 4: Now assume that we construct a secondary index on a non-ordering key field of the file that is 9 bytes long. The size of block pointer is 6 bytes.

Calculate-

i) The blocking factor of the index file.

ii) The total number of blocks needed to store the index file.

iii) The number of block access needed to access a data from the data file (Assume binary search scheme has been implemented as a search scheme)

Secondary Index : Solution

Example Ans.: Given the following data file:

Suppose that: key field size V=9 bytes,

Block Pointer P=6 bytes,

Then, we get: Size of each index entry: $R_i = (9+6) = 15$ bytes

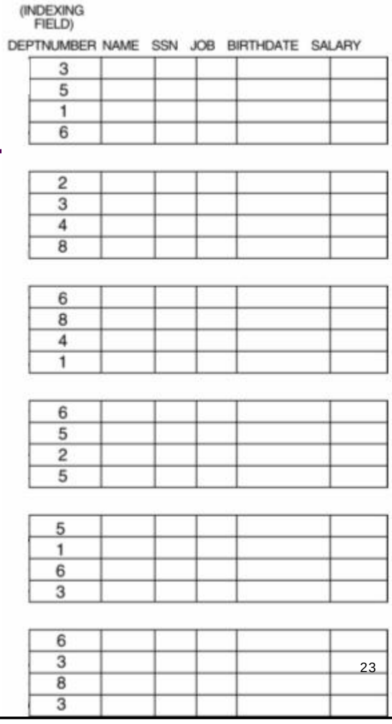
Blocking factor of the Index : $bfr_i = \left\lfloor \frac{B}{R_i} \right\rfloor = \left\lfloor \frac{1024}{15} \right\rfloor = 68$ records/block

Number of blocks to store the index: $b_i = \left\lceil \frac{r_i}{bfr_i} \right\rceil = \left\lceil \frac{30000}{68} \right\rceil = 442$ blocks

Total number of block access: $\lceil \log_2 b_i \rceil + 1 = \lceil \log_2 442 \rceil + 1 = 9 + 1 = 10$ block access

Secondary Index

Secondary indexes on a non-ordering non-key field of a file



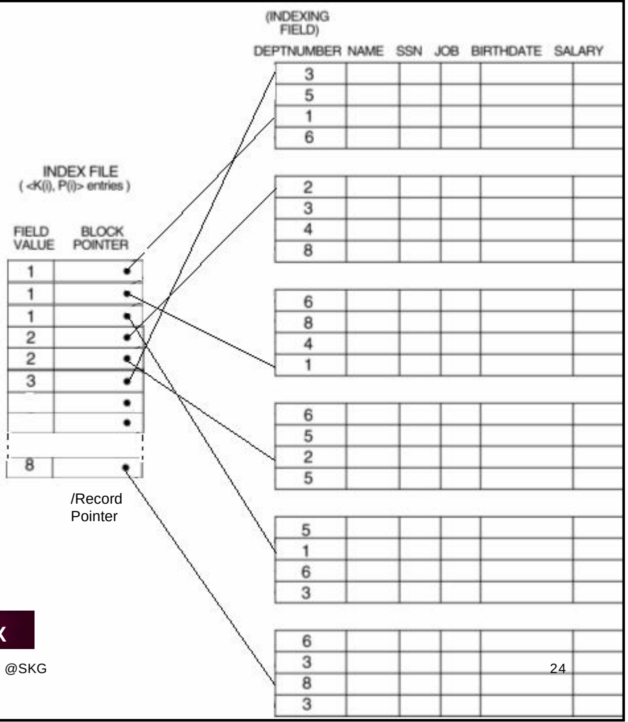
@SKG 23

Secondary Index

on a non-ordering non-key field of a file

Several options of implementing such an index:

Option1: Include several index entries with the same K (i) value



DENSE INDEX

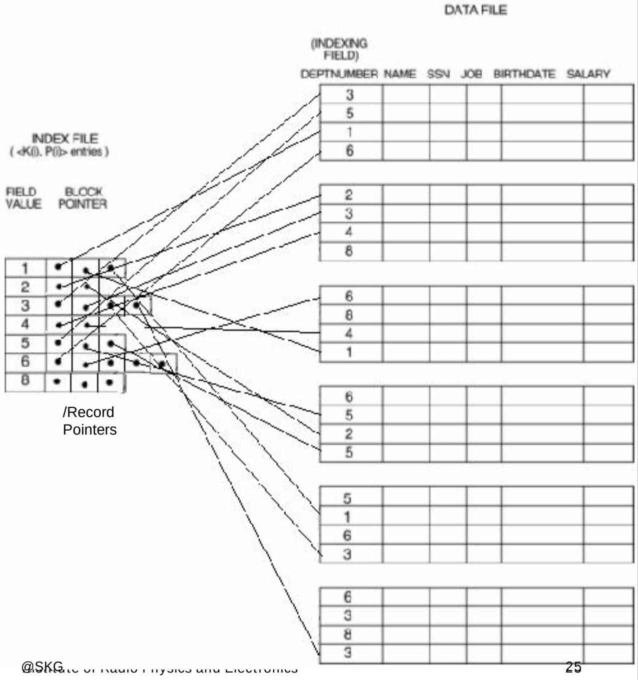
@SKG 24

Secondary Index

on a non-ordering non-key field of a file

Several options of implementing such an index:

Option2: Variable length records for the index entries with a repeating field for the pointers

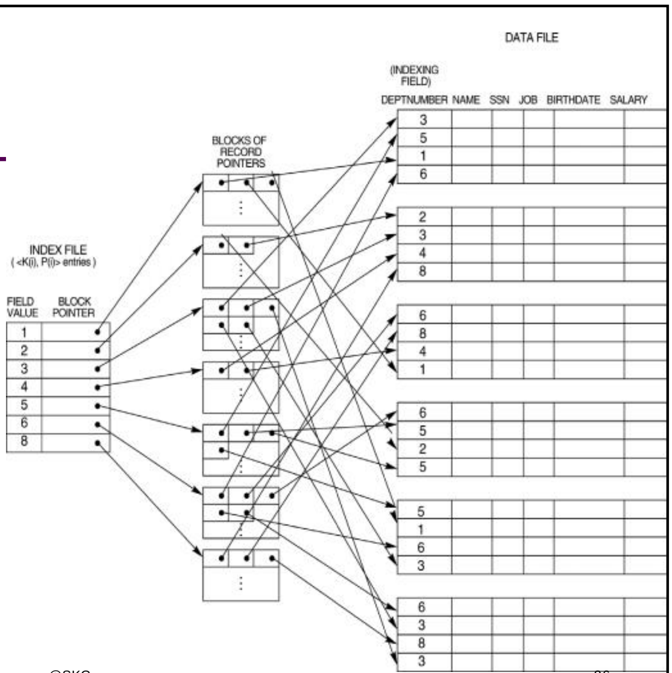


Secondary Index

on a non-ordering non-key field of a file

Several options of implementing such an index:

Option3: keep the index entries at a fixed length and have a single entry for each index field but to create a n extra level of indirection

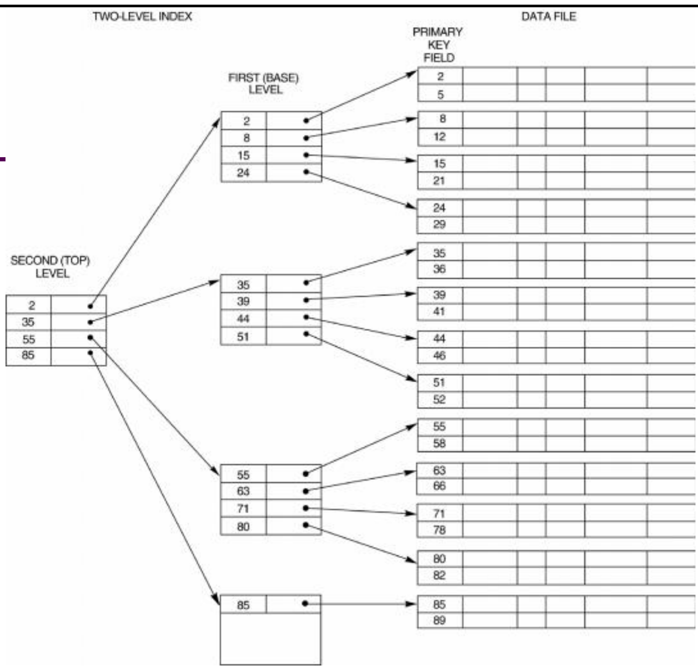


Multilevel Index

Multi- Level Indexing

- **Static**
- **Dynamic**

Static Multilevel Index



Static Multi-level Index : Problem

Problem 5: Assume that the dense secondary index of problem4 is converted into a multi-level index.

Calculate-

i) The number of levels and block access needed to access a data from the data file

Static Multi-level Index : Solution

Example: Given the following data file:

Suppose that: key field size V=9 bytes,

Block Pointer P=6 bytes,

Then, we get: Size of each index entry: $R_i = (9+6) = 15$ bytes

blocking factor of the Index : $bfr_i = \left\lfloor \frac{B}{R_i} \right\rfloor = \left\lfloor \frac{1024}{15} \right\rfloor = 68$ records/block

Number of blocks to store the index in the 1st level:

$$b_i = \left\lceil \frac{r_i}{bfr_i} \right\rceil = \left\lceil \frac{30000}{68} \right\rceil = 442 \text{ blocks}$$

Static Multi-level Index : Solution

Example: contd..

The number of 1st level block b_1= 442 blocks

The number of 2nd level block b_2 = $b_2 = \left\lceil \frac{r_2}{bfr_i} \right\rceil = \left\lceil \frac{b_1}{bfr_i} \right\rceil = \left\lceil \frac{442}{68} \right\rceil = 7$ blocks

The number of 3rd level block b_3 = $b_3 = \left\lceil \frac{r_3}{bfr_i} \right\rceil = \left\lceil \frac{b_2}{bfr_i} \right\rceil = \left\lceil \frac{7}{68} \right\rceil = 1$ blocks

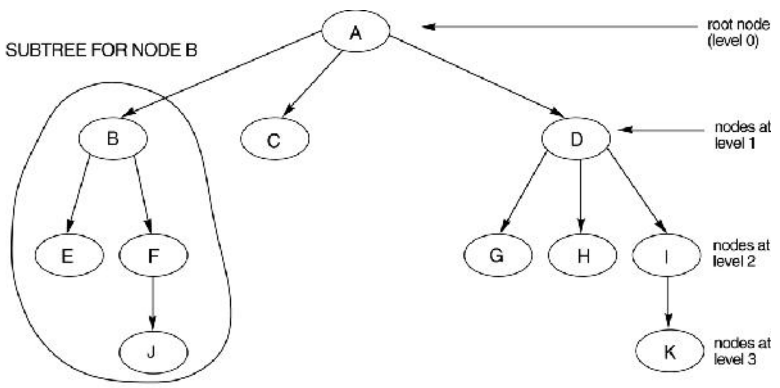
The 3rd level is the top level. So, t=3.

Total number of block access: t+1 = 3+1 = 4.

Dynamic Multi-level Indexing

Using B-tree and B+-tree

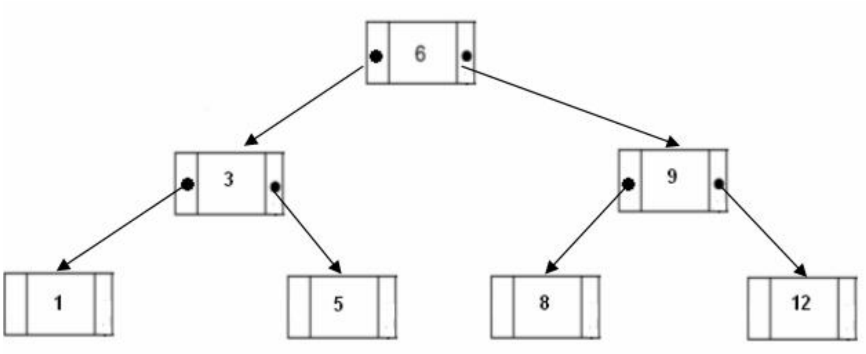
Tree structure Basic Issues



(nodes E,J,C,G,H, and K are leaf nodes of the tree)

Dynamic Multi-level Indexing

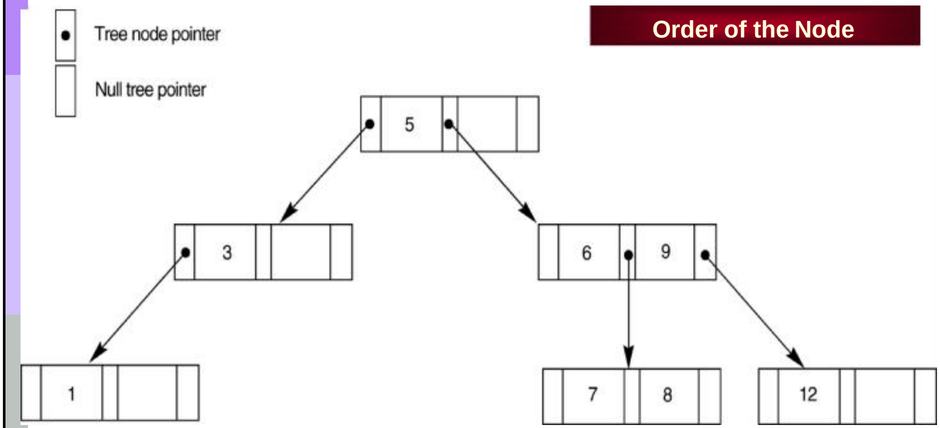
Tree structure Basic Issues



Dynamic Multi-level Indexing

Tree structure Basic Issues

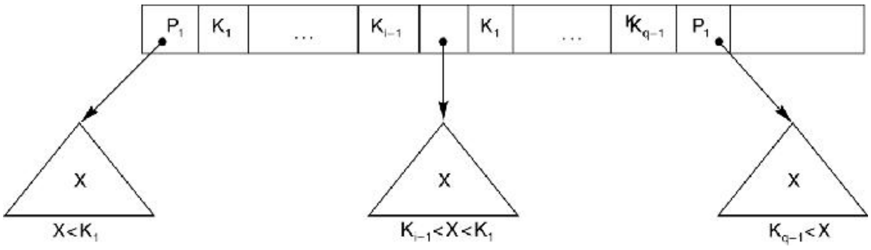
Order of the Node



Dynamic Multi-level Indexing

Tree structure Basic Issues

❖ A node in a search tree with pointers to subtrees below it

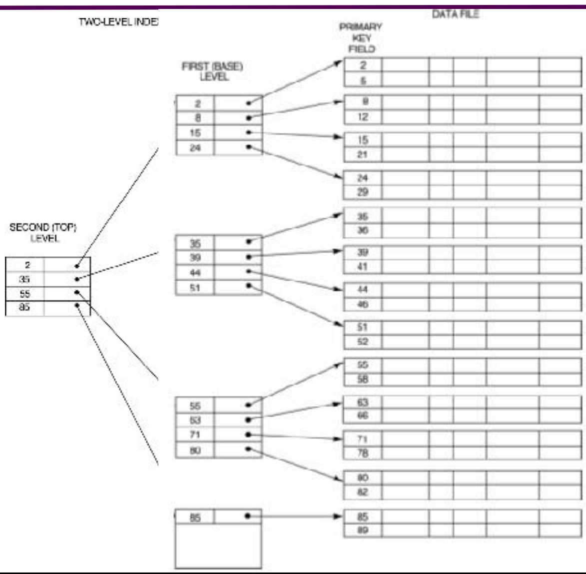


❖ Two constraints on search tree:

1. Within each node $K_1 < K_2 < . < K_{q-1}$
2. For all values X in the subtree pointed at by P_i we have $K_{i-1} < X < K_i$ for 1<i<q; $X < K_i$ for i=1; and $K_{i-1} < X$ for i=q.

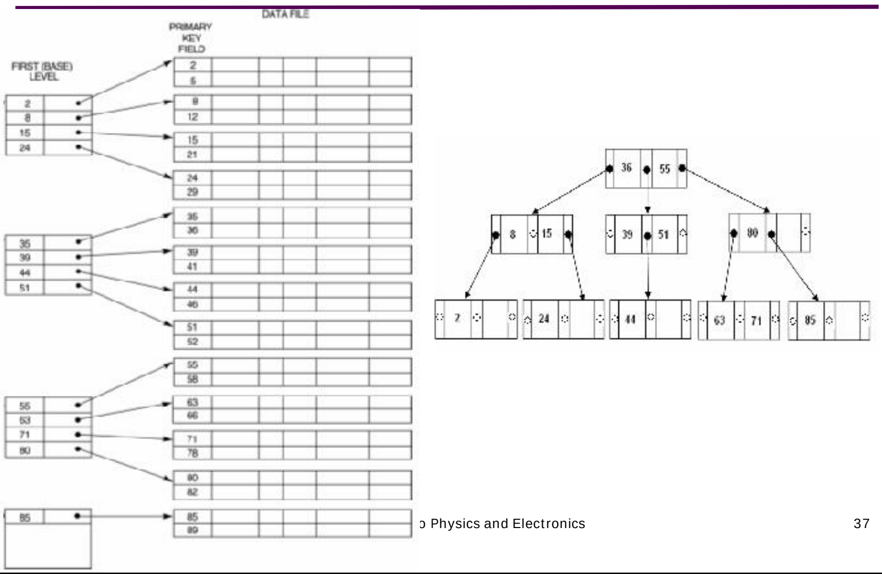
Dynamic Multi-level Indexing

Tree structure Basic Issues



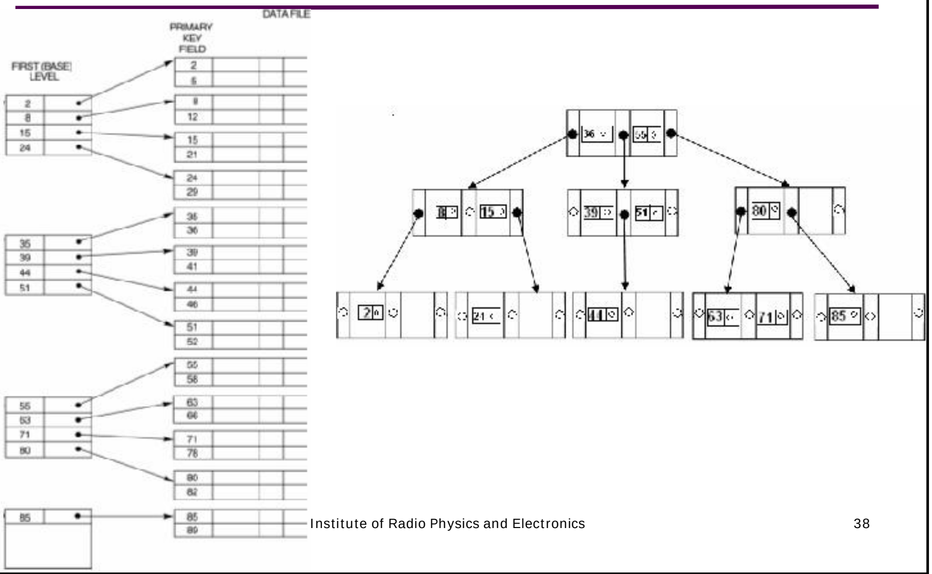
Dynamic Multi-level Indexing

Tree structure Basic Issues



Dynamic Multi-level Indexing

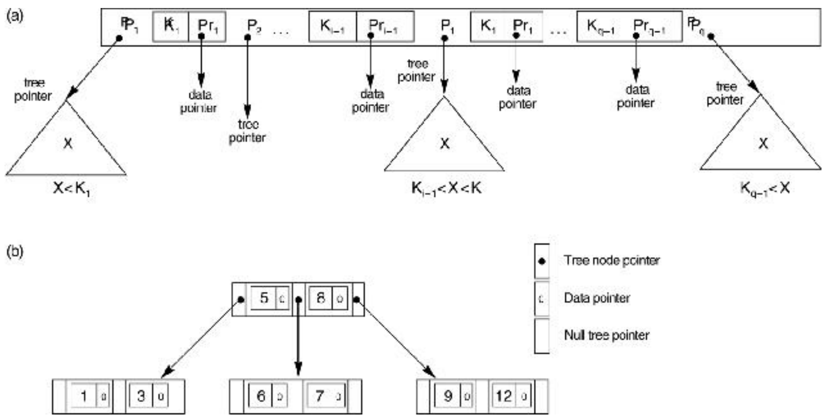
Tree structure Basic Issues



Dynamic Multi-level Indexing

B-Tree

B-tree structures. (a) A node of order p in a B-tree with q 1search values. (b) A B-tree of order p = 3. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



Dynamic Multi-level Indexing

B-Tree

Problem 6: Assume that in a data file search field is 9 byte long and the disk block size is 512 bytes. A record pointer size is of 7 bytes long and that of a block pointer is of 6 bytes long.

Calculate

- i) the maximum order of the each node of the B-tree to fit a single block into each B-tree node

ii) Tabulate the number of nodes, entries and pointers present in the B-tree up to Level 3 of the tree, assuming that each node of the B-tree is 69% full.

Dynamic Multi-level Indexing

B-Tree

Solution i):

Given, search field V=9 bytes, disk block size B=512 bytes, record pointer P_r= 7 bytes, and block pointer P= 6 bytes.

Let us assume each node can have maximum p- tree pointers, (p-1) data pointers, and (p-1) search key field values.

Then,

$$(p * P) + ((p-1) * (P_r + V)) < B$$

$$(p * 6) + ((p-1) * (7+9)) \leq 512$$

$$22 * p \leq 528$$

$$p = 24$$

@SKG

41

Dynamic Multi-level Indexing

B-Tree

Solution:

Since, each node is 69% full, we have

$p * 0.69 = 24 * 0.69$ 16.56 ≈16   tree pointers in the node

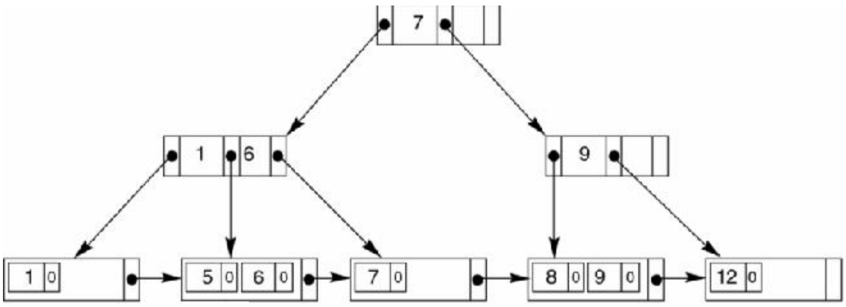
	Nodes	Entries	Tree pointers
Root	1 node	15 entries	16 pointers
Level1	16 nodes	240 entries	256 pointers
Level2	256 nodes	3840 entries	4096 pointers
Level3	4096 nodes	**61440 entries**	
		65535 entries	

@SKG

42

B+ Tree Structure

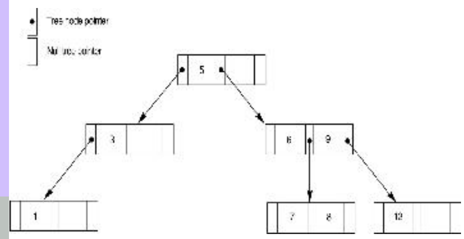
eid: {1,5,6,7,8,9,12}

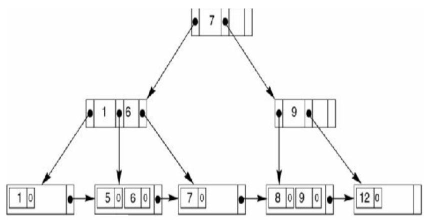


Dynamic Multi-level Indexing

Tree structure: Basic issue

Order of the Node

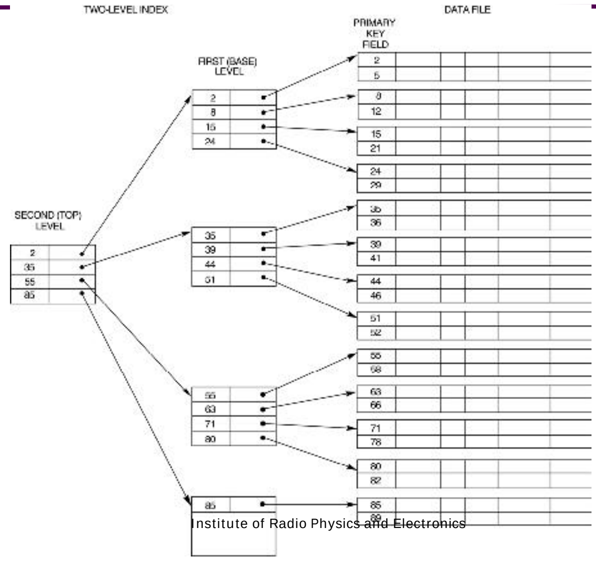




@SKG 44

Dynamic Multi-level Indexing

Tree structure: Basic Issues

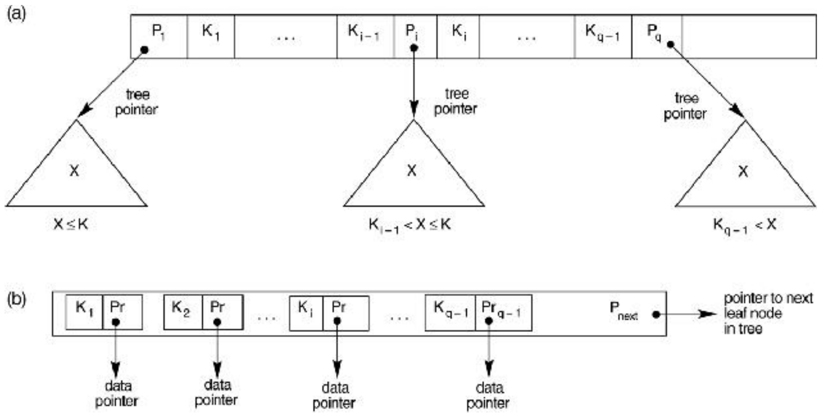


Institute of Radio Physics and Electronics

Dynamic Multi-level Indexing

B^+-Tree

The nodes of a B^+-tree. (a) Internal node of a B^+-tree with q 1search values. (b) Leaf node of a B^+-tree with q 1data pointers.



Dynamic Multi-level Indexing

B^+- Tree

Problem 7: Assume that in a data file search field is 9 byte long and the disk block size is 512 bytes. A record pointer size is of 7 bytes long and that of a block pointer is of 6 bytes long.

Calculate

- i) the maximum order of the each node of the B^+-tree to fit a single block into each B^+-tree node

ii) Tabulate the number of nodes, entries and pointers present in the B^+-tree up to Level 3 of the tree, assuming that each node of the B^+-tree is 69% full.

@SKG 47

Dynamic Multi-level Indexing

B^+- Tree

Solution i):

Given, search field V=9 bytes, disk block size B=512 bytes, record pointer P_r= 7 bytes, and block pointer P= 6 bytes.

Let us assume each internal node can have maximum p- tree pointers, and (p-1) search values. Then,

$$(p * P) + ((p-1) * V) \leq B \qquad (p * 6) + ((p-1) * 9) \leq 512$$

$$15 * p \leq 521 \qquad p = 34$$

Order of leaf node is , $(P_{leaf})(P_r + V)) + P \leq B$

$$(P_{leaf})(7 + 9)) + 6 \leq 512$$

$$(P_{leaf}) \leq 31 \text{ key value/data pointer}$$

@SKG 48

Dynamic Multi-level Indexing

B+- Tree

Solution:

Since, each node is 69% full, we have

$p * 0.69 = 34 * 0.69 \quad 23.46 \quad 23$ tree pointers in the node

$P_{leaf} * 0.69 = 31 * 0.69 \quad 21.39 \quad 21$ search values in the leaf node

	Nodes	Entries	Tree pointers
Root	1 node	22 entries	23 pointers
Level1	23 nodes	506 entries	529 pointers
Level2	529 nodes	11638 entries	12167 pointers
Level3	12167 nodes	**255507 entries**	

@SKG 49

Thank You

?

@SKG 50