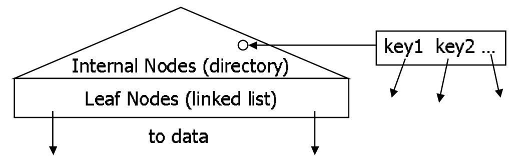
"Generalized Search Trees for Database Systems"

by Joseph M. Hellerstein, Jeffrey F. Naughton,
and Avi Pfeiffer

Andy Danner
Sara Sprenkle
CPS216
October 9, 2001

Generalizing DB Search Trees

- Balanced tree
- High fanout
- Keys → predicates, may overlap



Motivation

- Extensible data and query models
- Ease construction of index structures for new data and query types
  - adding new data types
- Generalized tree structure for database systems
  - maintaining data and asking queries

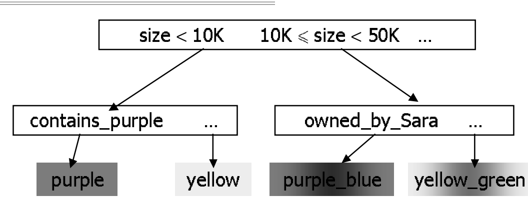
Generalizing DB Search Trees

- Generalized search key
  - any arbitrary predicate that holds for each datum below the key
  - flexible→arbitrary nested subcategories
- Database search tree:
  - "hierarchy of partitions of a data set, in which each partition has a categorization that holds for all data in the partition"

Outline

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

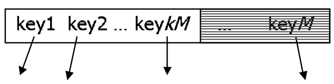
Running Example: Images



Warning: This index scheme may not be suitable for indexing real images.

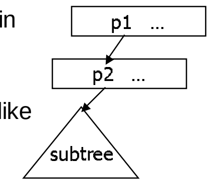
Properties

- Internal nodes (other than the root) have between *kM* and *M* index entries
  - *k*: minimum fill factor, < ½
  - **entry**: (key/predicate, pointer)



Properties

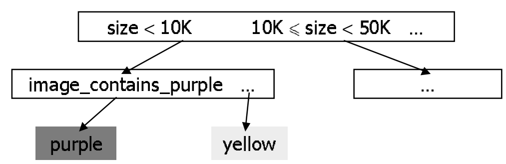
- Internal predicates evaluate to true for all data instances in subtree
- May not have hierarchical satisfiability of predicates like R-trees



Properties

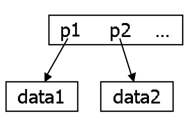
- Internal nodes (other than the root) have between *kM* and *M* index entries
- Root has at least two children unless it is a leaf node
- All leaf nodes appear on same level
- Balance tree
- Bound height of tree

Images



Properties

- Leaf node predicates evaluate to true given values of data instance



Key Methods

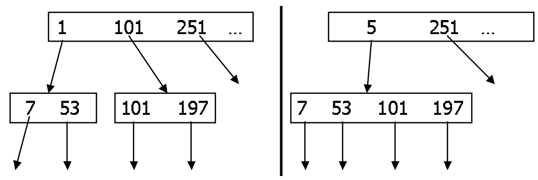
- Methods used by GiST to maintain invariants
- Implemented by index developer
- Application-specific policies

Key Methods

- Consistent(Entry E, Predicate q)
- Union(Entry E[])
- Compress(Entry E)
- Decompress(Entry E)
- Penalty(Entry E1, Entry E2)
- PickSplit(Entry E[])

Key Methods

- Union(Entry E[])
  - predicates from a set of entries are merged into one predicate



Key Methods

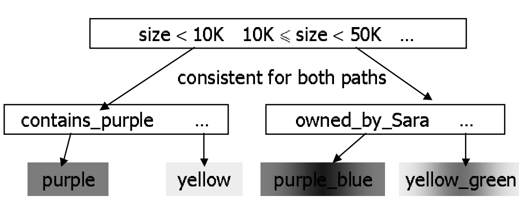
- Recall: **Entry** is (predicate p, pointer ptr)
- Consistent(Entry E, Predicate q)
  - returns false if p AND q are guaranteed unsatisfiable
  - determines which tree(s) to search
  - false positives but no false negatives

Key Methods

- Compress(Entry E)
  - compressed representation of predicate p

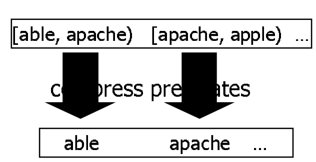
Image Search

Query: Image files that contain purple



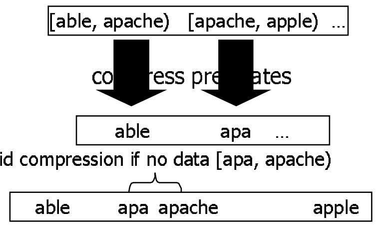
B+-tree Compress

lossless compression, from paper



B+-tree Compress #2

lossless compression

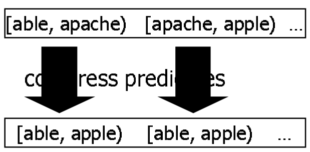


Key Methods

- Penalty(Entry E1, Entry E2)
  - penalty for inserting E2 into E1's subtree
    - local not global penalty
  - used for deciding where to insert entries or where to split a predicate
  - R-tree examples → minimizing increased area, minimizing overlap, minimizing perimeter

B+-tree Compress #3

lossy compression



Key Methods

- PickSplit(Entry E[])
  - splits set of entries E into two sets of entries, each with ~*kM* entries
  - may or may not use badness metric (e.g., multi-way penalty) to determine how to split entries

Key Methods

- Decompress(Entry E)
  - π = compressed(p)
  - r = uncompress(π), p→r
  - potentially lossy
    - do not require p iff r

Outline

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

Algorithms

- Searches, inserts, and deletes are based on the implemented key methods
  - generic algorithms for updating and accessing index structures
  - application-specific information is extracted into key methods
- Algorithms are handled by GiST, not defined by user

Algorithm: Insert

- Insert(GiST R, Entry E)
  - start at root
  - find leaf where E should be inserted
    - may require choosing among several different subtrees at each level along path
  - insert E
  - may require splitting leaf node and propagating/adjusting keys up the tree

Algorithm: Search

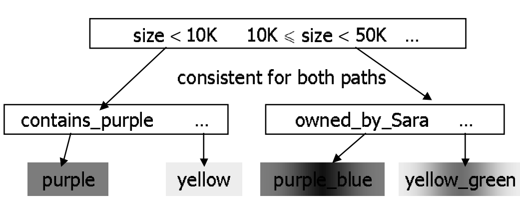
- Search(GiST R, Predicate q)
  - start at root
  - go down path or paths where key predicates are consistent q
  - reach leaf → final consistency check
  - return array of objects or array of object pointers
- Only use Consistent key method
- Generalization → exact match, range queries

Algorithm: Choose Subtree

- Calculate penalty of inserting entry in subtree
  - domain-specific penalty
  - minimize penalty locally not globally

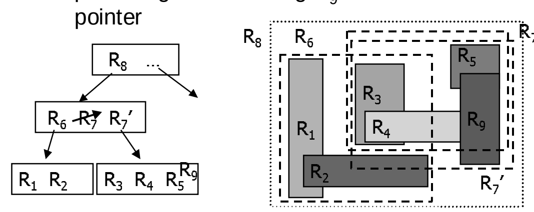
Image Search

Query: Image files that contain purple



R-tree Insert

- Insert R_9 into R-tree
  - pick a region containing R_9 and follow the child pointer



Algorithm: Split

- Union on new elements → create a new key
- Modify old key → reduce overlap, tighter control
- Adjust keys up touched path

Algorithm: Delete

- Delete(GiST G, Predicate q)
  - find element based on q
    - constrain query to return one element
  - delete
  - maintain balance, invariants up tree

Outline

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

Applications

- GiST confines application specific code to six key methods
- Implementing a new tree only requires coding of key methods. GiST handles insert, delete and search
- Paper discusses B+, R and RD Tree implementations

Application: B+-tree

- Contains([x, y), v)
  - If $x \leq v < y$, return true; otherwise, return false
- Equal(x, v)
  - If $x = v$, return true; otherwise, return false

Application: B+-tree

- Consistent(E, q)
  - If p=Contains($[x_p,y_p)$,v) AND q=Contains($[x_q,y_q)$,v), return true if ($x_p<y_q$) AND ($y_p>x_q$), false otherwise
  - If p=Contains($[x_p,y_p)$,v) AND q=Equal(x_q,v), return true if $x_p \leq x_q < y_p$, false otherwise
- Union($\{E_1, \ldots, E_n\}$)
  - $E_i=([x_i,y_i), ptr_i)$
  - return [Min($x_1, \ldots, x_n$), Max($y_1, \ldots, y_n$))

Application: B+-tree

- Compress(E=([x, y), ptr))
  - Return x, unless E is the leftmost key on an internal node (return a 0-byte object)
- Decompress(E=(π, ptr))
  - Construct an interval [x, y)
  - If E is leftmost key in internal node, $x = -\infty$; otherwise, $x = \pi$
  - If E is rightmost key in internal node, $y = \infty$; otherwise, y = nextKey();

Outline

- Motivation
- Generalized Search Trees (GiST)
- Algorithms
- Applications
- GiST Limitations, Extensions
- Conclusions

Application: B+-tree

- Penalty(E = ([x_1,y_1), ptr_1), F = ([x_2,y_2), ptr_2))
  - If E is leftmost pointer on its node, return Max($y_2 - y_1$, 0)
  - If E is rightmost pointer on its node, return Max($x_1 - x_2$, 0)
  - Otherwise, return Max($y_2 - y_1$, 0) + Max($x_1 - x_2$, 0)

GiST Limitations/Extensions

- Aggregate queries
- Nearest-neighbor, i.e., “like” queries
  - both addressed in “Generalizing ‘Search’ in Generalized Search Trees”, ICDE 1999
- Concurrency, recovery implementation
  - naïve: strict 2PL
  - addressed in “Concurrency and Recovery in Generalized Search Trees”, SIGMOD 1997

Application: B+-tree

- PickSplit(P)
  - P = { E_1, …, E_n }
  - $E_i < E_j$ for $i < j$
  - Return P_1 = { E_1, …, $E_{floor(n/2)}$ } and P_2 = { $E_{ceiling(n/2)}$, …, E_n }
  - Guarantees a minimum fill factor of *M*/2

GiST Conclusions

- Identify the fundamentals of search trees
- One ADT describes many search trees, e.g. B+-tree, R-tree, etc.
- Allows extensible data and query types

Discussion

- Questions?
- Time for quiz!

Quiz

- Define a GiST. What are its primary benefits?
- Would you use a GiST to implement a new DB search tree? Specifically, consider ease of implementing your tree. What are the tradeoffs?
- What is Sara's favorite color?
- By how much did this presentation improve your understanding of GiST?

  Scale: [1. more confused than ever, 5. damn-near an expert]