CS522 Advanced Database Systems

Indexes

Chengyu Sun
California State University, Los Angeles

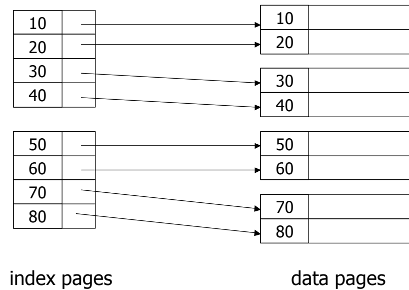
Outline

- Overview
- B-tree
- Hash indexes
- Multidimensional indexes

Indexes

- Auxiliary structures that speed up operations that are not supported *efficiently* by the basic file organization

A Simple Index Example



Index Issues

- Search key
- Index pages
  - What is stored in an **entry**?
  - How are index entries organized?
- One entry per record??
  - Dense or sparse
- Are the records sorted, hashed??
  - Clustered or un-clustered
  - Primary or secondary

Entries in an Index

- Actual data record
- <key,rid>
- <key, list of rid>

Organization of Index Entries

- Tree-structured
  - B-tree, R-tree, Quad-tree, kd-tree, ...
- Hash-based
  - Static, dynamic
- Other
  - Bitmap, signature, VA-file, ...

Clustered or Un-clustered

- Clustered index – records are sorted in the same order as the index entries
- Primary and secondary index
  - Stanford book: *primary = clustered*
  - Wisconsin book: index on primary key

Dense or Sparse

- Dense index – one entry per record
- Sparse – one entry per page
- Comparison??

Files and Indexes

- A primary index must be clustered??
- A secondary index cannot be clustered??
- We can build a clustered index on top of a hashed file??
- We can build a dense index on top of a sorted file??
- We can build a sparse index on top of a heap file??

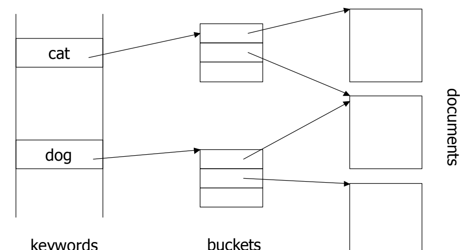
Managing Indexes During Data Modification

- Assume clustered index

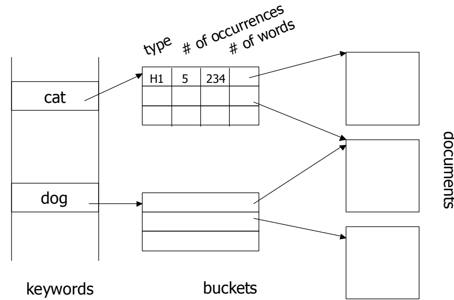
Action	Dense	Sparse
create *empty* overflow page		
delete *empty* overflow page		
create *empty* sequential page		
delete *empty* sequential page		
insert record		
delete record		
slide record		

Inverted Index

- Ever wondering how search engines work?



A Better Inverted Index



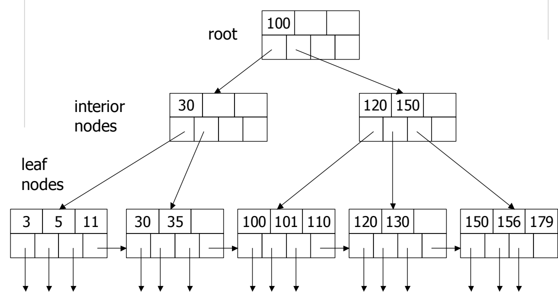
Outline

- ◈Overview
- ◈B-tree
- ◈Hash indexes
- ◈Multidimensional indexes

From BST to BBST to B

- ◈Binary Search Tree
  - Worst case??
- ◈Balanced Binary Search Tree
  - AVL, Red-Black, ...
- ◈B-tree
  - Why not just use BBST in databases??

B+ Tree Example



B+ Tree Properties

- ◈Order n
  - n keys, n+1 pointers
- ◈Occupancy – *at least half full*
  - Non-leaf: $\lceil (n+1)/2 \rceil$ pointers
  - Leaf: $\lfloor (n+1)/2 \rfloor$ pointers
  - Except ROOT
- ◈Balance
  - *All leaf nodes on the same level*

B+ Tree Insert

- ◈Find the appropriate leaf
- ◈Insert into the leaf
  - there's room → we're done
  - no room
    - split leaf node into two
    - insert a new <key,pointer> pair into leaf's parent node
- ◈ *Recursively apply previous step if necessary*
  - A split of current ROOT leads to a new ROOT

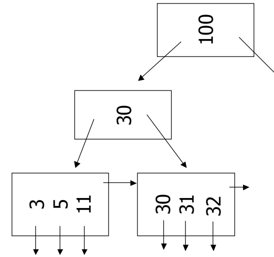
B+ Tree Insert Examples

- (a) simple case
  - space available in leaf
- (b) leaf overflow
- (c) non-leaf overflow
- (c) new root

HGM Notes

(a) Insert key = 32

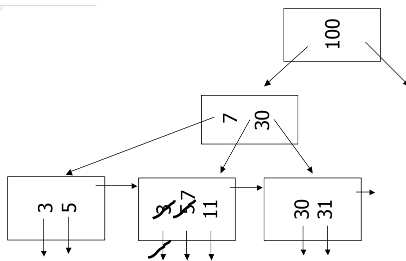
n=3



HGM Notes

(a) Insert key = 7

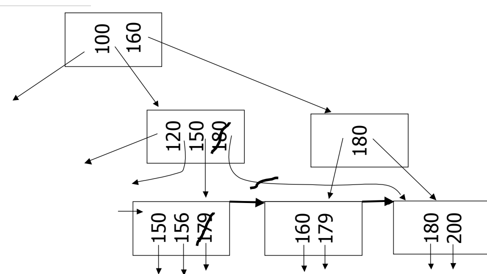
n=3



HGM Notes

(c) Insert key = 160

n=3

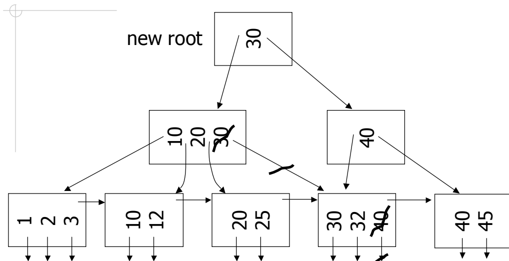


HGM Notes

(d) New root, insert 45

n=3

new root



HGM Notes

B+ Tree Delete

- Find the appropriate leaf
- Delete from the leaf
  - still at least half full → we're done
  - below half full
    - borrow a <key,pointer> from one sibling node, *or*
    - merge with a sibling node, and delete from a parent node
- *Recursively apply previous step if necessary*
  - When do we need a new ROOT (or decrease the height of the tree)??

B+ Tree Delete Examples

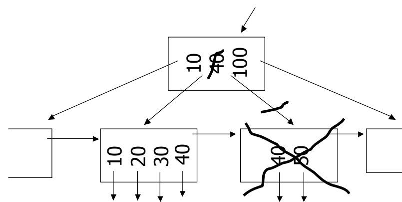
- (a) simple case
  - no example
- (b) leaf overflow
- (c) non-leaf overflow
- (d) new root

HGM Notes

(b) Coalesce with sibling

- Delete 50

n=4

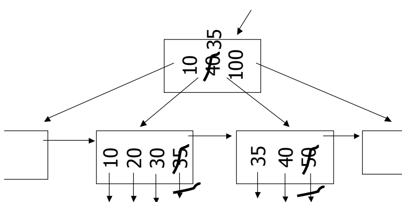


HGM Notes

(c) Redistribute keys

- Delete 50

n=4

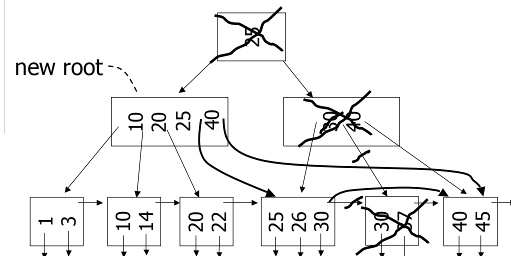


HGM Notes

(d) Non-leaf coalesce

- Delete 37

n=4



HGM Notes

B+tree deletions in practice

– Often, coalescing is not implemented

- Too hard and not worth it!

HGM Notes

B+ Tree Implementation

- Search
- Insert
- Delete

Implementation Issues

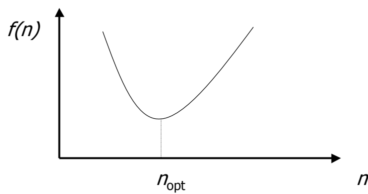
- Node size
  - How many entries do we want to put in a node?
- Buffering
  - Is LRU a good strategy for buffering B+ tree?
- Duplicates
  - What changes need to be made to the *insert*, *delete*, and *search* algorithms?

Calculate n_{opt} – Assumptions

- Time to read from disk
  - (70+0.05n) ms
- Once the page in memory, do binary search to locate the key
  - $(a+b\log_2 n)$ ms
- Assume B+ tree is full, $\log_n N$ nodes need to examined.

Caculate n_{opt}

- f'(n) = 0
  - n is a few hundreds

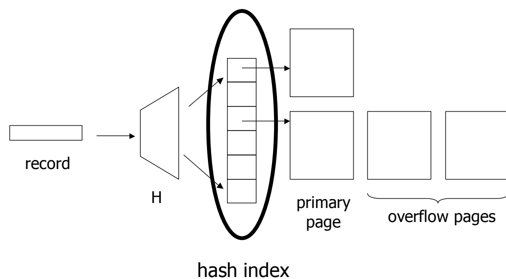


HGM Notes

Outline

- Overview
- B-tree
- Hash indexes
- Multidimensional indexes

Static Hashing



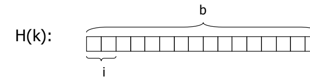
Hash Function

- A commonly used hash function: K%B
  - *K* is the key value
  - *B* is the number of buckets
    - prime, or
    - 2^n
- String??

Dynamic Hashing

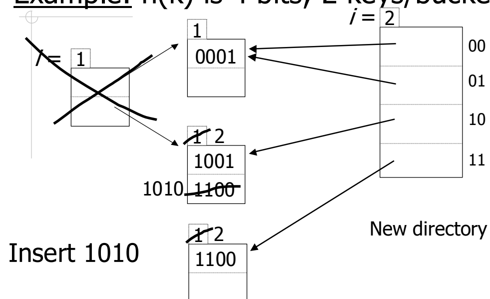
- ◈Problem of static hashing??
- ◈Dynamic hashing
  - Extensible hashing
  - Linear hashing

Extensible Hashing



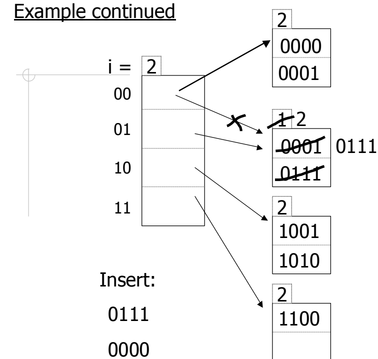
- ◈b-bit hash value, only **i** bits used, and i <= b
  - 2^i buckets
- ◈When insert into a page that is already full
  - Split the page into two
  - Increment i
  - Double the number of buckets

Example: h(k) is 4 bits; 2 keys/bucket



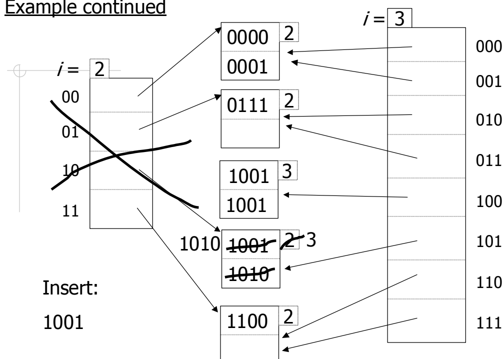
HGM Notes

Example continued



HGM Notes

Example continued

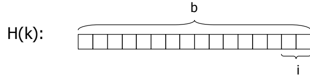


HGM Notes

Efficiency of Extensible Hashing

- ◈No overflow block
- ◈Guaranteed single I/O access
- ◈*Problems*??

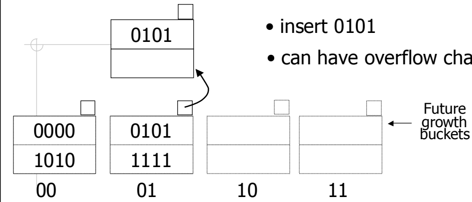
Linear Hashing



- Start with n buckets, i = $\lceil \log_2 n \rceil$
- Insert: let last i bits of H(k) be m
  - m < n, insert into bucket m
  - m >= n, insert into bucket $m-2^{i-1}$
- Add one bucket when *on average* each page is more than 80% full

Example *b*=4 bits, *i* =2, 2 keys/bucket

- insert 0101
- can have overflow chains!



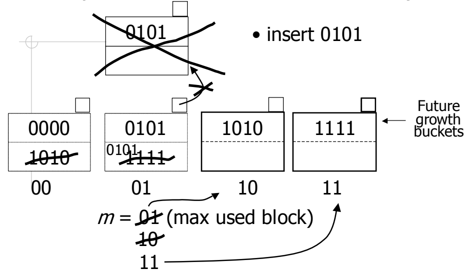
m = 01 (max used block)

Rule: If h(k)[*i*] ≤ *m*, then look at bucket h(k)[i] else, look at bucket h(k)[*i*] - 2^{i-1}

HGM Notes

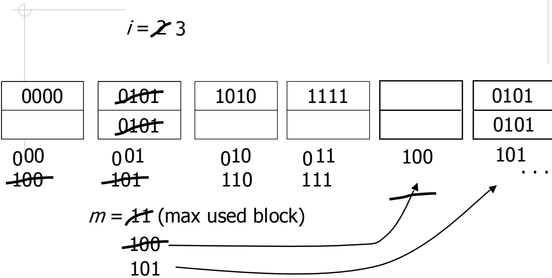
Example *b*=4 bits, *i* =2, 2 keys/bucket

- insert 0101



HGM Notes

Example Continued: How to grow beyond this?



HGM Notes

About Linear Hashing

- Advantages??
- Overflow pages??
- Why 80%??

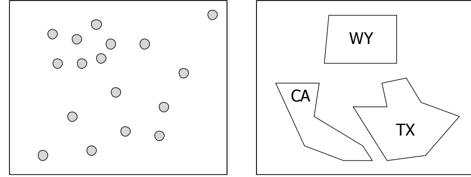
Outline

- Overview
- B-tree
- Hash indexes
- Multidimensional indexes

Concept of Dimension

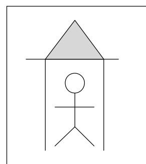
- Dimensions in real life
  - 3D
  - 4D if you're a physicist
- Dimensions in databases
  - Each attribute is considered as a dimension
  - Number of dimensions (*dimensionality*) could be very high

GIS Data



- Attributes correspond to physical dimensions
  - e.g. longitude and latitude

Multimedia Data

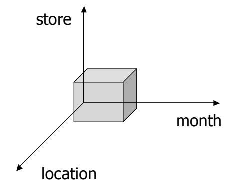


feature vector
[73,14,0,99,....,]

- Dimensions in a mathematical sense

Commercial Data

store	location	month	sales
1	PA	1	1000
2	AH	1	900
1	PA	2	1500
3	SG	3	890



- "Cubing" for aggregation queries
  - e.g. total sales in January, or total sales of San Gabriel stores in months from January to May

Queries on Multidimensional Data

- Data
  - points or *regions* in a multidimensional space
- Queries
  - Partial match queries
  - Range queries
  - Nearest-neighbor queries
  - Point-in-shape queries

Example 1: Assumptions

- 1,000,000 points, uniform distribution
- x-coordinate range: [0,1000]
- y-coordinate range: [0,1000]
- 100 entries per B-tree leaf node

- Query: find points where 450<=x<=550 and 450<=y<=550

Example 1: Two Indexes

- Search x-index: 1000 leaf node accesses
- Search y-index: 1000 leaf node accesses

- To find 1% of the data, we need to access 10% of leaf nodes, twice
- Plus an intersection of two sets of pointers, which may or may not fit in memory

Example 1: One Multi-attribute Index

- Would it help??
- Problems??

Example 2

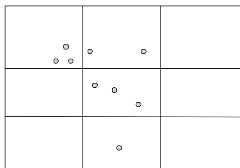
- *Nearest-neighbor query*: can we do it with a B-tree?

Partitioned Hashing

- Given n attributes (A_1,A_2,...,A_n)
- Use n hash functions H_1, H_2,..., H_n
- Apply H_i to A_i and concatenate the result

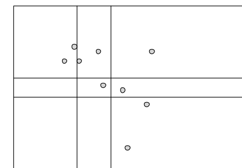
$$H(K) = \underbrace{H_1(A_1)}_{k\text{ bits}}\ H_2(A_2) \ldots \underbrace{H_i(A_i)}_{k_i\text{ bits}} \ldots H_n(A_n)$$

Grid File – Static



- Good for what type of queries??
- Problems??

Grid File – Dynamic

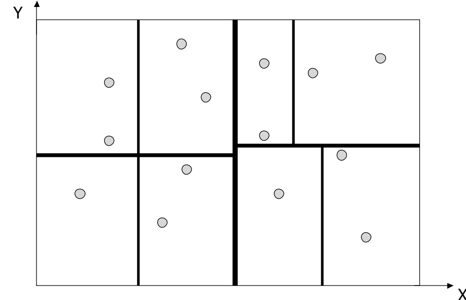


- Split – which dimension?? which position??
- Problems??

kd-Tree

- A binary tree
- At each node, the space is partitioned into two along certain dimension
- Alternate dimensions between levels

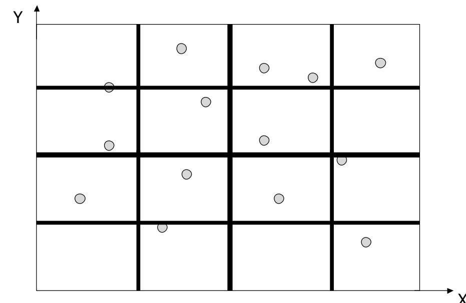
kd-Tree Example



kd-Tree Issues

- What if there's a "un-splitable" dimension??
- Unbalance??
- From main memory to secondary storage??

Quad-Tree Example

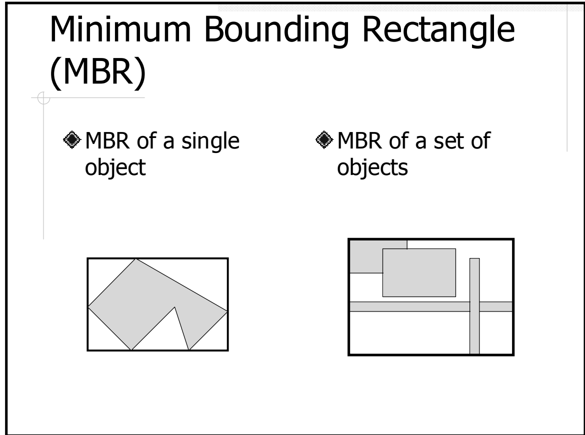


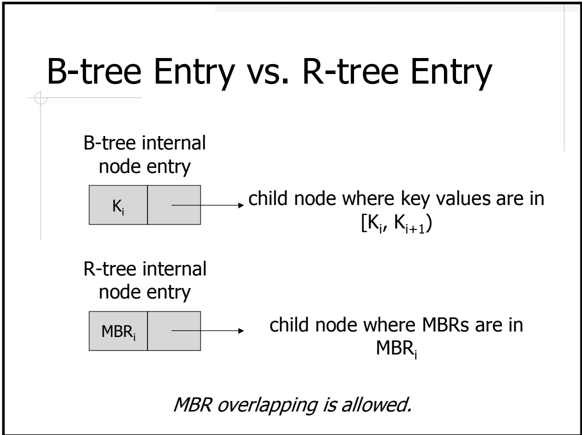
R-tree Insert

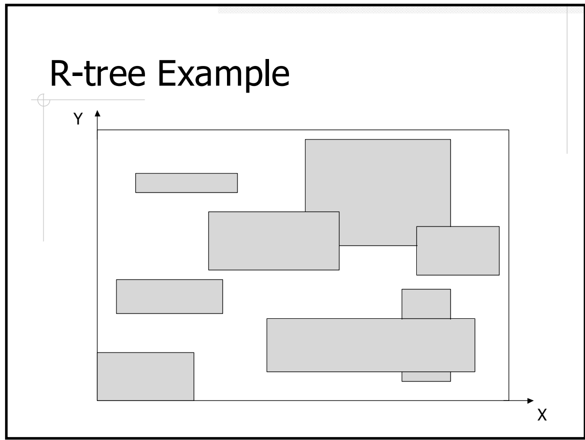
- Which subtree to choose??
- How to split a full node??

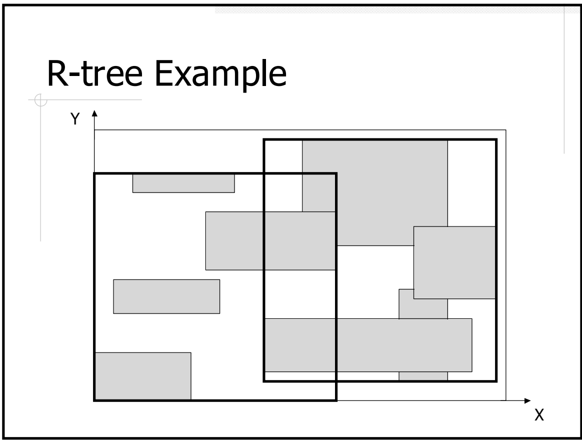
R-tree

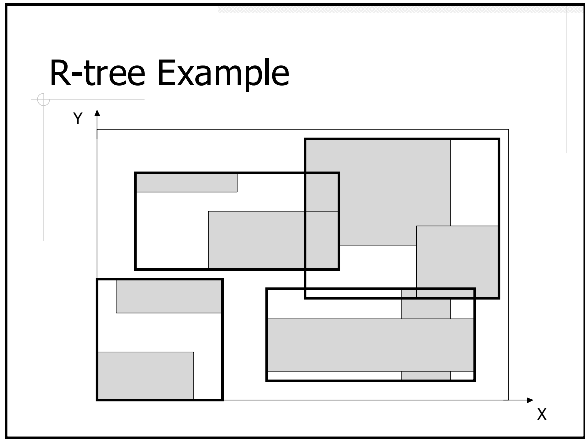
- A B-tree for multidimensional data
- Works naturally for points, segments, and regions

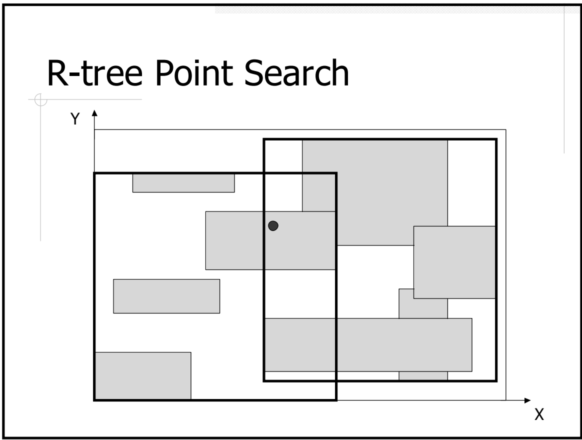

Minimum Bounding Rectangle (MBR)
◈MBR of a single object
◈MBR of a set of objects


B-tree Entry vs. R-tree Entry
B-tree internal node entry
K_i
child node where key values are in [Ki, Ki+1)
R-tree internal node entry
MBRi
child node where MBRs are in MBRi
MBR overlapping is allowed.

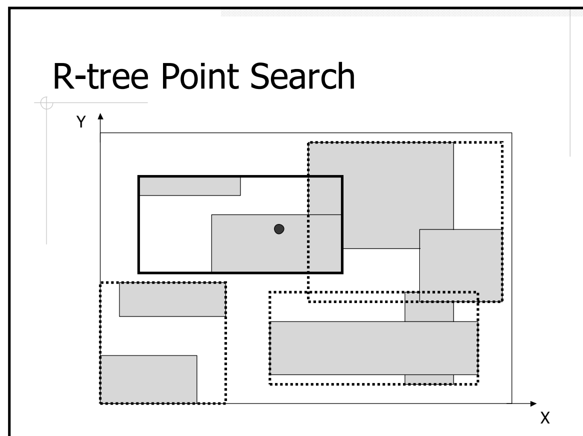

R-tree Example
Y
X


R-tree Example
Y
X

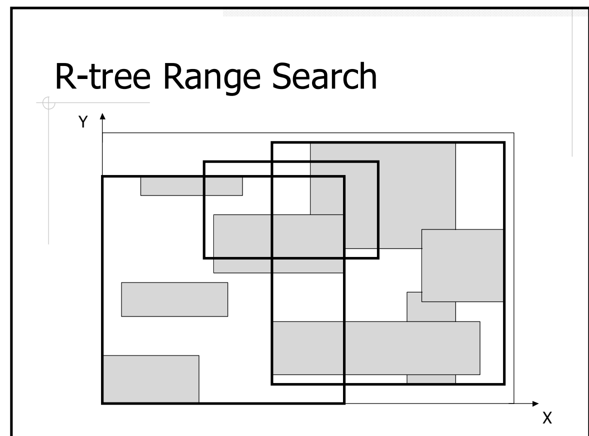

R-tree Example
Y
X


R-tree Point Search
Y
X

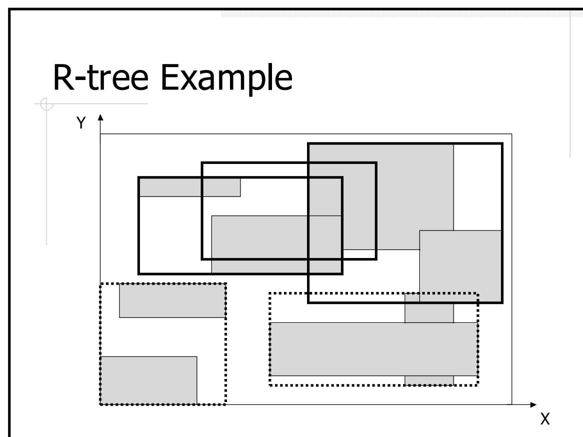
R-tree Point Search



R-tree Range Search



R-tree Example



Some Observations So Far

- ◈Arrange entries by spatial locality
- ◈MBR overlapping may leads to searching multiple subtrees
  - Queue vs. Stack

R-tree Insert

- ◈Which subtree to choose??
- ◈How to split a full node??

R-tree Distance Query Optimization

- ◈Minmax distance
- ◈Maxmin distance

Bitmap Index

- Assume n records, m unique key values
- A bitmap index consists of m bit-vectors of length n such that
  - Each bit-vector corresponds to a unique key value
  - Each bit corresponds to a record
  - A bit is 1 if the record has the key value, and 0 otherwise

Bitmap Index Example

Data:
6 records
3 distinct values

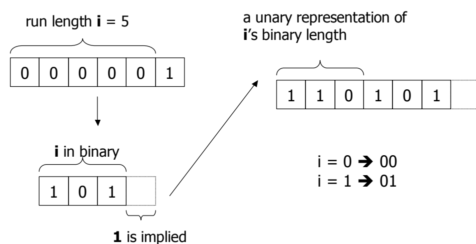
('john', 'A')
('smith', 'B')
('joe', 'A')
('sue', 'C')
('lisa', 'A')
('jen', 'B')

Index:

'A':	1	0	1	0	1	0
'B':	0	1	0	0	0	1
'C':	0	0	0	1	0	0

Compress Bitmaps

- Run-length encoding



Bitmap Compression Examples

100000001000 —encode→ ??

11101101001011 —decode→ ??

Other Bitmap Index Issues

- Given value or value range, find bit vectors??
- Retrieve kth record efficiently??
- Insert and delete??

Why Bitmap Index

- Bitmap performs well for what types of queries??

Readings

- Stanford book: Chapter 13, 14
- [Wisconsin book: Chapter 9, 10]
- [Kleinberg paper]
- [VA-file paper]